Terraform with Azure

Introduction video

Terraform Azure

- **Terraform** allows you to write your **cloud setup in code**
- If you have used Azure before, you'll know that setting up your infrastructure using the **Azure Portal** (the Web UI) is **far from ideal**
- Terraform allows you use **Infrastructure as Code**, rather than executing the steps manually by going through the correct steps in the Azure Portal
- If you’re **working with Azure**, and you want to advance your career, then **terraform is the best** tool to start learning and using

Terraform Azure

```
# demo instance
resource "azurerm_virtual_machine" "demo-instance" {
  name                = "${var.prefix}-vm"
  location            = var.location
  resource_group_name   = azurerm_resource_group.demo.name
  network_interface_ids = [azurerm_network_interface.demo-instance.id]
  vm_size             = "Standard_A1_v2"

  # this is a demo instance, so we can delete all data on termination
  delete_os_disk_on_termination = true
  delete_data_disks_on_termination = true

  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "16.04-LTS"
    version   = "latest"
  }
  storage_os_disk {
    name              = "myosdisk1"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }
  os_profile {
    computer_name  = "demo-instance"
    admin_username = "demo"
    #admin_password = "..."
  }
  os_profile_linux_config {
    disable_password_authentication = true
    ssh_keys {
      key_data = file("mykey.pub")
      path     = "/home/demo/.ssh/authorized_keys"
    }
  }
}
```

- This course will teach you how to write HCL, the HashiCorp Configuration Language, to bring up your infrastructure on Azure
- Terraform is **cloud agnostic**, so the terraform skills learned in this course are easily transferrable when using terraform for other cloud providers.

Terraform Azure

- Your instructors will be **Jorn Jambers** and **Edward Viaene (that's me)**
- Together we have **years of experience** with terraform
- We both work as **Cloud Consultants** and have experience in both **enterprises and startups**, from system administration to Security & DevOps
- We are both **driven** to **provide** you a lot of **value** in this course, using the knowledge we gained by using terraform and cloud computing in our day-jobs

Course overview

Introduction	Terraform HCL	Terraform with Azure	Azure Services	Advanced Terraform
Course Intro	Introduction to terraform HCL	Introduction	MySQL & MSSQL	Remote State
Objectives	Variables	Resource Manager	CosmosDB	Conditionals
Terraform Setup	Terraform commands	Virtual Networks	Storage Accounts	Functions
Azure Setup	First steps	Virtual Machines	Azure AD	For & Foreach loops
SSH Key Setup		Network Security Groups	Application Gateway	
		Availability, Scaling, LoadBalancing	Streams	

Introduction

Course introduction

- **Thank you** for taking this course!
- My name is **Edward,** and I will be your **instructor** together with **Jorn**
- We both have **years of experience with terraform**, and we want to share our knowledge with you
- At the **end of 2016** I launched **my first terraform course**, covering terraform with AWS - which I still keep up-to-date
- Terraform has improved since then
- Using all those new features that came out, we'll be covering **terraform with Azure** in this course

Course introduction

- The beginning of this course is focussed on **getting your environment ready**
    - I have steps outlined for MacOS, Windows, and Linux
- After that, I'll show you a **first example of launching VMs** on Azure
- I'll then spend some time on going over the **basic components that you need to understand in Azure** to be able to launch infrastructure (like Resource Groups and Networking basics)
- We are then all set to start a **deep dive** in lots of different **Azure Services**. Jorn and I will guide you through the different Azure services and how to deploy them on Azure
- After you get used to provisioning infrastructure on Azure, we will spend time on explaining the **more advanced features** in terraform

Objectives

- To be able to **provision cloud infrastructure** with terraform
- To be able to **write and understand** the programming language terraform is using - **HCL** (the HashiCorp Configuration Language)
- To understand **Azure infrastructure setup** using terraform
- To get familiar with **provisioning Azure Services**
- To be able to use **advanced features** in Terraform like loops and conditionals
- To know how to use the **terraform documentation**
- To be able to **start your own terraform project** with Azure Cloud

Support

- If you need help, use the Q&A of the course and we'll try to answer as quick as we can!

- We also have a Facebook group, which you can join, called "Learn DevOps: Continuously Deliver Better Software".

  The URL is https://www.facebook.com/groups/devops.courses/

  or scan the QR code:

Who is Edward Viaene

- My name is Edward Viaene
- I am a **consultant** and **trainer** in DevOps & Cloud Technologies
- **DevOps & Cloud Advocate**
- Held various roles from **banking** to **startups**
- I have a **background** in Unix/Linux, Networks, Security, Risk, and distributed computing

Who is Jorn Jambers

- Jorn is a **freelance DevOps consultant** and **trainer**
- A DevOps **advocate**
- Worked in banks, consultancy companies and then in startups where he found his **passion** for **DevOps**
- He has a **background** in Unix/Linux, Hadoop, DBA, Networks, automations
- Today he helps companies **succeed** on the public cloud at **IN4IT**

Online Training

- We are both training instructors on Udemy
  - **DevOps**, **Distributed Computing**, **Cloud, Big Data**
  - Using online video lectures
  - 100,000+ enrolled students in 100+ countries

Terraform installation

Terraform installation

- First of all, you'll need to **download & install terraform**
- You can install it in different ways, depending on your Operating System
  - For MacOS, the easiest way is to use "brew install terraform" in a terminal
    - Alternatively, you can download the zip file manually from their website and move it to a "bin" directory
  - For Windows, you'll need to download the zip for windows, and change the PATH system variable to the directory you extracted it

Demo

Installing terraform

Demo

Installing terraform on Windows

Demo

Installing the Azure client

Demo

Installing the Azure client - Windows

Demo

Creating SSH certificates

Installing Azure Client

Demo

Terraform HCL

Terraform Commands

Terraform commands

- To get started with terraform you need to know the **basic commands**:
  - The command you'll type the most is:

    terraform apply

  - This command will read your *.tf files and apply the terraform code to the cloud provider that you have configured
  - Terraform will output the changes it will make and ask if it can make the changes
  - You can respond with "yes" to apply the changes (or use the -auto-approve argument to automatically approve the changes without asking)

Terraform commands

- If you only want to run a “plan” - to see what changes terraform would do without applying it, you can run:

  terraform plan

Terraform commands

- Every time you add a new module, a provider, or the first time you want to use terraform within a project directory, you'll have to run:

  terraform init

Terraform commands

- When you finish a demo and you'd like to remove all the infrastructure you created, you run

  terraform destroy

Terraform commands

- To get a full list of all commands you can use:

  terraform help

First steps

Demo

First steps

- In this demo, I'll setup
  - An Azure **Resource Group**
    - A **logical container** that holds Azure resources like Network, VM, Databases
  - A **Virtual Network**
    - Which has a private **address space**, and is created in a specific Azure **Region**
  - A **Virtual Machine**

First steps

- A **Disk for VM storage**
- A **Network interface** which can give us a private & public IP address
- Attached to the **network interface**:
  - A **Network Security Group** to allow **SSH access** to our VM
  - A **public IP address**

Terraform with Azure

Terraform with Azure: Introduction

Introduction

- In the previous demo I showed you how we can **start a simple VM**
- It already showed you a few **basic concepts of Azure**
- I'll now spend some time explaining these concepts while working towards a **new demo** with **Autoscaling, Scale Sets and a Load Balancer**
- If you're already familiar with Azure, you might skip some of the next lectures and skip straight to the demos
- If you're not familiar with Azure yet, then make sure to follow these lectures as they explain the **basic concepts in Azure**

Introduction

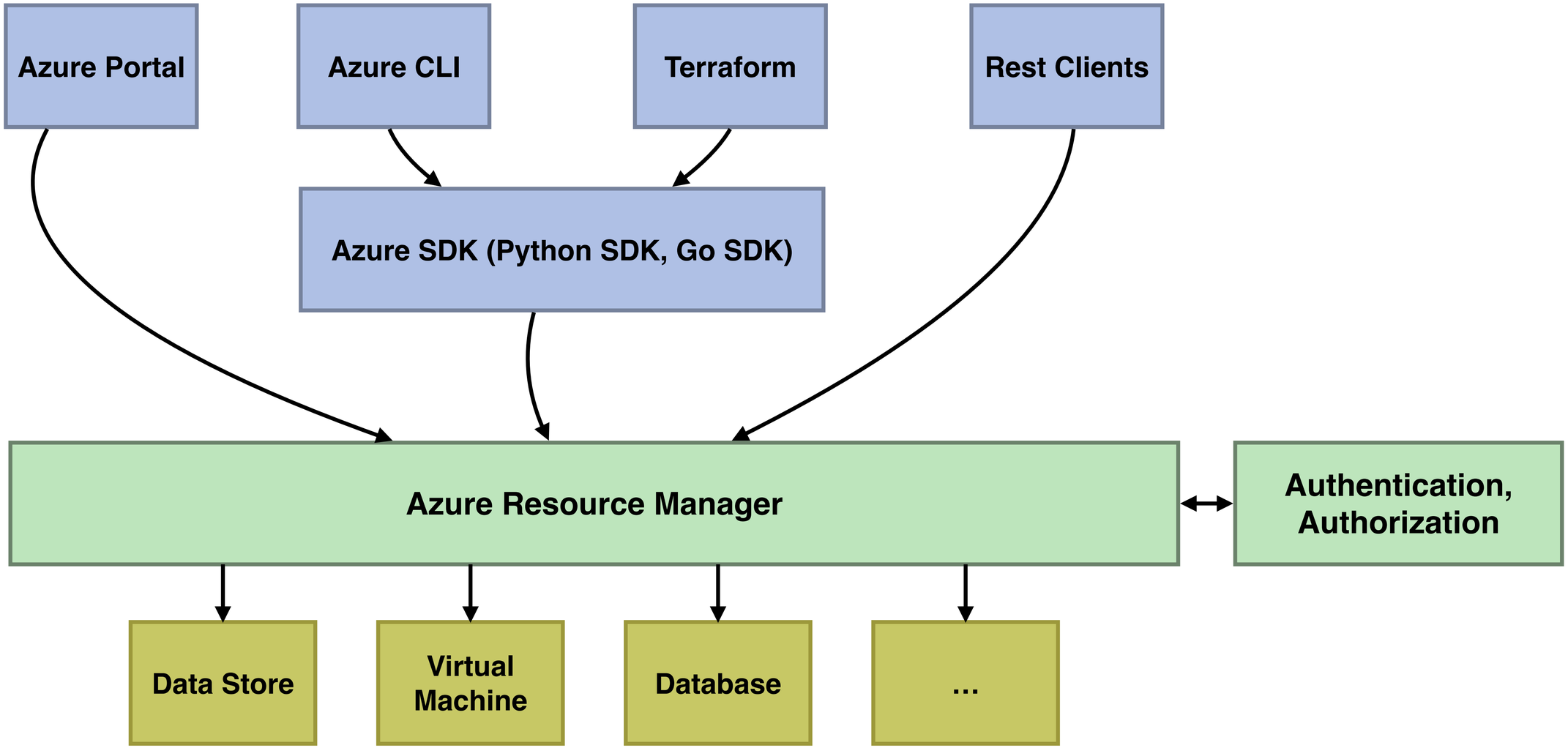
- The concepts I'm going to cover in the next lectures are:
    - The Resource Manager
    - Virtual Networks
    - Azure Virtual Machines
    - Network Security Groups & Application Security Groups
    - Availability
    - (Auto)Scaling & Load Balancing

Resource Manager

Resource Manager

- The Resource Manager is a **deployment and management service** in Azure
- It's the management layer to **create, update and delete** resources in your Azure subscription
- The terraform AzureRM plugin uses the Azure SDK to connect to the Resource Manager
    - The resource manager provides **authentication and authorization**

Resource Manager



Scope

Management Groups	Groups to manage your subscriptions
Subscriptions	Trials, Pay as you go, or Enterprise Agreements
Resource Groups	Container that holds your resources
Resources	VNets, VMs, Storage, …

Resource Groups

- **Azure Resources** (VMs, Network interfaces, VNets) are grouped in **Resource Groups**
    - Resource groups are the **logical containers** that hold your Azure resources
- Resource groups are **part of the Resource Manager**

Resource Groups

- A resource can **only exist within a single Resource Group**
    - A resource from one Resource Group can still use a resource from another resource group if the permissions allow it
    - For example, you can use a VNet created in one Resource Group, within another Resource Group
- Even though you assign a Resource Group to a single **region**, this is only where the **metadata** is saved
    - You can **still create resources in other regions**
- You can also **move a resource** from one resource group to the other

Resource Groups

- In the short demos that we do, we'll just use 1 resource group
- In real life scenarios you will want to use Resource Groups for better **governance** and **cost management**
  - **Role Based Access Control (RBAC)** can be applied on the resource group level, allowing you to provide access to users on a resource group level
  - **Tagging** resources can help for billing purposes, but also for automated processes, or audits
  - Resource Groups will also allow you to **effectively manage your costs**

Demo

Azure resource group
arguments & attributes

Virtual Networks

Virtual Networks

- A **Virtual Network** or **VNet** provides you with a **private network** in Azure
- A VNet is the first resource you need to have before creating VMs and other services that need **private network connectivity**
- You need to specify the **location** (region) where you want to create a VNet and the **address space**
- The address space is the **private IP range** you can then use
    - For example within the 192.168.0.0/16, 10.0.0.0/8, 172.16.0.0/12 ranges

CIDR table

Subnet Mask	CIDR Prefix	Total IP Addresses
255.255.255.255	/32	1
255.255.255.254	/31	2
255.255.255.252	/30	4
255.255.255.248	/29	8
255.255.255.240	/28	16
255.255.255.224	/27	32
255.255.255.192	/26	64
255.255.255.128	/25	128
255.255.255.0	**/24**	**256**
255.255.254.0	/23	512
255.255.252.0	/22	1024
255.255.248.0	/21	2048
255.255.240.0	/20	4096
255.255.224.0	/19	8192
255.255.192.0	/18	16,384
255.255.128.0	/17	32,768
255.255.0.0	**/16**	**65,536**

Subnet Mask	CIDR Prefix	Total IP Addresses
255.254.0.0	/15	131,072
255.252.0.0	/14	262,144
255.248.0.0	/13	524,288
255.240.0.0	/12	1,048,576
255.224.0 0	/11	2,097,152
255.192.0.0	/10	4,194,304
255.128.0.0	/9	8,388,608
255.0.0.0	**/8**	**16,777,216**
254.0.0.0	/7	33,554,432
252.0.0.0	/6	67,108,864
248.0.0.0	/5	134,217,728
240.0.0.0	/4	268,435,456
224.0.0.0	/3	536,870,912
192.0.0.0	/2	1,073,741,824
128.0.0.0	/1	2,147,483,648
0.0.0.0	/0	4,294,967,296

Virtual Networks

- Once a VNet is created you can create **subnets**
    - For example if you create a 10.0.0.0/16 VNet, you could create the following **subnets**:
        - VM subnet: 10.0.0.0/21 (10.0.0.0 - 10.0.7.255)
        - Database subnet: 10.0.8.0/22 (10.0.8.0 - 10.0.11.255)
        - Load Balancer subnet: 10.0.12.0/24 (10.0.12.0 - 10.0.12.255)
- You then launch your VM in one specific subnet

Virtual Networks

- When creating a subnet, azure will reserve **5 IP addresses** for own use:
  - x.x.x.0: Network address
  - x.x.x.1: Reserved by Azure for the default gateway
  - x.x.x.2, x.x.x.3: Reserved by Azure to map the Azure DNS IPs to the VNet space
  - x.x.x.255: Network broadcast address

Virtual Networks

- For each subnet you create, Azure will create a default route table
- This ensures that IP addresses can be routed to other subnets, virtual networks, a VPN, or to the internet
- You can override the default routes by creating your own custom routes

Virtual Networks

Address prefix	Next hop type
10.0.12.0/24 if the virtual network is 10.0.12.0/24	Virtual network
0.0.0.0/0	Internet
10.0.0.0/8	None
192.168.0.0/16	None
172.16.0.0/12	None
100.64.0.0/10	None

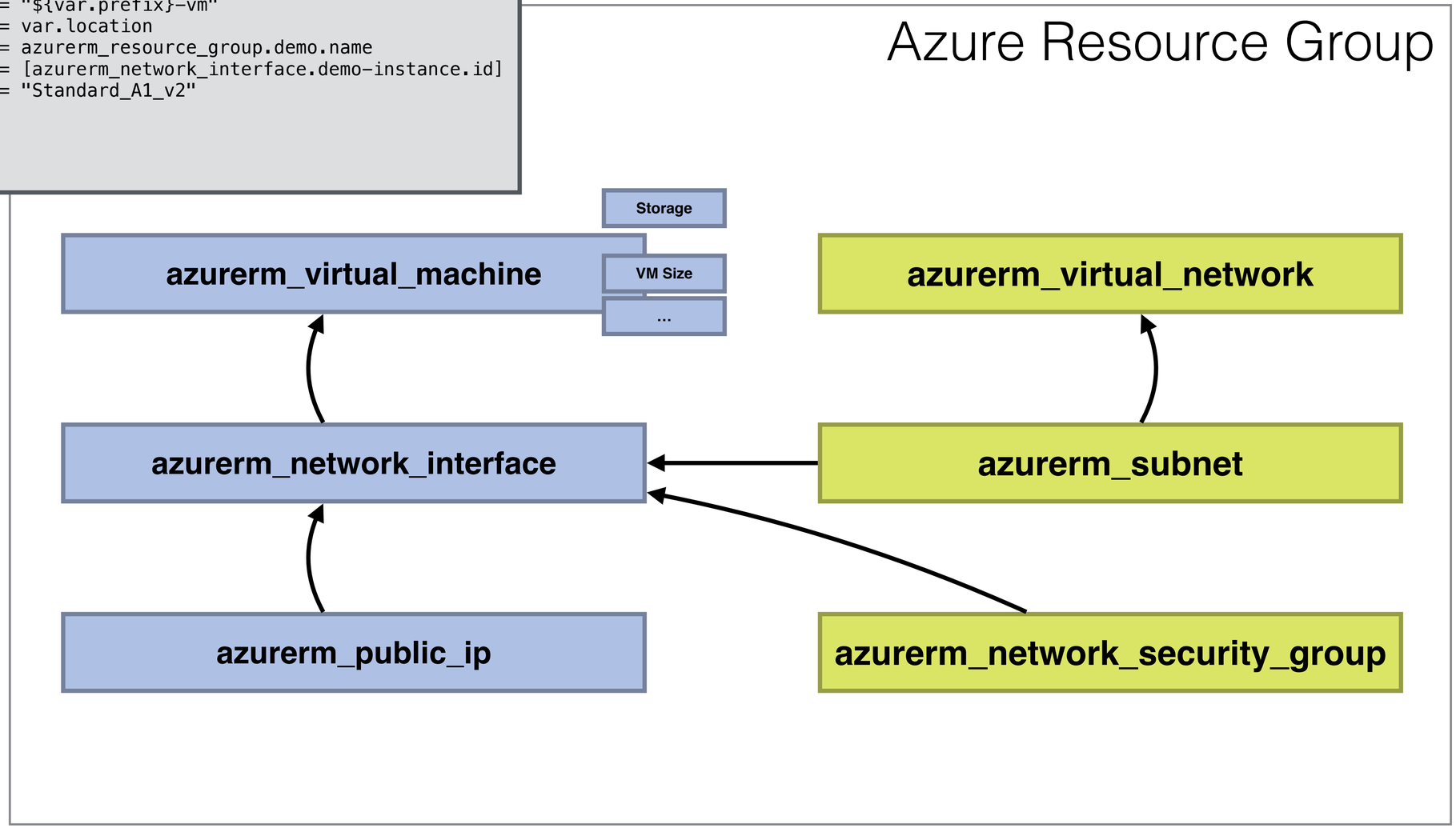
Virtual Machines

Virtual Machines

- In our first steps demo I showed you how to launch a Virtual Machine
    - It might help you to open the instance.tf file again and have another look at it
- In this lecture I want to go over some of **common the arguments** you can use with Virtual Machines
- Besides the arguments I'm going to explain here, there are **many others**
    - You should always refer to the **terraform documentation** for a full list of arguments

Virtual Machines

```
resource "azurerm_virtual_machine" "demo-instance" {
  name                  = "${var.prefix}-vm"
  location              = var.location
  resource_group_name   = azurerm_resource_group.demo.name
  network_interface_ids = [azurerm_network_interface.demo-instance.id]
  vm_size               = "Standard_A1_v2"

  […]
}
```



Virtual Machines

- You typically need **the following to launch a VM**:
  - A name
  - The location
    (typically the same region as your other resources)
  - The Resource Group
  - A Network Interface
  - The Image (for example Ubuntu)
  - Storage for the OS disk
  - The VM Size
  - The OS profile (and a Linux or Windows profile)

```
resource "azurerm_virtual_machine" "demo-instance" {
  name                  = "${var.prefix}-vm"
  location              = var.location
  resource_group_name   = azurerm_resource_group.demo.name
  network_interface_ids = [azurerm_network_interface.demo-instance.id]
  vm_size               = "Standard_A1_v2"

  # this is a demo instance, so we can delete all data on termination
  delete_os_disk_on_termination = true
  delete_data_disks_on_termination = true

  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "16.04-LTS"
    version   = "latest"
  }
  storage_os_disk {
    name              = "myosdisk1"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }
  os_profile {
    computer_name  = "demo-instance"
    admin_username = "demo"
    #admin_password = "..."
  }
  os_profile_linux_config {
    disable_password_authentication = true
    ssh_keys {
      key_data = file("mykey.pub")
      path     = "/home/demo/.ssh/authorized_keys"
    }
  }
}
```

Network Interface

- Network interface
  - You can assign a **Network Security Group** to create firewall rules for your instance
  - You can assign a **private and/or public IP address to a network interface**
    - The public IP is an external internet routable IP address
    - The private IP is within your Virtual Network range
    - The allocation can be Dynamic or Static

Network Interface

- For a **private IP addresses**:
  - IP addresses will be released when the network interface is deleted
  - When using Dynamic allocation, the next unassigned IP address within the subnet's IP range will be assigned
    - For example within a subnet 192.168.0.0/24:
    - 192.168.0.1-192.168.0.3 is reversed
    - 192.168.0.4 will be assigned first (and if this one is taken, then 192.168.0.5, and so on)
  - When using Static allocation, you can pick the private IP yourself

Network Interface

- For a **public IP addresses:**
  - You have a **Basic SKU** (default) and a **Standard SKU** (which supports Availability Zone scenarios)
  - Basic SKUs can be Dynamic or Static, Standard SKUs only Static
  - When assigning a Dynamic public IP, the IP will not be assigned yet when you create the public_ip resource. It'll only be assigned when the VM is started
    - The IP is deleted when you stop or delete the resource
  - If you want a static IP (immediately assigned), then you can choose for Static type, and you'll get a static IP from an available public IP pool, until you delete the public_ip resource
    - The IP will not be deleted when you stop or delete the resource, enabling you to attach it to another resource

Virtual Machines

- The Image:
  - You can find images using the **marketplace**
  - Typically when you find an publisher, you can list the offers and SKUs that you need in terraform by using:

    az vm image list -p "Microsoft"
    az vm image list -p "Canonical"

Virtual Machines

- **OS Storage** is needed to launch a Virtual Machine
- This is provided by an **Azure Managed disk**
    - This is a highly durable and available virtualized disk with three replicas of your data
- Interesting arguments to mention here are:
    - caching: you can choose what kind of caching you want locally (on the VM): None, ReadOnly, or ReadWrite
    - managed_disk_type:
        - LRS stands for “locally redundant storage” which replicates the data three times within one datacenter
        - You can currently choose Standard_LRS, StandardSSD_LRS, Premium_LRS or UltraSSD_LRS

Virtual Machines

- VM Size:
  - General Purpose, Compute optimized, Memory optimized, Storage optimised, GPU, High performance Compute
  - For the demo I picked within the **General Purpose** the **Av2**-series which is good for **entry level workloads**
    - Within General Purpose you have **much more types** with each their own characteristics:
      - **B**, Dsv3, Dv3, Dasv4, Dav4, DSv2, Dv2, **Av2**, DC
    - The **B-series** is another interesting type, because it is **burstable** - ideal for workloads that do not need **full performance** of the CPU continuously

Virtual Machines

- OS Profile (os_profile):
  - This is where you can set computer name, login and password
- OS Profile for Linux (os_profile_linux_config):
  - Here you can configure an SSH key instead of a password if desired, which I would recommend

Network Security Groups

Network Security Groups

- Network Security Groups **can filter traffic** from and to Azure resources
- A Network security Group **consists of security rules**, which have the following parameters:
  - Name: **unique name** of the security group
  - Priority: A **number** between **100 and 4096,** with lower numbers processed first
  - **Source or destination IP range** (or alternatively a service tag / application security group)
  - **Source & Destination Port Range**
  - **IP Protocol**: TCP / UDP / ICMP / Any
  - Direction: **incoming / outgoing**
  - Action: **Allow / Deny**

Network Security Groups

```
resource "azurerm_network_security_group" "allow-ssh" {
    name                = "${var.prefix}-allow-ssh"
    location            = var.location
    resource_group_name = azurerm_resource_group.demo.name

    security_rule {
        name                       = "SSH"
        priority                   = 1001
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = var.ssh-source-address  # “*” or 1.2.3.4/32
        destination_address_prefix = "*"
    }
}
```

Network Security Groups

- A newly created Network Security Group has these default **inbound** rules:

Priority	Source	Source Ports	Destination	Destination ports	Protocol	Access
65000	VirtualNetwork	0-65535	VirtualNetwork	0-65535	Any	**Allow**
65001	Azure LoadBalancer	0-65535	0.0.0.0/0	0-65535	Any	**Allow**
65500	0.0.0.0/0	0-65535	0.0.0.0/0	0-65535	Any	**Deny**

Network Security Groups

- A newly created Network Security Group has these default **outbound** rules:

Priority	Source	Source Ports	Destination	Destination ports	Protocol	Access
65000	VirtualNetwork	0-65535	VirtualNetwork	0-65535	Any	**Allow**
65001	0.0.0.0/0	0-65535	Internet	0-65535	Any	**Allow**
65500	0.0.0.0/0	0-65535	0.0.0.0/0	0-65535	Any	**Deny**

Network Security Groups

- Our **inbound** Network Security Group rules for the first steps VM will look like this:

Priority	Source	Source Ports	Destination	Destination ports	Protocol	Access
1001	var.ssh-source-address (* or IP range)	*	*	22	Tcp	Allow
65000	VirtualNetwork	0-65535	VirtualNetwork	0-65535	Any	Allow
65001	Azure LoadBalancer	0-65535	0.0.0.0/0	0-65535	Any	Allow
65500	0.0.0.0/0	0-65535	0.0.0.0/0	0-65535	Any	Deny

Network Security Groups

- When creating security groups, instead of IP addresses, you can use **Service Tags** or **Application Security Groups**

- **Service Tags** are predefined by Azure, for example:

  - VirtualNetwork: The VirtualNetwork address space, for example 10.0.0.0/16

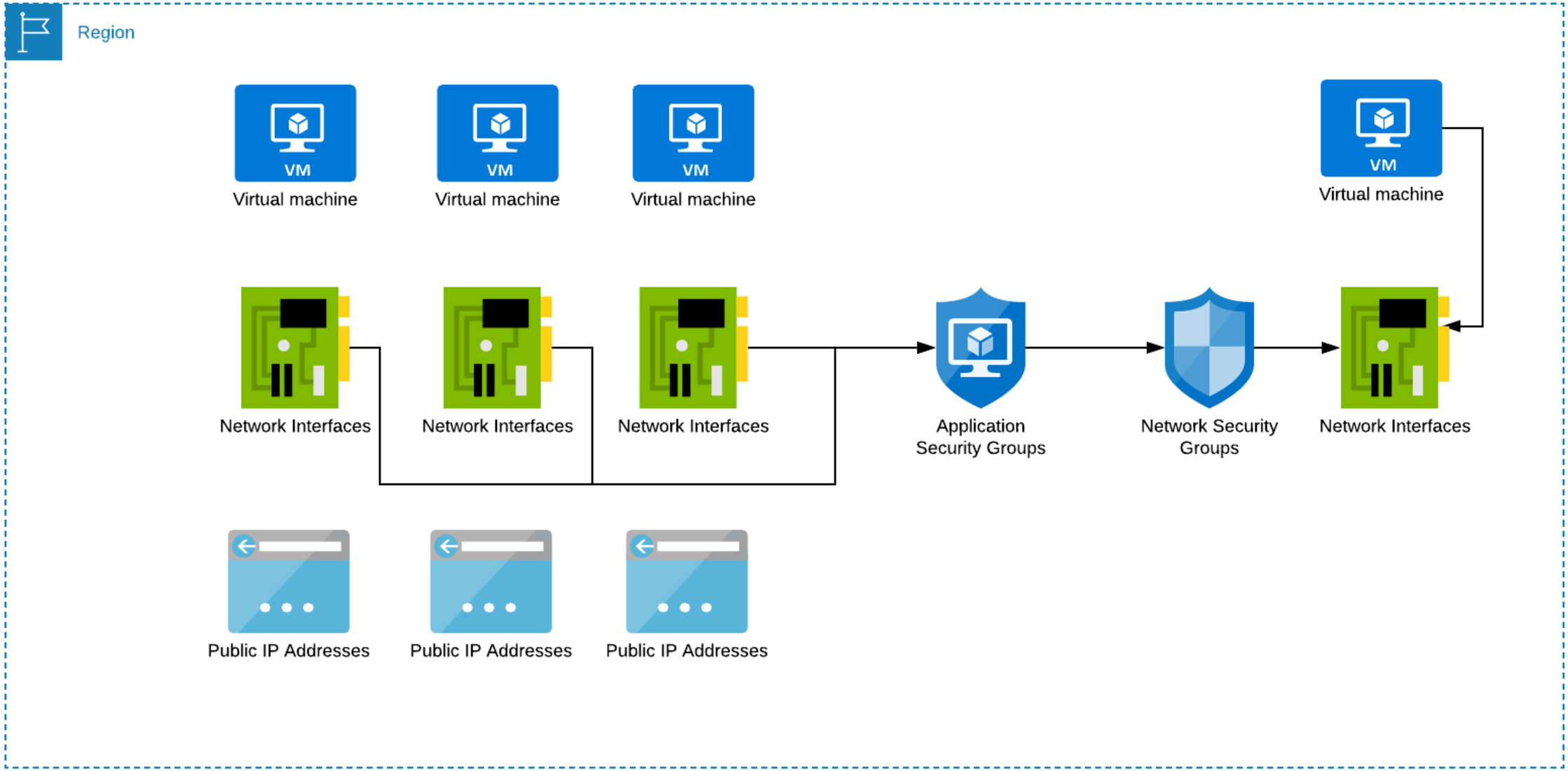
  - AzureLoadBalancer: translates to the Virtual IP where Azure health checks originates from

  - Internet: Outside the VirtualNetwork, reachable by the public internet

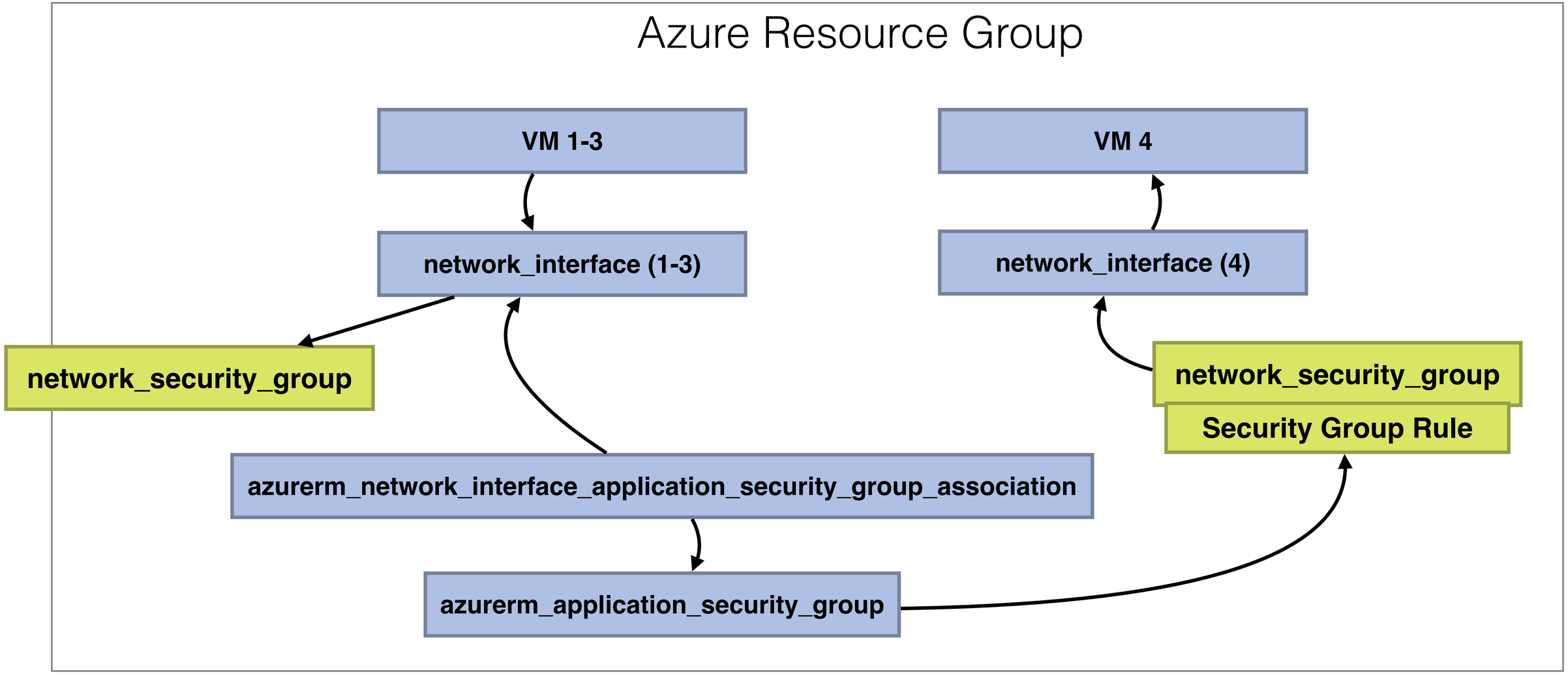
Application Security Groups

- Application Security Groups allow you to **group Virtual Machines**
- Instead of using IP addresses, you can use **group names** instead, making your Network Security Groups much easier to maintain
- You will need to associate (link) one or more Network Interfaces to an Application Security Group
    - You can associate **multiple network interfaces** that make up 1 application and call the Application Security Group “MyApplication”
- Afterwards you’ll be able to use that “MyApplication” within a network security rule, rather than specifying the single IP addresses

Application Security Groups



Application Security Groups



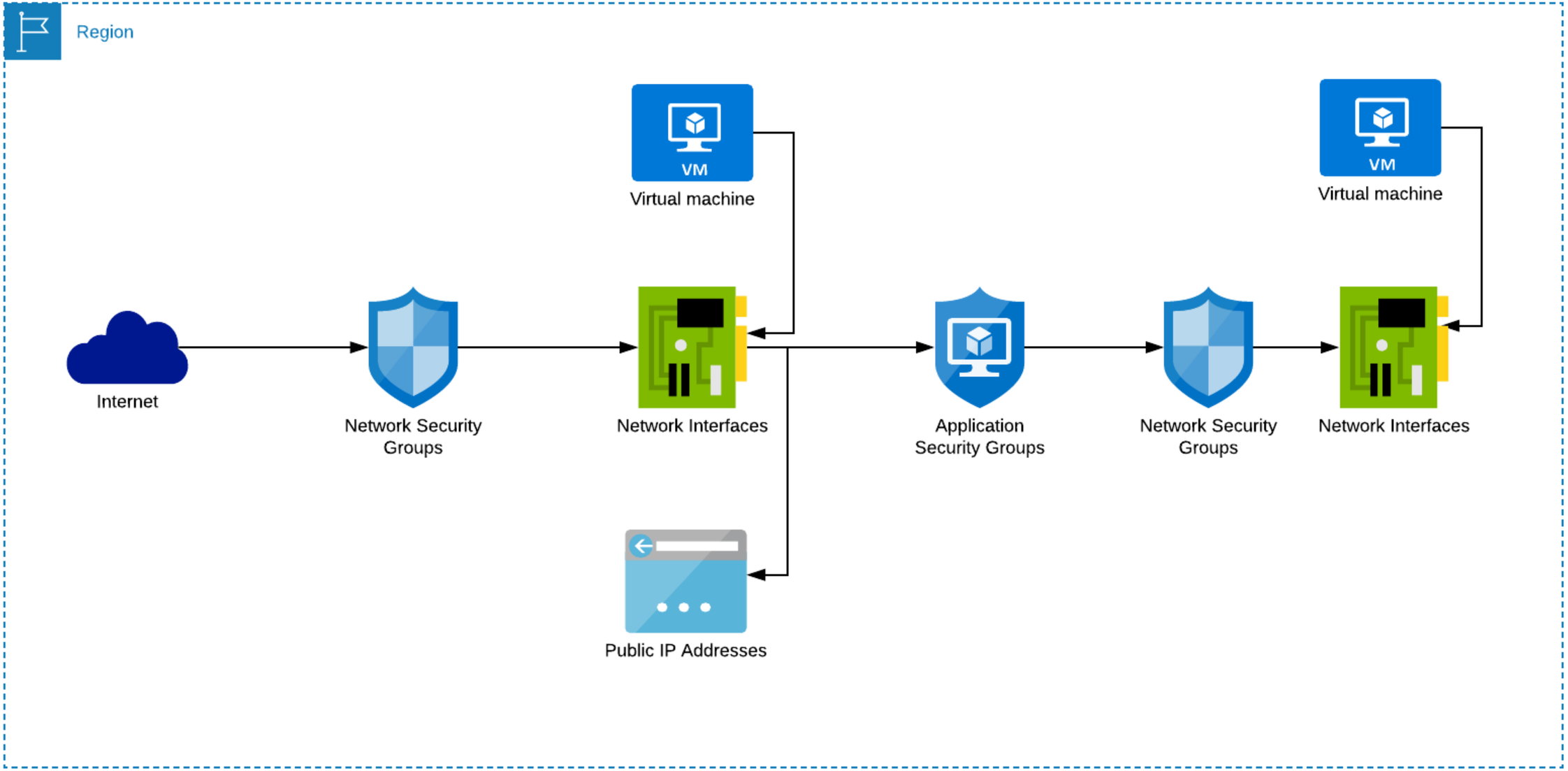
Security Groups Troubleshooting

- When creating infrastructure, how do you **troubleshoot** Security Groups?
- A few general tips:
  - If you're getting "Connection timeout" then it's most likely the security group that is blocking you
    - It can also be that the VM is not responding or you're using the wrong DNS / IP
  - If you're getting "Connection refused", you can reach the VM, and it's the VM that sends you back that the port is not open
  - If you're getting a SSH key error, check whether you're using the correct key, and whether you're passing your private key (-i in macos/linux)

Network Security Groups

Demo

Application Security Groups



Availability, Scaling, LoadBalancing

Availability

Availability Zones

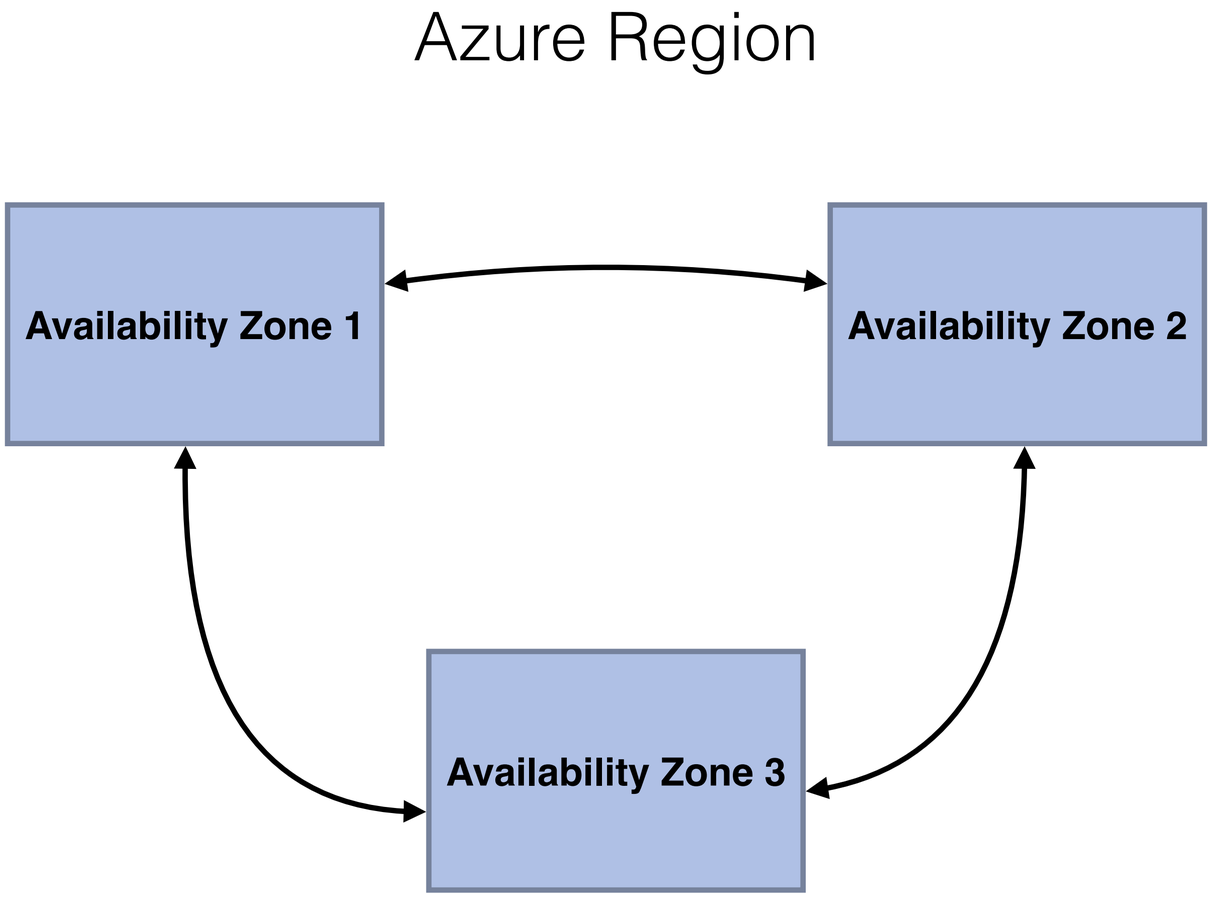
- Availability Zones can **protect your applications and data against datacenter failure**

- Not all regions support Availability Zones, you'll have to check the **region map** at https://azure.microsoft.com/en-us/global-infrastructure/regions/ to see whether your region supports Availability Zones

- Each Availability Zone is a **unique physical location** within the same region

    - They are made up of one or more datacenters with **independent power, cooling, and networking**

Availability Zones



Availability Zones

- There are 2 categories of services that support Availability Zones:
  - **Zonal services**: you specify in what Availability Zone they run (for example a VM, Managed Disk, …)
  - **Zone-Redundant**: services that automatically replicate across zones (for example zone redundant storage)
- Be aware that Availability Zone identifiers (1,2,3) are **mapped differently** for each subscription
- Availability Zone 1 can be different in subscription A than in subscription B

Fault & Update Domains

- **Fault Domain**: logical group of underlying hardware with **common power source and network switch**, like a rack in on-premises terminology

- **Update Domain**: logical group of underlying hardware that can undergo **maintenance or be rebooted** at the same time

- You generally want to make sure that your Virtual Machines are in a different fault domain and update domain, to ensure high availability for your application when a power source / network switch fails or when an update is performed and the machine is temporary offline

    - This especially when you can't place your VMs cross-zone, for example when the region you're in doesn't support multiple Availability Zones

Scaling & LoadBalancing

Scale Sets

- A scale set launches a group of Virtual Machines
- You can manually or automatically **scale up or down** by **adding or removing** VMs
- This is horizontal scalability, you add or remove VMs, the size or type of the VM stays the same
- You typically create an autoscaling group with x amount of instances
- You can then create autoscaling rules or manually change the size when demand is higher

Scale Sets

- Scale sets provide **high availability and application resiliency**
    - If one of the VMs has a problem, another VM can still handle requests
- All VMs should have the same VM type, base OS and configuration, making it **easy to handle one, ten, or hundreds VMs** in a scale set
- You typically put a **Load Balancer** in front of the VMs to load balance the requests over the multiple VMs
- Using scale sets can also save you money, by **better resource utilization**
    - You can scale up when demand is high, but also scale down when demand is low

Scale Sets

- Virtual Machine Scale Sets are created with **5 fault domains by default in a region without Availability Zones**
    - This ensures that the VMs are spread over the datacenter to increase availability
- If the region supports **Availability Zones**, then the value of fault domains will be **1 in each of the zones**
    - In this case the VM instances will be **spread across multiple zones**, across as many racks on a best effort basis

Scale Sets

- Another advantage of Scale Sets is that you can enable "**Automatic OS image upgrades**"
- During the upgrade the **OS disk of the VM will be replaced** with the latest version, and a configured health probe will check whether it was succesful
- This can be done one by one or in batch, taking into account a max percentage of images that can be unhealthy
    - The process will also stop if there more than a certain percent **unhealthy VMs post-upgrade**
- Currently offered on the official **UbuntuServer** images, **CentOS**, and specific **WindowsServer** versions

Load Balancers

- Once you have your scale set, you typically put a **Load Balancer** in front of it
- The Azure Load Balancer supports **inbound and outbound traffic**
  - Inbound: **from internet to the Load Balancer** to your backend VMs
  - Outbound: **from your backend** VMs **to the internet**
- To route the traffic from the Load Balancer to the backends, you setup Load Balancer Rules
  - For example, port 80 (http) to port 8080 (application) on the VM backends

Load Balancers

- Azure Load Balancers are available with **2 different SKUs**: Basic & Standard
- Basic is currently available at **no extra charge**
- **Standard incurs a charge**, but supports extra features and scaling (it supports Availability Zones)
- The Standard Load Balancer provides a **zone-redundant frontend** for inbound and outbound traffic
    - Only 1 public IP of type Standard (instead of Basic) needs to be assigned, which will **automatically reroute traffic if a zone failure would occur** (a 2 public IP zone-specific solution is also possible for more granular control)

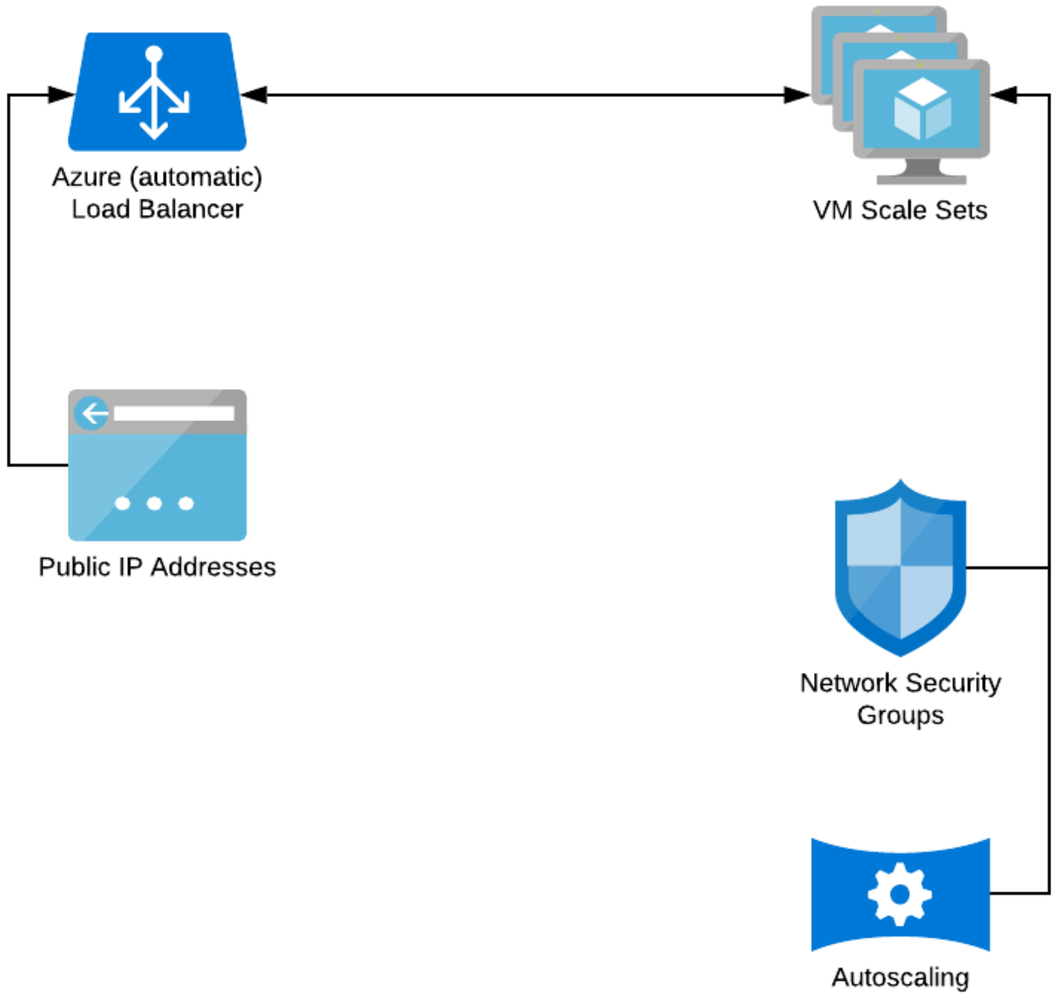
Load Balancers

- Besides Load Balancing you can also do **port-forwarding**, creating an **inbound NAT rule** to forward a port from the Load Balancer to a specific backend
    - Used for example to map unique ports on the Load Balancer to port 22 on the backends
    - Port 50002 On Load Balancer => backend1:22
    - Port 50003 On Load Balancer => backend2:22
- This type of Load Balancer **doesn't terminate, respond or interacts with the payload of UDP / TCP packets**, it only forwards it: **it's not a proxy**
    - If you're looking for a Level-7 Load Balancer (which acts like a proxy), then you'll have to implement an "**Application Load Balancer**" - which can also do application-layer processing and terminate TLS

Scaling & LoadBalancing

Demo

Demo Load Balancer & Scale Set



Scaling & LoadBalancing
Availability Zones

Demo

Terraform Usage

Understand terraform basics

Understanding terraform basics

- Terraform **installation**
    - Covered in the beginning of the course
- Terraform uses **providers**, which are **shipped separately** with their own version numbering
- The **terraform core** contains the language interpreter, the CLI, and how to interact with those providers (not the providers itself)
- It doesn’t contain the **code to interact with the API** of the cloud providers to create resources, that code will be found in the "**providers**", which will be installed separately when invoking "terraform init"

Understanding terraform basic

- The **terraform registry** is the main directory for providers and can be found at https://registry.terraform.io/browse/providers

- The most known providers are:

  - Cloud providers: Azure, AWS, GCP, Oracle Cloud, Alibaba Cloud

  - Kubernetes, Helm

  - Active Directory, DNS, HTTP

  - Hashicorp Vault

Understanding terraform basic

- You can immediately start using terraform resources from a specific provider (for example azurerm_linux_virtual_machine), and **terraform init** will install this provider
- Terraform by default will download the latest available version of that provider
- If you want, and it's good practice, you can specify the provider requirements in the terraform block
- Besides the terraform provider requirements, you can also specify the minimum terraform version

Understanding terraform basic

```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "2.42.0"
    }
  }
  required_version = ">= 0.14"
}

provider "azurerm" {
  # Configuration options
}
```

```
terraform {
  required_providers {
   mycloud = {
      version = ">= 1.0.0"
      source  = "mycorp/mycloud" # terraform cloud's private registry (terraform-provider-mycloud)
    }
  }
  required_version = ">= 0.14"
}
```

Understanding terraform basic

- Terraform released breaking changes between 0.12, 0.13, 0.14, etc
- Bugfixes are performed in the patch releases: 0.12.1, 0.12.2, etc
- Terraform provider versioning follows **semantic versioning**:
  - MAJOR.MINOR.PATCH
    - PATCH = bug fixes only
    - MINOR = new features
    - MAJOR = possible breaking changes

Understanding terraform basic

- Once you are using a provider, you can also specify provider configuration

```
provider "azurerm" {
  subscription_id = “00000000-0000-0000-0000-000000000000"
  tenant_id       = “11111111-1111-1111-1111-111111111111"
}
```

Understanding terraform basic

- You can also use multiple providers, with the "alias" meta-argument

```
provider "azurerm" {
  subscription_id = “00000000-0000-0000-0000-000000000000"
  tenant_id       = “11111111-1111-1111-1111-111111111111"
}

provider "azurerm" {
  alias           = "tenant2"
  subscription_id = “00000000-0000-0000-0000-000000000000"
  tenant_id       = “22222222-2222-2222-2222-222222222222"
}

resource "azurerm_virtual_machine" "myvm" {
  provider = azurerm.tenant2
  […]
}
module "mymodule" {
  source = "./mymodule"
  providers = {
    azurerm = azurerm.tenant2
  }
```

Understand terraform basics

Provisioners

Understanding terraform basics

- We have **multiple ways of provisioning VMs**:
  - Local-provisioner (execute something locally after spinning up a VM)
  - Remote-provisioner (execute something remote on the VM)
  - Packer (build source image, then launch virtual machine based on that source image)
  - Cloud init (using custom_data, pass provisioning to Azure API so VM can provision at creation)

Understanding terraform basics

- **Provisioners** (local-exec / remote-exec) are **separate flows** that cannot be fully controlled by terraform
    - Provisioners add a considerable amount of **complexity and uncertainty**
    - More coordination required: security groups need to be open, network access to the instances to run provisioning
- Therefore, you should only use provisioners as **last resort**, when other approaches are not possible

Understanding terraform basics

- For most use cases, you'll be able to use **cloud init**
  - Cloud init (custom_data in azurerm_linux_virtual_machine), will run after the virtual machine will launch for the first time
  - Other cloud providers have a similar approach (Google Cloud has metadata, AWS user_data in aws_instance, etc)
- Since **Kubernetes** & other **container orchestrators** are used for provisioning, **virtual machine provisioning** becomes **less of an issue**
  - Provisioning happens when building the container, then the container is launched on a container platform

Using terraform CLI

Use the terraform CLI

- For the certification, you need to know about a few **CLI commands** (besides init / plan / apply). Let's summarize a few of these commands in this lecture

Use the terraform CLI

Command	Description
terraform fmt	Format the *.tf files by entering "terraform fmt" or "terraform fmt filename.tf"
terraform taint	For example: terraform taint azurerm_virtual_machine.myvm Next time you run terraform apply the instance my instance will be destroyed and recreated
terraform import	If you have already resources created manually and you want to manage them in terraform, then first create the terraform code in a *.tf file, then run terraform import resource_type.resource_name unique-identifier
terraform workspace	new, list, show, select and delete Terraform workspaces
terraform state	Manipulate the terraform state file. You can move (mv), remove (rm), list, pull, push, replace-provider within the state, and show the state

Use the terraform CLI

- Terraform starts with a **single workspace** "default"
- You can create a **new workspace** using "terraform workspace new"

```
$ terraform workspace new mytestworkspace
Created and switched to workspace "mytestworkspace"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
```

- Switching to another workspace (or back to default) can be done with "terraform workspace select name-of-workspace"

Use the terraform CLI

- Once your are in a **new workspace**, you'll have an "empty" state
- Your previous state is still accessible if you select the "default" workspace again
- When you run terraform apply in your new workspace you will be able to **re-create all the resources**, and those resources will be managed by this **new state** in this new workspace
- This can be useful if you for example want to **test something** in your code without making changes to your existing resources, for example create a new instance with encrypted root devices in a new workspace to test whether your new code works, rather than immediately trying this on your existing resource

Use the terraform CLI

- To avoid **naming collisions** you can use the variable terraform.workspace

```
resource "azurerm_virtual_machine" "myvm {
  name = "/myapp/myname-${terraform.workspace}"

  […]
}
```

- Or only enable resource creation in a specific workspace:

```
resource "azurerm_virtual_machine" "myvm {
  count = terraform.workspace == "default" ? 1 : 0

  […]
}
```

Use the terraform CLI

- The workspaces cannot be used for a "fully isolated" setup that you'd need when you want to run terraform for multiple environments (staging / testing / prod)

- Even though a workspace gives you an "empty state", you're still using the **same state**, the **same backend configuration** (workspaces are the technically equivalent of renaming your state file)

- Therefore workspaces only have **limited use cases**

- In real world scenarios you typically use **re-usable modules** and really split out the state over multiple backends (for example your staging backend will be on Azure Blob Storage on your staging account, and your prod backend will be in an Azure Blob Storage bucket on the prod account, following **multi-account strategy**)

Use the terraform CLI

- If something goes really wrong, you hit a **bug**, or terraform just "**hangs**", you might want to **enable debugging mode**

- To enable more logging, you need to set the **TF_LOG environment variable**

- You can also prepend it to the terraform command on MacOS / Linux like this: TF_LOG=DEBUG terraform apply

- On windows, in Powershell, you can use:

  $Env:TF_LOG = "DEBUG"

Use the terraform CLI

- Valid log levels are:
  - TRACE
  - DEBUG
  - INFO
  - WARN
  - ERROR

Interact with Terraform modules

Interact with Terraform modules

- In this course, we covered a lot of material on modules, so let's rehearse what we learned in this lecture

- This is a typical module declaration:

```
module "consul_cluster" {
  source  = "hashicorp/consul/azurerm"
  version = "0.1.0"
}
```

- This will download a specific module version from the terraform registry

- We can also see that the module is owned by hashicorp, because it starts with hashicorp/

Interact with Terraform modules

Source	Description
Terraform registry: <NAMESPACE>/<NAME>/<PROVIDER> hashicorp/consul/azurerm	Namespace = hashicorp name = consul provider = azurerm
Terraform private registry: <HOSTNAME>/<NAMESPACE>/<NAME>/ <PROVIDER>	When no hostname is provided registry.terraform.io is assumed. A private registry example could be: registry.mycorp.com/myteam/myspecialmodule/azurerm If you're using a private registry, you might have to specify an access token in the CLI Config (terraform.rc in %APPDATA% in Windows, .terraform.rc on linux/Mac systems) See https://www.terraform.io/docs/commands/cli-config.html#credentials

Interact with Terraform modules

- You don’t necessarily need to use the registry, you can also use the modules directly if you create a directory for example:

```
module "mymodule" {
  source = "./mymodule" # refers to a local path
}
```

Interact with Terraform modules

- Terraform will also recognize GitHub (HTTPS):

```
module "mymodule" {
  source = "github.com/in4it/terraform-modules"
}
```

- And also over SSH:

```
module "mymodule" {
  source = "git@github.com:in4it/terraform-modules.git"
}
```

- These examples work with bitbucket as well (replace GitHub.com in bitbucket.org)

Interact with Terraform modules

- More examples:

```
module "mymodule" {
  source = "git::https://example.com/mymodule.git"
}

module "mymodule" {
  source = "git::ssh://username@example.com/mymodule.git"
}

module "mymodule" {
  source = "git::https://example.com/mymodule.git?ref=v1.3.0"
}

module "mymodule-over-https" {
  source = "https://example.com/mymodule.zip"
}

module "mymodule-in-s3" {
  source = "s3::https://s3-eu-west-1.amazonaws.com/examplecorp-terraform-modules/mymodule.zip"
}
```

Interact with Terraform modules

- Passing input:

```
module "mymodule" {
  source  = "./mymodule"
  myValue = "123"
}
```

- In ./mymodule/vars.tf:

```
variable "myValue" {

}
```

Interact with Terraform modules

- Getting output:

```
module "mymodule" {
  source  = "./mymodule"
  myValue = "123"
}

module "other_module" {
  network_security_group = module.mymodule.network_security_group_id
}
```

- In ./mymodule/output.tf:

```
output "network_security_group_id" {
  description = "id of the security group provisioned"
  value       = azurerm_network_security_group.vm.id
}
```

Interact with Terraform modules

- In a module you can **only use the variables that are declared** within that module
- In the root module (the root project), you can only access parameters that are **defined as output in that module**
- To access data from the root module or other modules, you can use **inputs** to pass information to the module
- To provide data to the root module, you can use **outputs** to pass information to the root module

Terraform Module Registry

Terraform Module Registry

- When using modules (and also providers), you can specify a **version constraint**

```
version = ">= 1.2.0, < 2.0.0"
```

- This version allows every version greater or equal than 1.2.0, but needs to be less than 2.0.0

- You can **separate conditions with a comma**

- The version numbering should follow semantic versioning (major.minor.patch)

Terraform Module Registry

- The following operators can be used with version conditions
  - **=** (the default, so you can as well remove it): Exactly one version
    - "=1.0.0" or "1.0.0"
  - **!=** : Excludes an exact version
    - For example when there's a known bug in a specific version
  - **>, >=, <, <=**: Greater than, greater than or equal, less than, less than or equal
  - **~>**: Allows right most version to increment

Terraform Module Registry

- ~>: Allows **right most version** to increment
  - "~> 1.2.3" will match 1.2.4, 1.2.5, but not 1.3.0
  - "~>1.2" will match 1.3, 1.4, but not 2.0

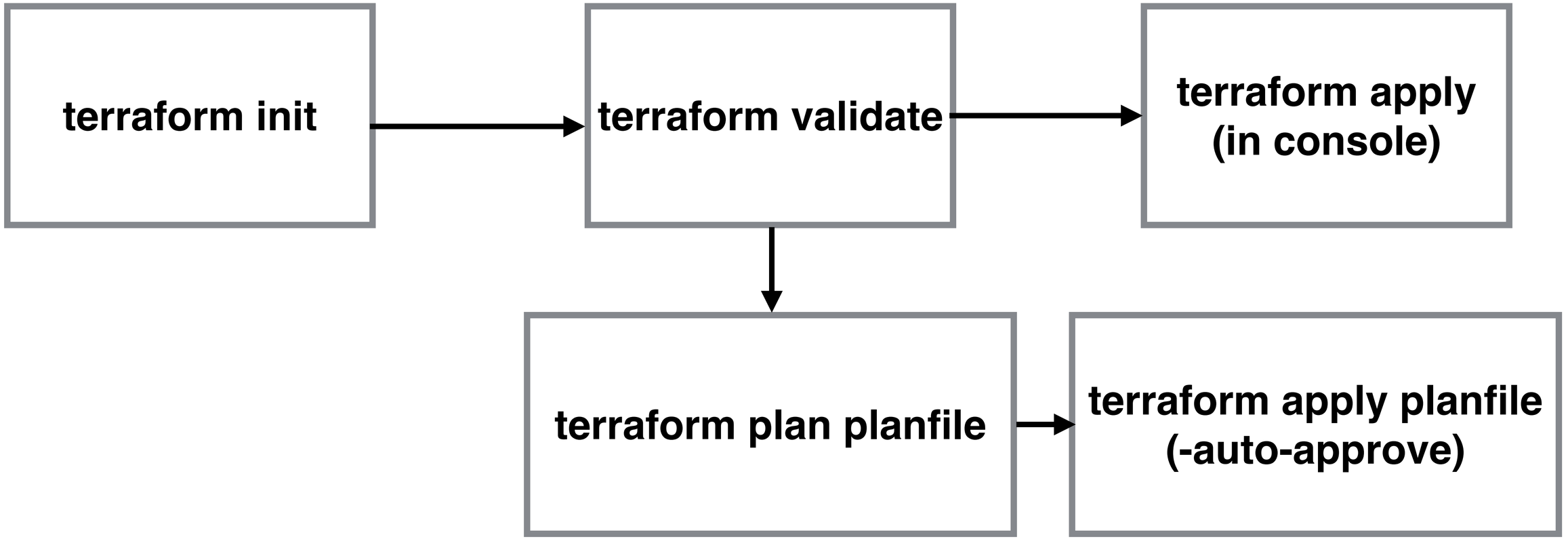
Terraform Module Registry

- Best practices
  - Terraform documentation recommends to use **specific versions for third party modules** (so that you can upgrade when convenient only)
  - For **modules within your organization**, you can **use a range**, for example "~>1.2.0" to avoid big changes when you bump to 1.3.0 (you can then bump when you need new features)
  - Within modules (when you write a module), you should supply a **minimum terraform core version** to ensure compatibility (e.g. >=0.14.0)
  - For providers you can use the **~> constraint to set lower and upper bound**
    - For example ~> 2.42.0 on the azure module will still give you new features, but will not introduce a major bump

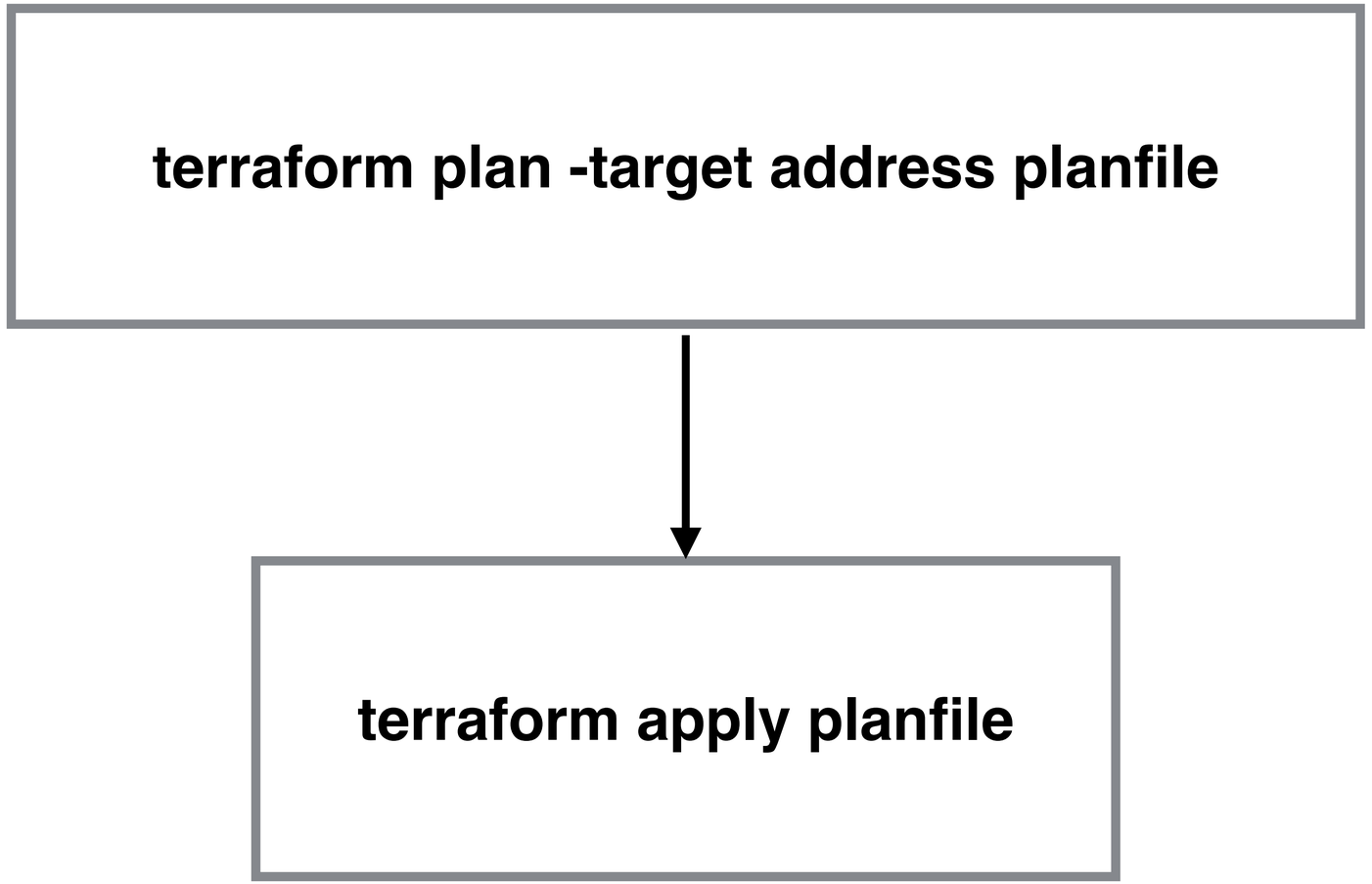
Navigate terraform workflow

Navigate terraform workflow

Navigate terraform workflow



Navigate terraform workflow



Implement and maintain state

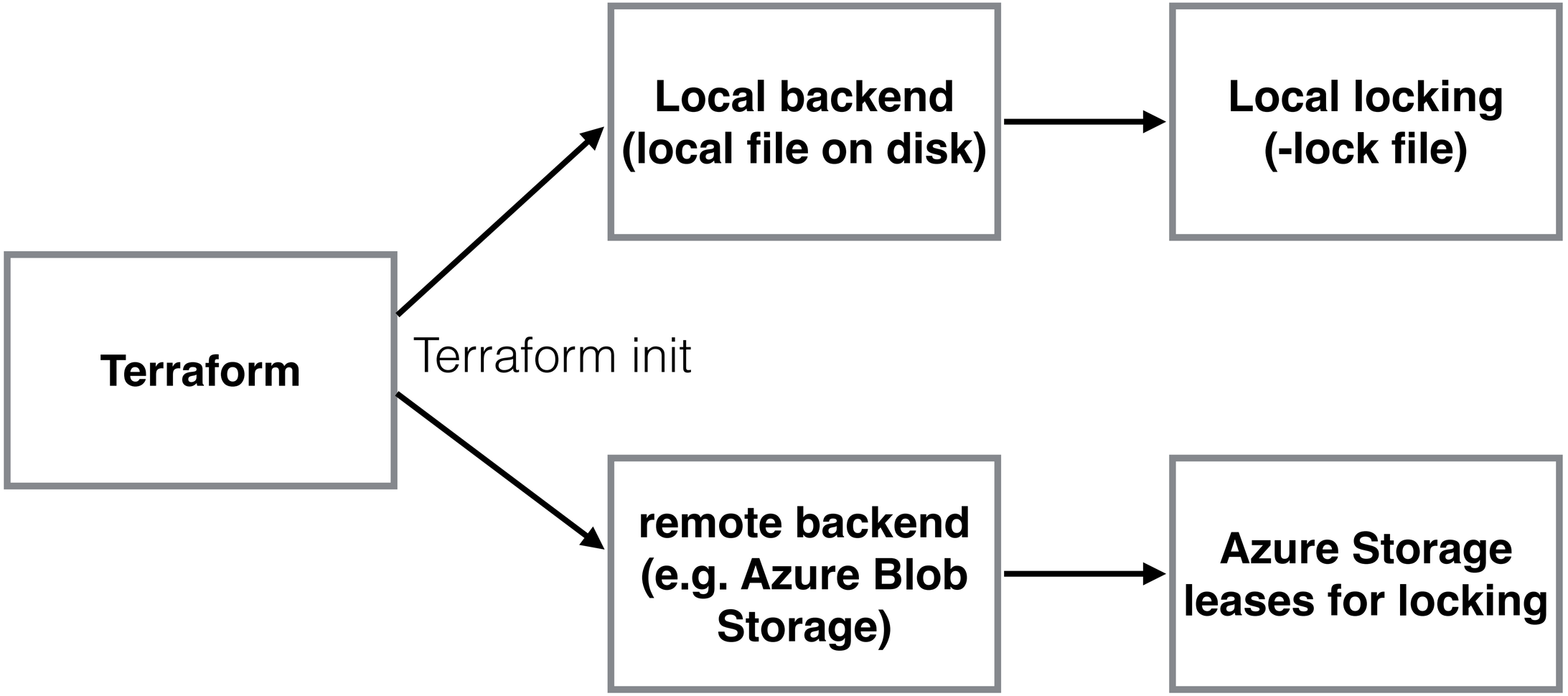
Implement and maintain state

- The **default backend** in terraform is the **local** backend, this requires no configuration
- A **terraform.tfstate** file will be written to your project folder
  - This is where the **state** is **stored**
  - Every time you run **terraform apply**, the **state** will be **changed**, and the file will be updated
- Once you start working in a team, you are going to want to use a **remote backend**

Implement and maintain state

- Working with a remote state has **benefits**:
    - You can **easily work in a team**, as the state is separate from the code (alternatively, you would have to commit the state to version control - which is far from ideal if you need to work in a team)
    - A remote backend can **keep sensitive information off disk**
        - Azure Blob Storage supports **encryption at rest**, **authentication** & **authorization**, which protects your state file much more than having it on your disk / version control
    - **Remote operations**: terraform apply can run for a long time in bigger projects. Backends, like the "remote" backend, supports remote operations that are executed fully remote, so that the whole operation runs asynchronously. You don't need to be connected / keep your laptop running during the terraform apply (more about that in the Terraform Cloud lecture)

Implement and maintain state



Azure Blob Storage: at rest encryption, versioning

Implement and maintain state

- State **locking** ensures nobody can write to the state at the same time
- Sometimes, when terraform crashes, or a users' internet connection breaks during terraform apply, the lock will stay
- "terraform force-unlock <id>" can be used to force unlock the state, in case there is a lock, but nobody is running terraform apply
    - This command will not touch the state, it'll just remove the lock file, so it's safe, as long as nobody is really still doing an apply
- There's also an option -lock=false that can be passed to terraform apply, which will not use the lock file. This is discouraged and should only be used when your locking mechanism is not working

Implement and maintain state

- Supported **standard backends**:
  - Artifactory (artifact storage software)
  - Azurerm (azure)
  - Consul (hashicorp key value store)
  - Cos (Tencent cloud)
  - Etcd, etcdv3 (similar to consul)
  - Gcs (google cloud)

Implement and maintain state

- Supported standard backends:
  - http
  - Kubernetes
  - Manta (also object storage)
  - oss (Alibaba cloud storage)
  - pg (postgres)
  - S3
  - Swift (openstack blob storage)

Implement and maintain state

- Every backend will also have a **specific authentication method** (which is explained in the terraform docs on a per backend basis)

- The configuration is done within the terraform {} block:

```
terraform {
  backend "azurerm" {
    storage_account_name = "abcd1234"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"
    use_msi              = true
    subscription_id      = "00000000-0000-0000-0000-000000000000"
    tenant_id            = "00000000-0000-0000-0000-000000000000"
  }
}
```

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "eu-west-1"
  }
}
```

Implement and maintain state

- You can have a partial backend configuration, where you leave away some of the information
- This can be useful if you would like to use **different backends** when executing the code (for example for staging / qa / prod environments)
    - This is often then **scripted** with shell scripts that call terraform with the correct arguments - this to avoid having to do this manually every time
- Most commonly this is used to **avoid having to hardcode secrets** in the terraform files, which would end up in version control

Implement and maintain state

- There are **3 ways to pass** this backend information:
  - Interactively, when the information is missing, terraform init will ask for it (only works for required values in the backend configuration)
  - A file
  - Key/Value Pairs

```
$ terraform init -backend-config=path-to-file

$ terraform init -backend-config="storage_account_name=tfstorage" \
    -backend-config="container_name=tfstate" \
    -backend-config="access_key=$(az keyvault secret show --name tfstate-storage-key --vault-name
tfseries-state-kv --query value -o tsv)" \
    -backend-config="key=terraform-tfstate"
```

Implement and maintain state

- If at some point you'd like to **update your state** file to reflect the "actual" state of your infrastructure, but you don't want to run terraform apply, you can run "**terraform refresh**"
- Terraform refresh will look at your infrastructure that has been applied and will update your state file to reflect any changes
- It'll not modify your infrastructure, it'll only update your state file
- This is often useful if you have outputs that need to be refreshed, or something changed outside terraform and you need to make terraform aware of it without having to run an apply

Implement and maintain state

- You need to be aware that **secrets can be stored in your state file**
    - For example when you create a database, the **initial database password** will be in the state file
- If you have a **remote state**, then locally it'll **not be stored on disk** (it'll only be kept in memory when you run terraform apply)
    - As a result, storing state remote can increase security
- Make sure your **remote state backend** is **protected sufficiently**
    - For example for Azure blob storage, make sure only terraform administrators have access to this, enable encryption at rest. Also make sure that for every backend TLS is used when communicating with the backend.

Read, generate, and modify configuration

Input, output, locals

Configuration

- There are 3 types of variables in terraform:
  - **Input** variables
    - variable "a-variable" { … }
  - **Output** variables
    - output "an-output" { … }
  - **Local** variables (locals { …})
    - They are like a temporary variables that you can use
    - Used for calculations, concatenations, conditionals where the result is later used within the resources

Configuration

- Input variables can have the following **optional arguments**:
  - default
  - type
  - description
  - validation
  - sensitive

Configuration

- Type **constraints**:
  - string
  - number
  - bool

Configuration

- **Complex** types:
  - list(<TYPE>)
  - set(<TYPE>)
  - map(<TYPE>)
  - object({<ATTR NAME> = <TYPE>, …})
  - tuple([<TYPE>, …])
  - "**any**" can be used as a type as well to indicate any type is acceptable

Implement and maintain state

- Let’s go over a few examples of variables

```
variable "myvariable" {
  description = "this is myvariable, it’s a string"
  type        = string
  default     = "123"
}

// from the module-flatten demo
variable "parameters" {
  type = list(object({
    prefix = string
    parameters = list(object({
      name  = string
      value = string
    }))
  }))
  default = []
}
```

Implement and maintain state

```
variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [
    {
      internal = 8300
      external = 8300
      protocol = "tcp"
    }
  ]
}
```

Implement and maintain state

- Since terraform 0.13, there is also support for validation rules (which is very handy when developing modules):

```
variable "server_id" {
  type        = string
  description = "The id of the server."

  validation {
    condition     = length(var.server_id) > 4 && substr(var.server_id, 0, 4) == "srv-"
    error_message = "The server_id value must be a valid server id, starting with \"srv-\"."
  }
}
```

Implement and maintain state

- You can also use "sensitive" in input variables, to prevent terraform from outputting the variable during plan & apply

- This is useful if you’re handling secrets (user info, password information) in a variable:

```
variable "password" {
  type      = string
  sensitive = true
}

resource "resource_type" "resource_name" {
  parameter = var.password
}
```

```
Terraform will perform the following actions:

  # resource_type.resource_name will be created
  + resource "resource_type" "resource_name" {
      + parameter = (sensitive)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Implement and maintain state

- There's one exception: if your sensitive information becomes part of the identifier (id) of the resource, it'll be disclosed:

```
  # resource_type.resource_name will be created
  + resource "resource_type" "resource_name" {
      + id        = (known after apply)
      + parameter = (sensitive)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

...

resource_type.resource_name: Creating...
resource_type.resource_name: Creation complete after 0s [id=my-sensitive-information]
```

Configuration

- Output variables have a mandatory argument: value
- The following are **optional arguments**:
  - description
  - sensitive
  - depends_on

```
output "password" {
  value       = resource_type.resource_name.password
  description = "The password for logging in."
  sensitive   = true
}
```

Configuration

- In **rare cases** you need depends_on to depend on another resource, before outputting the value

- It works in the same way as "depends_on" in regular resources:

```
output "password" {
  value       = resource_type.resource_name.password
  description = "The password for logging in."
  sensitive   = true
  depends_on  = [resource_type.another_resource_name]
}
```

Configuration

- Local values can be useful to avoid repeating the same values
- I find it can also help you to **move some complexity away from the resource itself** for readability (when doing for/for_each loops for example)
- You should only use local values in moderation, as it can be a bit harder for maintainers to figure out where the values come from

Configuration

```
resource "azurerm_linux_virtual_machine" "group1" {
  count = var.group1_count
  …
}
resource "azurerm_linux_virtual_machine" "group2" {
  count = var.group2_count
  …
}

locals {
 vm_ids = concat(azurerm_linux_virtual_machine.group1.*.id, azurerm_linux_virtual_machine.group2.*.id)
}
```

```
resource "azurerm_key_vault_secret" "secret" {
  name  = "vm_ids"
  value = join(",", local.vm_ids)
  key_vault_id = azurerm_key_vault.example.id
}
```

Read, generate, and modify configuration

Resources and data sources

Configuration

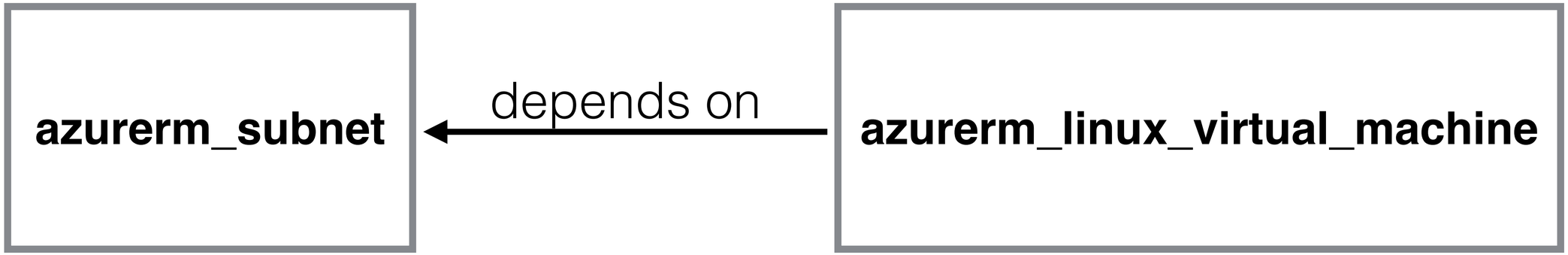
- In terraform, you can create "**resources**" and "**datasources**"
- Datasources allow data to be **fetched** or **computed** from **outside** of **terraform**
    - For example, an source images list that can be filtered to extract source images IDs, or the lookup of an external VPC to retrieve the VPC ID
- **Resources**, unlike datasources, describes one or more infrastructure objects
    - They typically create infrastructure components, like an VM resource, or a subnet, a database

Configuration

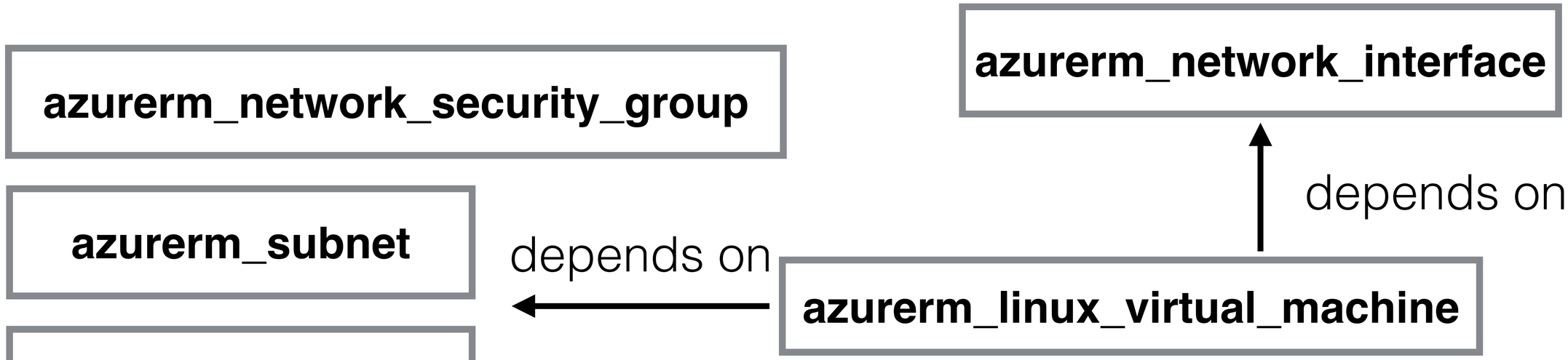
```
data "azurerm_image" "webapi" {
  name                = "webapi"
  resource_group_name = "companyimages"
}

output "image_id" {
  value = data.azurerm_image.webapi.id
}

resource "azurerm_linux_virtual_machine" "webapi-vm" {
  name            = "webapi-vm"
  source_image_id = data.azurerm_image.webapi.id
  […]
}
```

Configuration

```
data "azurerm_subnet" "my-subnet"
  name                 = "mysubnet1"
  resource_group_name = “networking"
}

resource "azurerm_linux_virtual_machine" "webapi-vm" {
  name             = "webapi-vm"
  […]
  subnet_id     = data.azurerm_subnet.my-subnet.id
}
```



Configuration




```
resource "azurerm_linux_virtual_machine" "webapi-vm" {
  […]
  network_interface_ids = [azurerm_network_interface.myinterface.id]
  […]
}

resource "azurerm_network_interface" "myinterface" {
  […]
  name                          = "myNIC"
  […]
}
```

Configuration

- If for some reason normal **dependency management** by terraform doesn't work, you can force a resource to depend on another resource
- This can be done using the keyword "**depends_on**"
- depends_on expects a list of other resource names:

```
resource "azurerm_linux_virtual_machine" "webapi-vm" {
  […]
  network_interface_ids = [azurerm_network_interface.myinterface.id]
  depends_on = [azurerm_network_interface.myinterface]
}
```

Configuration

- During terraform apply, terraform will:
  - **Refresh** the data sources
  - **Create resources** that exist in the *.tf files, but not in the state file
  - **Destroy resources** that exist in the state, but not in the *.tf files
  - **Update resources** that have different arguments in the *.tf files than on the cloud provider
    - **Destroy and re-create** resources that have arguments changed that require re-creation (for example a change to custom_data in an virtual machine always needs re-creation)
    - **In-place updates** are possible if the infrastructure API supports it (update to a security group for example)

Configuration

- Resources can be addressed using:
  - <RESOURCE TYPE>.<NAME>.<ATTRIBUTE>
- Data sources can be addressed using:
  - data.<RESOURCE TYPE>.<NAME>.<ATTRIBUTE>
- Local resources can be addressed using:
  - local.key_name

Configuration

```
resource "azurerm_linux_virtual_machine" "webapi-vm" {
  name              = "webapi-vm"
  source_image_id = data.azurerm_image.webapi.id
  os_disk {
    name                 = "myOsDisk"
    caching               = "ReadWrite"
    storage_account_type = "Premium_LRS" }
  […]

}
```

- azurerm_linux_virtual_machine id: azurerm_linux_virtual_machine.webapi-vm.id (but also azurerm_linux_virtual_machine.webapi-vm. name will work)

Configuration

- You can use functions in terraform to create all different sorts of behaviour
    - Numeric functions (min, max, …)
    - String functions (formatting strings)
    - Collection functions (merging lists, maps)
    - Encoding functions (base64, json, yaml)
    - Date and Time functions
    - Hash and crypto functions (uuid, SHA)
    - IP Network functions (subnet calculations)
    - Type conversion (tolist, tomap, toset, …)

Configuration

- For and for_each can help you to iterate over variables
  - count / for_each at a resource level can create multiple instances of a resource

```
resource "azurerm_linux_virtual_machine" "vm" {
  count                 = length(var.instances)
  […]
}
```

  - **Dynamic blocks** can iterate over **blocks**
  - For loops can iterate over complex data types and output a specific value (as a value of a parameter, or as a value in locals { … })

Configuration

```
locals {
  my_result = [ for x in var.y: x.id ]
}

resource "resource_type" "resource_name" {
  parameter = local.my_result
}
```

Terraform cloud

Terraform Cloud

- **Terraform cloud** is a HasiCorp product
- It helps teams use terraform together
- Instead of running terraform from your own machine, or on your own Jenkins, Terraform Cloud will run terraform **on their machines in a consistent and reliable environment**
- You have easy access to **shared state**, **version control** integration, **secret data**, **access controls** for approving changes to infrastructure, **policy controls** and other enterprise features
- It includes a **private terraform registry** to share terraform modules

Terraform Cloud

- Terraform cloud is hosted at https://app.terraform.io/
- You can create a **free account** for small sized teams
- There are **paid plans** for medium size businesses
- For large enterprises terraform has “**Terraform Enterprise**”, which is the self-hosted version of Terraform cloud (to run within your own environment)

Terraform Cloud

- Terraform Cloud **workspaces are different** than the local terraform workspaces

- When you locally use workspaces, you're still in the **same directory**, using the **same variables and credentials**. The state is empty for every new workspace, but the state is just another file within the same project

- With Terraform Cloud Workspaces, it's much more isolated. It's much more like a separate "project" with its own variables, secrets, credentials, and state

    - The state also supports multiple versions, so you can see the previous state versions, and how they match with a specific terraform run

- Terraform recommends to use workspaces in Terraform Cloud to split your monolithic terraform project in smaller projects, for example split out networking, different apps

Terraform Cloud

- Terraform **Sentinel** is a **paid feature**, available in Terraform Cloud
- Sentinel is an **embedded policy-as-code framework** integrated with the other HashiCorp Enterprise products
- Sentinel allows administrators to write **policy rules** to put **controls** in place to **protect or restrict** what can be **applied to the infrastructure**

Terraform Cloud

- A few use cases:
  - Enforce that every resource needs to be tagged
  - Restrict App Service to use https
  - Restrict VM Size
  - Restrict VM image ID
  - Enforce AKS node pools max_count
  - Restrict inbound source address

Configuration

```
# This policy uses the Sentinel tfplan/v2 import to require that
# all Azure VMs have vm sizes from an allowed list

# Import common-functions/tfplan-functions/tfplan-functions.sentinel
# with alias "plan"
import "tfplan-functions" as plan

# Allowed Azure VM Sizes
# Include "null" to allow missing or computed values
allowed_sizes = ["Standard_A1", "Standard_A2", "Standard_D1_v2", "Standard_D2_v2"]

# Get all Azure VMs using azurerm_virtual_machine
allAzureVMs = plan.find_resources("azurerm_virtual_machine")

# Filter to Azure VMs with violations that use azurerm_virtual_machine
# Warnings will be printed for all violations since the last parameter is true
violatingAzureVMs = plan.filter_attribute_not_in_list(allAzureVMs,
                    "vm_size", allowed_sizes, true)

# Main rule
violations = length(violatingAzureVMs["messages"])

main = rule {
  violations is 0
}
```

Azure Services

Azure Database for MySQL

Azure Database for MySQL

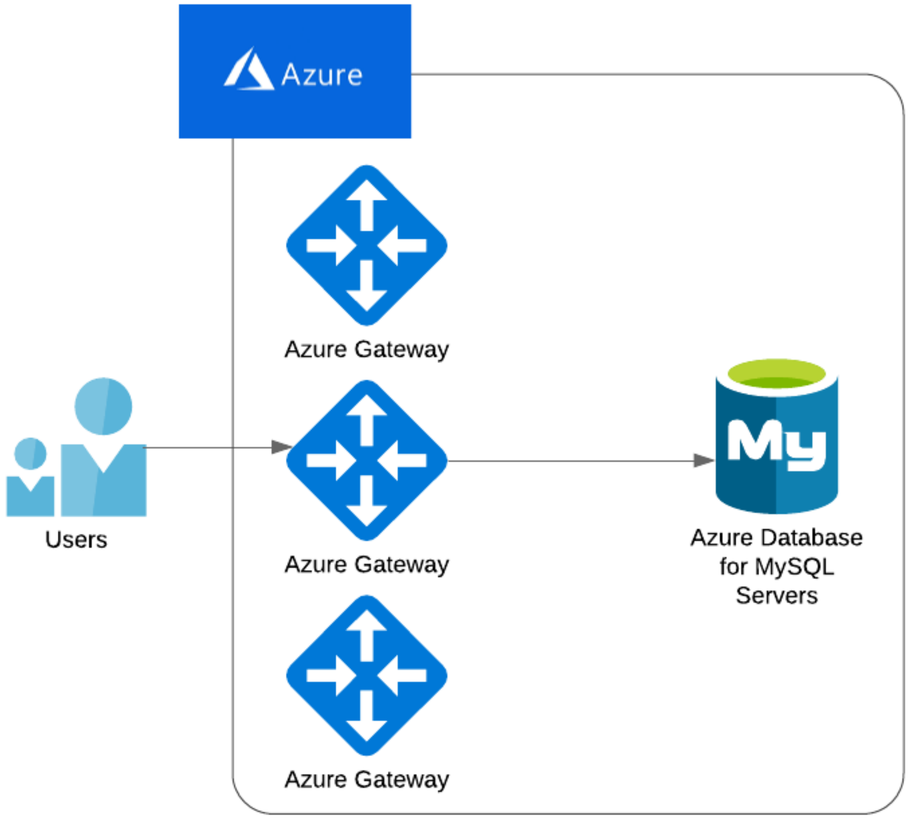
- MySQL
  - MySQL is an **open-source** relational database management system.
- Azure Database for MySQL is a **managed service**
  - Automatic database patching, automatic backups, built-in monitoring, security
- **High availability**
  - Scale **highly available**
    - **Application retry logic is essential**

Azure Database for MySQL

- Information protection and encryption
  - In-transit
  - At-rest

Azure Database for MySQL

- Network security
  - Connectivity architecture
  - IP firewall rules
  - Virtual network firewall rules



DEMO

Azure Database for MySQL

Azure Database for MySQL

- Create an Azure Database for MySQL
- Configure the MySQL service firewall
- Use Virtual Network (VNet) service endpoints
- Create virtual machine
- Use MySQL command-line tool to test connection to a database

Azure Database for Azure SQL

Azure Database for Azure SQL

- Azure SQL
  - **Azure SQL Database** is a **fully managed database engine**.
  - Handles almost all of the database management functions like: **upgrading, patching, backups, and monitoring**
  - Azure SQL Database is always running on the **latest stable version** of SQL Server Database Engine

Azure Database for Azure SQL

- Azure SQL
  - **Azure SQL Database** is a **fully managed database engine**.
  - Handles almost all of the database management functions like: **upgrading, patching, backups, and monitoring**
  - Build in business continuity and global scalability
  - Azure SQL Database is always running on the **latest stable version** of SQL Server Database Engine

Azure Database for Azure SQL

- Build in business continuity and global scalability
  - Automatic backups
  - Point-in-time restores
  - Active geo-replication
  - Auto-failover groups
  - Zone-redundant databases

Azure Database for Azure SQL

- Deployment models
  - **Azure SQL Database**
    - A single fully managed and isolated database
  - **Elastic pools**
    - A collection of single databases with shared resources
  - **Managed instance**
    - Fully managed instance of SQL Server on-premises (Enterprise Edition)
    - **Instance pools**

SQL Databases


Elastic Database Pools

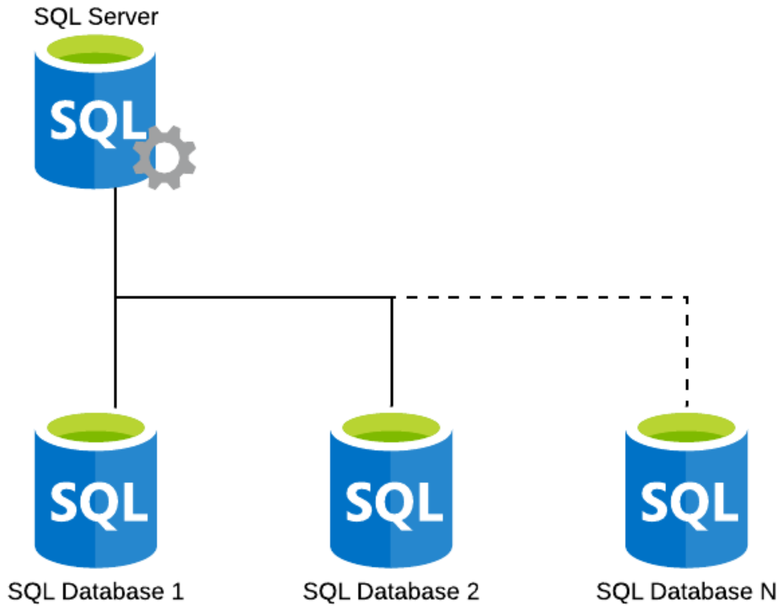
SQL Managed Instances

Azure Database for Azure SQL

- **Elastic Pool**
  - **Without**
    - Over or under provision
  - **With**
    - Good for unpredictable load (SaaS environments)
    - Cost-effective
    - Performance elasticity

Azure Database for Azure SQL

- **SQL Database** deployment consists out of 2 components
  - **SQL Database server** (central administrative point)
  - 1 or more **SQL Database(s)**



Azure Database for Azure SQL

- Pricing models
  - **vCore** (Virtual Core)
    - Provisioned compute
    - Serverless (vCore cost more then provisioned compute)
  - **DTU** (Database Transaction Unit)
    - Compute and storage will scale together

Azure Database for Azure SQL

- Want to know what version to use?
  - DMA is there to help you

DEMO

Azure Database for Azure SQL

Azure Database for Azure SQL

- Create an Azure Database for Azure SQL
- Configure the Azure SQL service firewall
- Use Virtual Network (VNet) service endpoints
- Create virtual machine
- Use SQL command-line tool to test connection to a database

Azure Cosmos DB

Azure Cosmos DB

- Cosmos DB
  - Azure **Cosmos DB** is Microsoft's **globally distributed**, **multi-model** database service.
  - **Cosmos DB** enables you to scale throughput and storage across any number of Azure regions worldwide

Azure Cosmos DB

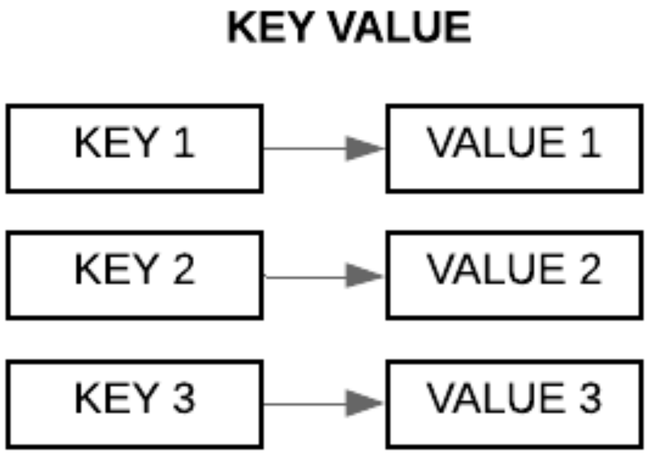
- Features
  - **Turnkey global distribution** - transparent multi region
  - **Regional presence** - available in all regions
  - **Elastic scale** - up to hundreds millions of requests/s
  - **Guaranteed low latency at the 99th percentile** - under 10ms
  - **Comprehensive SLA's** - SLA missed = money back
  - **5 Well defined consistency models**
  - **No Schema or index management needed**

Azure Cosmos DB

- Data access using your favourite tools **SQL, MongoDB, Cassandra, Tables, Gremlin or etcd.**

Azure Cosmos DB

- Key Value



Azure Cosmos DB

- Column family

COLUMN FAMILY

TYPE	CAR	COLOR
A4	AUDI	SILVER
3 SERIES	BMW	BLACK
...	...	...

Azure Cosmos DB

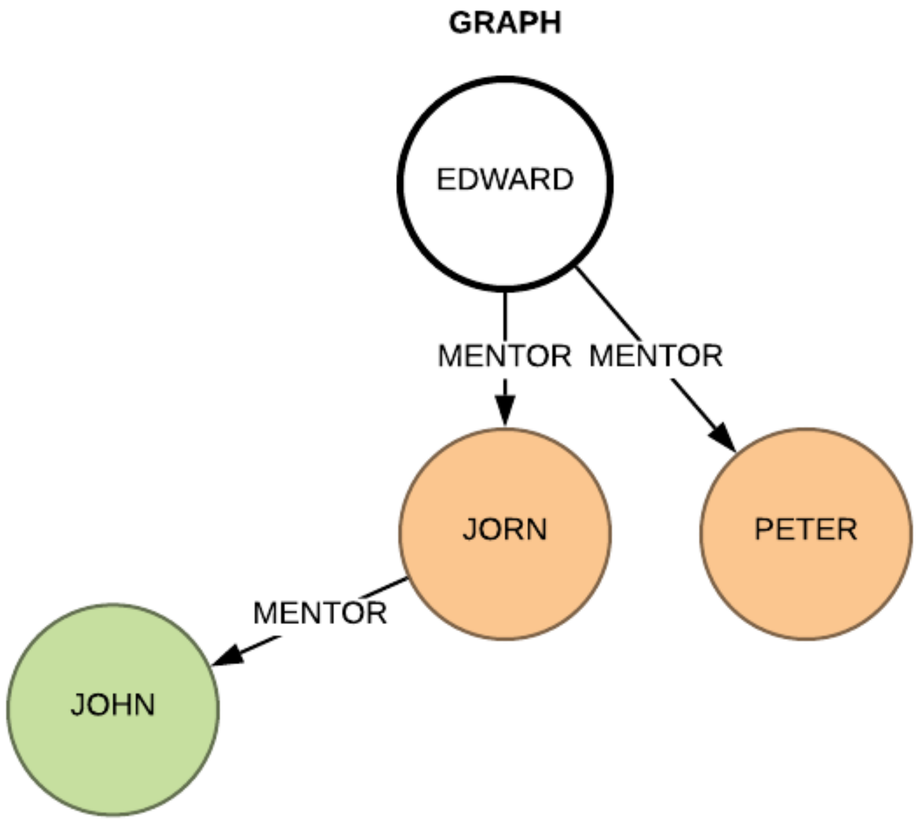
- Document

DOCUMENT

```
{
  "_id": 1,
  "name" : { "first" : "John", "last" : "Doe" },
  "contribs" : [ "Golang", "Java", "PHP", "C++" ],
  "awards" : [
    {
      "award" : "Super DevOps",
      "year" : 2018,
      "by" : "DevOps Acedemy"
    }, {
       "award" : "Super DevOps",
      "year" : 2019,
      "by" : "DevOps Acedemy"
    }
  ]
}
```

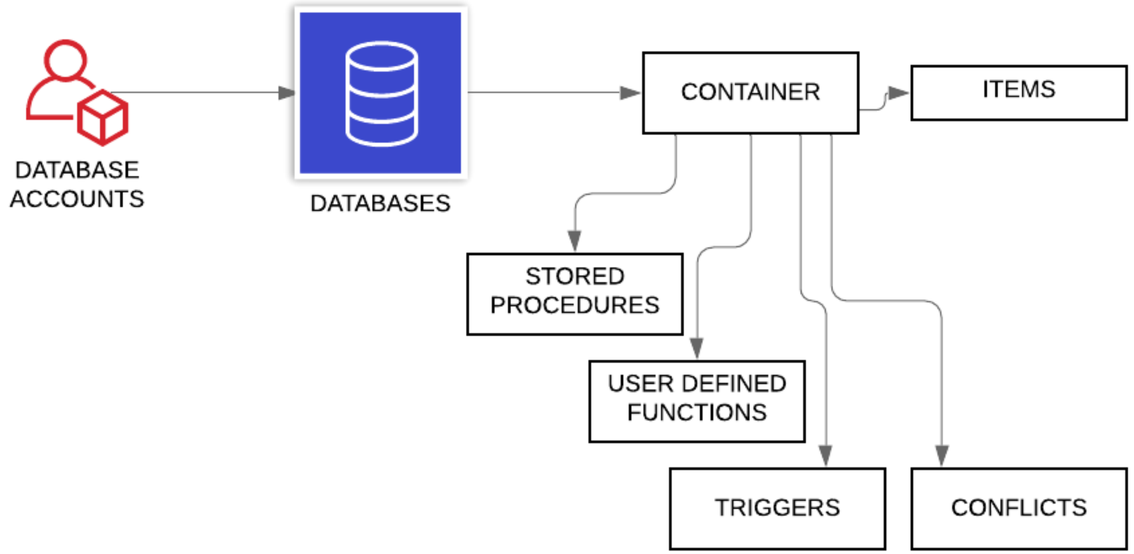
Azure Cosmos DB

- Graph



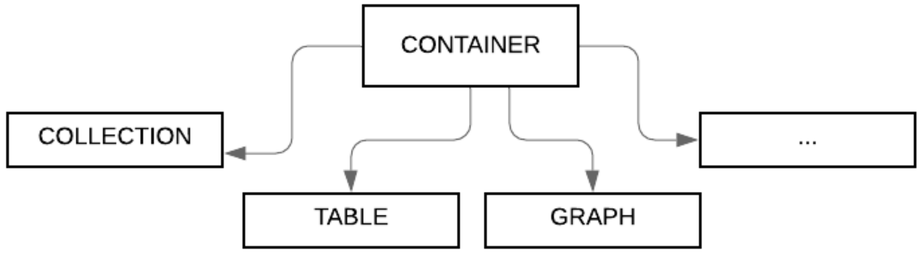
Azure Cosmos DB

- DB structure



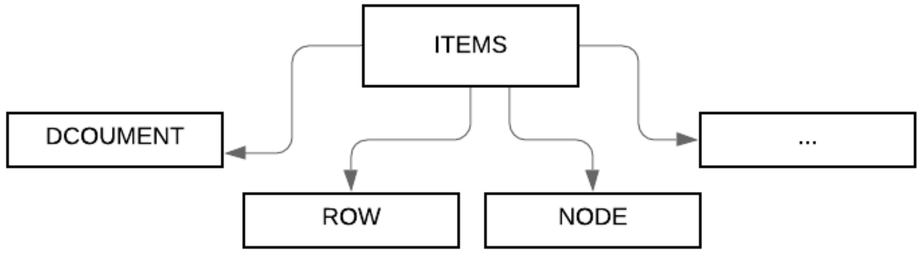
Azure Cosmos DB

- Container structure



Azure Cosmos DB

- Items structure



Azure Cosmos DB

- Request units
  - Clear way to provision
  - Pay for resources used
- RUs considerations
  - Item size, item indexing, item property count, indexed properties, data consistency, query patterns and script usage

Azure Cosmos DB

- Consistency models
  - **Strong**
  - **Bounded-stateless**
  - **Session**
  - **Consistent prefix**
  - **Eventual**

DEMO

Azure Cosmos DB

Azure Cosmos DB

- Create an Azure Cosmos DB
- Configure the Azure Cosmos DB service firewall
- Use Virtual Network (VNet) service endpoints
- Create virtual machine
- Use mongodb command-line tool to test connection to a database

Azure Storage

Azure Storage

- What is Azure Storage?
  - Modern solution
  - Virtually limitless
  - Pay-as-you-go
  - Clients: .NET, Ruby, Java, …
  - Access to Azure Storage via the Azure storage Account

Azure Storage

- Azure storage features:
  - **Durable and high availability**
  - **Scalability**
  - **Security**
  - **Accessibility**

Azure Storage

- Azure storage Types:
  - **Azure blob storage** - Unstructured data, BLOB's
  - **Azure table storage** - Part of Azure CosmosDB, NoSQL data
  - **Azure file storage** - Managed file sharing, SMB
  - **Azure queue storage** - Message storage queue, HTTP(S)
  - **Azure Data Lake Storage Gen2** - big data analysis
  - **Disk storage** - Managed(behind the hood) and unmanaged disks (VHD)

Azure Storage

- Azure Storage redundancy
  - Locally redundant storage (LRS)
  - Zone-redundant storage (ZRS)
  - Geo-redundant storage (GRS)
    - Read-access geo-redundant storage (RA-GRS)
  - Geo-zone-redundant storage (GZRS)
    - Read-access geo-zone-redundant storage (RA-GZRS)

Azure Storage Account

Azure Storage Account

- Storage account
  - Collection of all storage data objects
  - Unique namespace
  - Accessible for anywhere
  - Highly available
  - Secure
  - Super Scalable

Azure Storage Account

- Types
  - General-purpose v2 - **Recommended**
  - General-purpose v1 - **Legacy**
  - BlockBlobStorage - For **high, low latency** transaction scenarios
  - FileStorage accounts - For **high performance** scale applications.
  - BlobStorage accounts - **Legacy Blob-only** storage

Azure Blob Storage

Azure Blob Storage

- **Azure blob storage**
  - Blob = Binary Large Object
  - Azure Blob is a service that stores unstructured data
  - Accessed from anywhere via HTTP(S)

Azure Blob Storage

- **When use Azure blob storage?**
  - Storing files for shared access
  - Video and audio streaming
  - Storing data for analysis
  - Writing log files
  - Storing data for disaster recovery, backup, and archiving

Azure Blob Storage

- **Types of BLOB storage**
  - **Block blobs**
    - Store text and binary data, up to 4.7TB
  - **Append blobs**
    - Optimized for append operations as loggin data(app/webserver/…)
  - **Page blobs**
    - Random access files up to 8TB, VHD

Azure Blob Storage

- **Access tiers (Only for Generale Purpose v2 storage accounts)**
  - Hot - Expected to be accessed R/W frequently
  - Cool - Short-term backup and disaster recovery datasets
  - Archive - Long-term backup, secondary backup, and archival datasets

DEMO

Azure Blob Storage

Azure Blob Storage

- Create a storage account
- Create a container
- Upload file using Terraform
- Create virtual machine
- Download file using Role-based access control (RBAC) & access token

Azure Table Storage

Azure Table Storage

- **Azure table storage**
  - NoSQL
  - Schemaless
  - Scalable - Store TB's of data
  - Good for:
    - Data without joins, foreign keys
  - Protocol - OData, LINQ

Azure File Storage

Azure File Storage

- **Azure file storage**
  - Fully managed file share
  - SMB
  - Shares can be mounted concurrently
  - Ability to be cached by Windows Server with Azure File Sync
  - Good for:
    - Replacing local file servers, Lift and shift applications

Azure Queue Storage

Azure Queue Storage

- **Azure queue storage**
  - Ability to scale at bursts
  - Build-in resilience
  - Good for: Decoupling services
  - Data accessible via the REST API
  - Clients: .NET, Java, Android, C++, Node.js, PHP, Ruby, and Python

Azure Disk Storage

Azure Disk Storage

- **Disk storage**
  - **Unmanaged** disks - **Legacy**
    - Storage account necessary
  - **Managed** disks - **Recommended**
    - **No** storage account necessary

Azure Disk Storage

- **Managed** disks features
  - Encryption
  - Disk roles - Data disk, OS disk, Temporary disk
  - Managed disk snapshots / images

Azure Disk Storage

- **Managed** disks **benefits**
  - Highly durable and available
  - Simple and scalable VM deployment
  - Integration with availability sets
  - Integration with Availability Zones
  - Azure Backup support
  - Granular access control

Azure Disk Storage

- **Considerations of disk usage**
  - Cached vs uncached
  - Use the correct type of vm and storage tier

Size	vCPU	Memory: GiB	Temp storage (SSD) GiB	Max temp storage throughput: IOPS / Read MBps / Write MBps	Max data disks / throughput: IOPS	Max NICs / Expected network bandwidth (Mbps)
Standard_A1_v2	1	2	10	1000 / 20 / 10	2 / 2x500	2 / 250

Azure Data Lake Storage Gen2

Azure Data Lake Storage Gen2

- **Azure Data Lake Storage Gen2**
  - Build on **Azure blob storage**
  - **Combination** of: **Azure Blob storage** and **Azure Data Lake Storage Gen1**
  - Designed to service **multiple petabytes**
  - Designed to deliver **hundreds of gigabits** of throughput
  - Support for HDInsight, Hadoop, Cloudera, Azure Databricks, Hortonworks

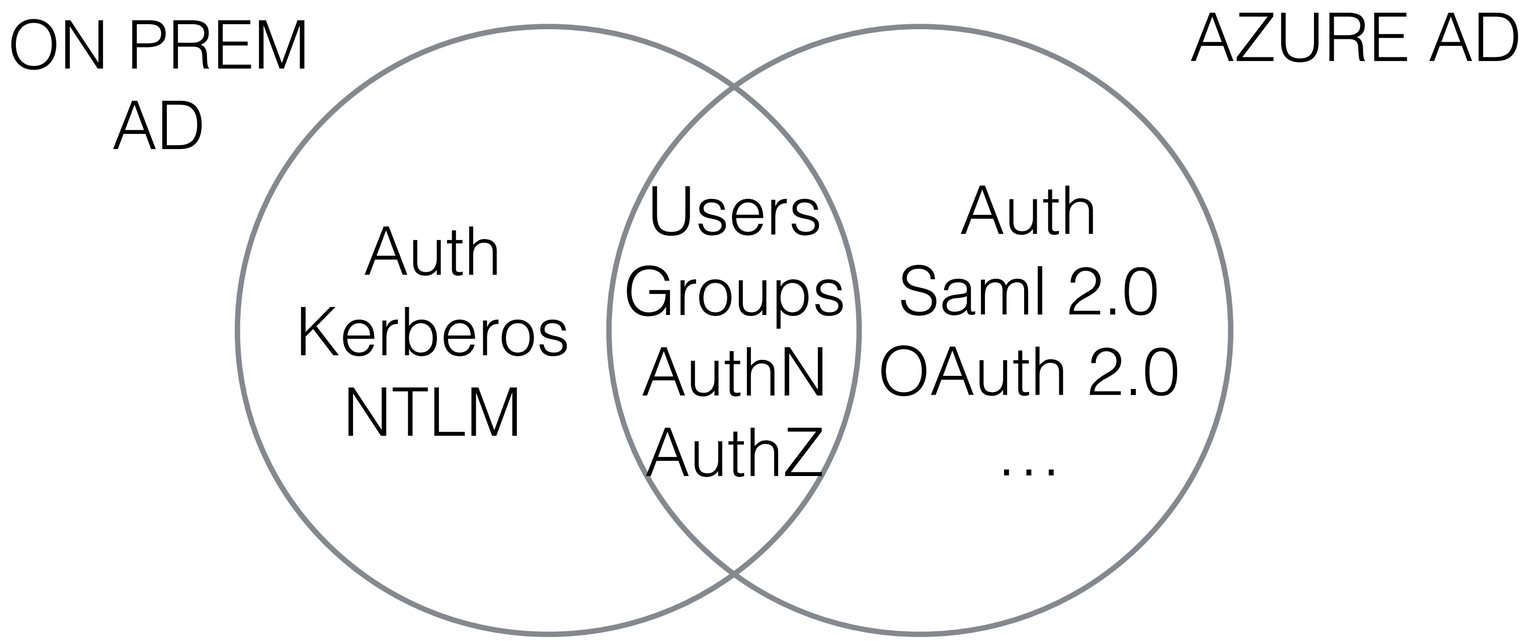
Azure AD

Azure AD

- What is Azure Active Directory?
  - Cloud-based identity and access management
  - Pay as you go
  - For:
    - External - Azure Portal, Office 365 and thousands of SaaS applications
    - Internal - Custom developed apps / intranet / ..

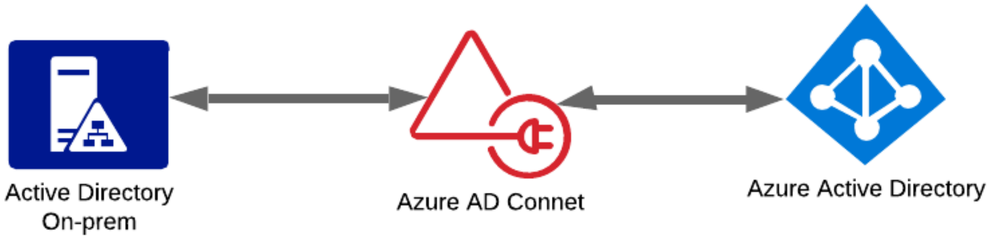
Azure AD

- What is Azure Active Directory **not**?
  - Azure AD is **not** a replacement for Windows Server Active Directory.



Azure AD

- What is Azure AD Connect?
  - Is a tool for connecting on premises identity infrastructure to Microsoft Azure AD



Azure AD

- Azure Active Directory **B2C**
  - Mange customers sign up /sign in
  - Mainly for custom public applications

Azure AD

- Azure Active Directory **B2B**
  - Good for “guests”
  - Share company apps with other orgs
  - Remain in control of your corporate data

Azure AD

- Azure Active Directory **Terraform Provider**
  - Infrastructure Azure Active Directory
  - Azure Resource Manager API's
- **Mangages** Azure AD:
  - Applications
  - **Service principals**
  - Groups / Users

DEMO

Azure AD

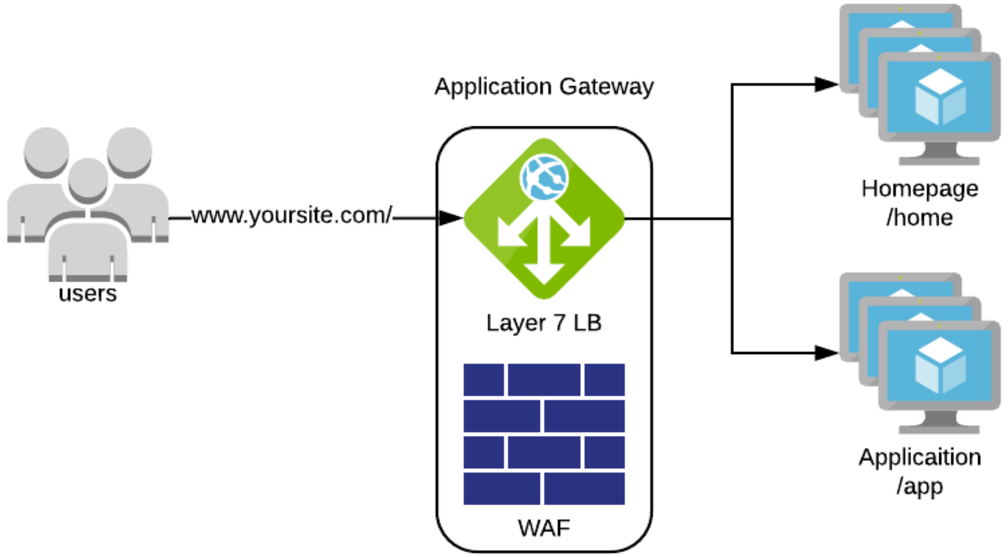
Azure AD

- Create an Azure Active Directory Application
- Create a Service Principal
- Create a Password for that Service Principal
- Output all the needed info

Application Gateway

Application Gateway

- What is an Azure Application Gateway?
- Main Features
  - **Web traffic load balancer**
  - **Operates on the OSI layer 7**
  - **SSL/TLS termination**
  - **Redirecting**



Application Gateway

- **Request based routing**
  - Routing decisions based on additional attributes in an HTTP request
    - URI path (ex: /home)
    - Host Headers (ex: Host: www.example.com)
      - This allows the Application Gateway to be used for multiple domains

Application Gateway

- **WAF (Web application firewall)**
  - Central protection against common exploits, bots and scanners
  - SQL injection protection
  - Cross site scripting attacks
- Application protection code can be challenging
- Helps making security management much easier to maintain/update
- WAF helps remediate threats faster

Application Gateway

- **Autoscaling**
  - Scale up or down based on traffic
- **Zone redundancy**
- **Static VIP**
- **Ingress Controller for AKS**

Application Gateway

- **Session affinity**
- **Connection draining**
- **Custom error pages**
- **Rewrite HTTP headers**
- **Sizing (Small, Medium, and Large)**
  - Example:

Average back-end page response size	Small (dev/test)	Medium	Large
6 KB	7.5 Mbit/s	13 Mbit/s	50 Mbit/s
100 KB	35 Mbit/s	100 Mbit/s	200 Mbit/s

Application Gateway

- **Pros**
  - Simple
  - **Protection against threats** via the **WAF** (if enabled)
  - Public/Private load balancing
  - **SSL/TLS Termination**
  - **Custom health probes**

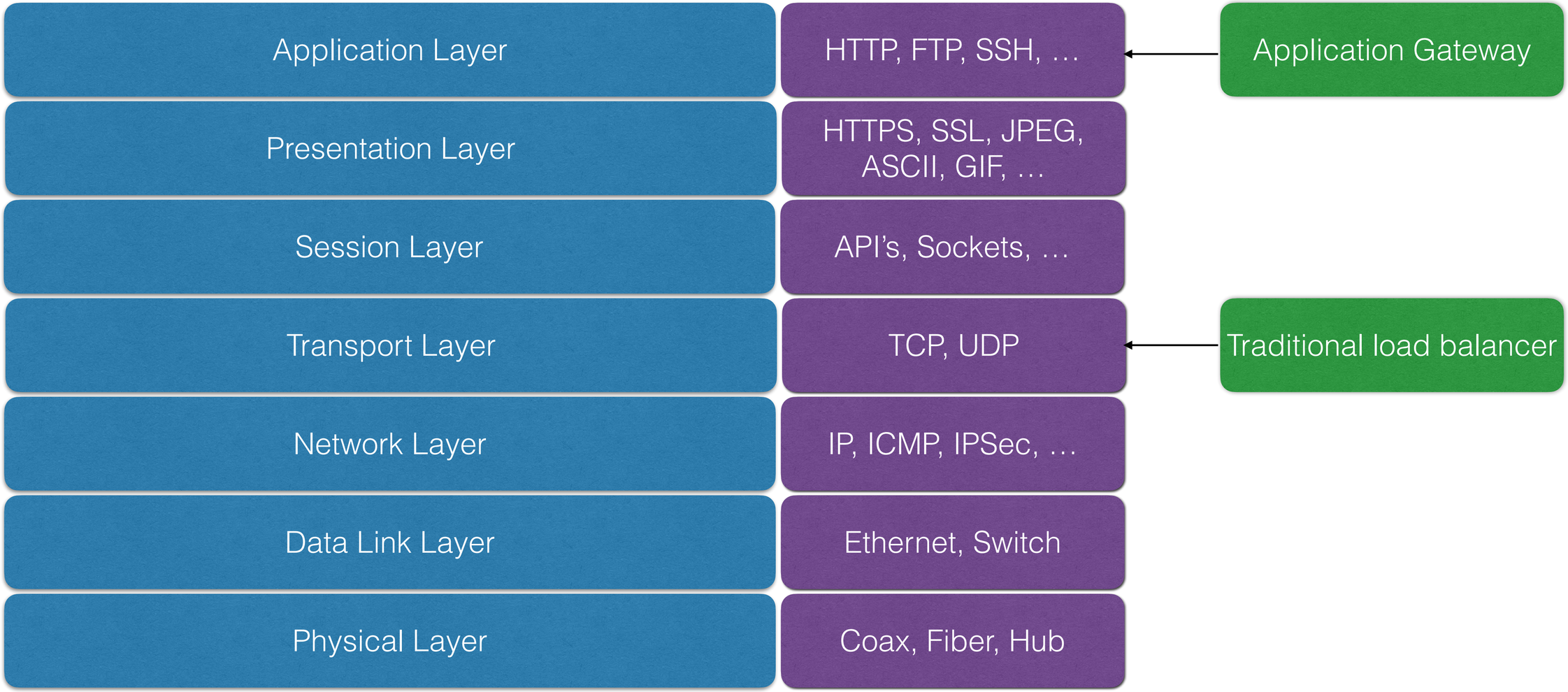
Application Gateway

- **Differences** with a traditional load balancer
  - Traditional load balancers operate on **OSI layer 4**
  - Application gateways operate on the **OSI layer 7**

- **OSI?**

Application Gateway

- OSI (Open Systems Interconnection)



DEMO

Application Gateway

Application Gateway

- Create an Azure Application Gateway
- Extra subnet
- Security group
- Create a scale set
  - Install Nginx using a script automatically pulled from Github

Azure Stream Analytics

Azure Stream Analytics

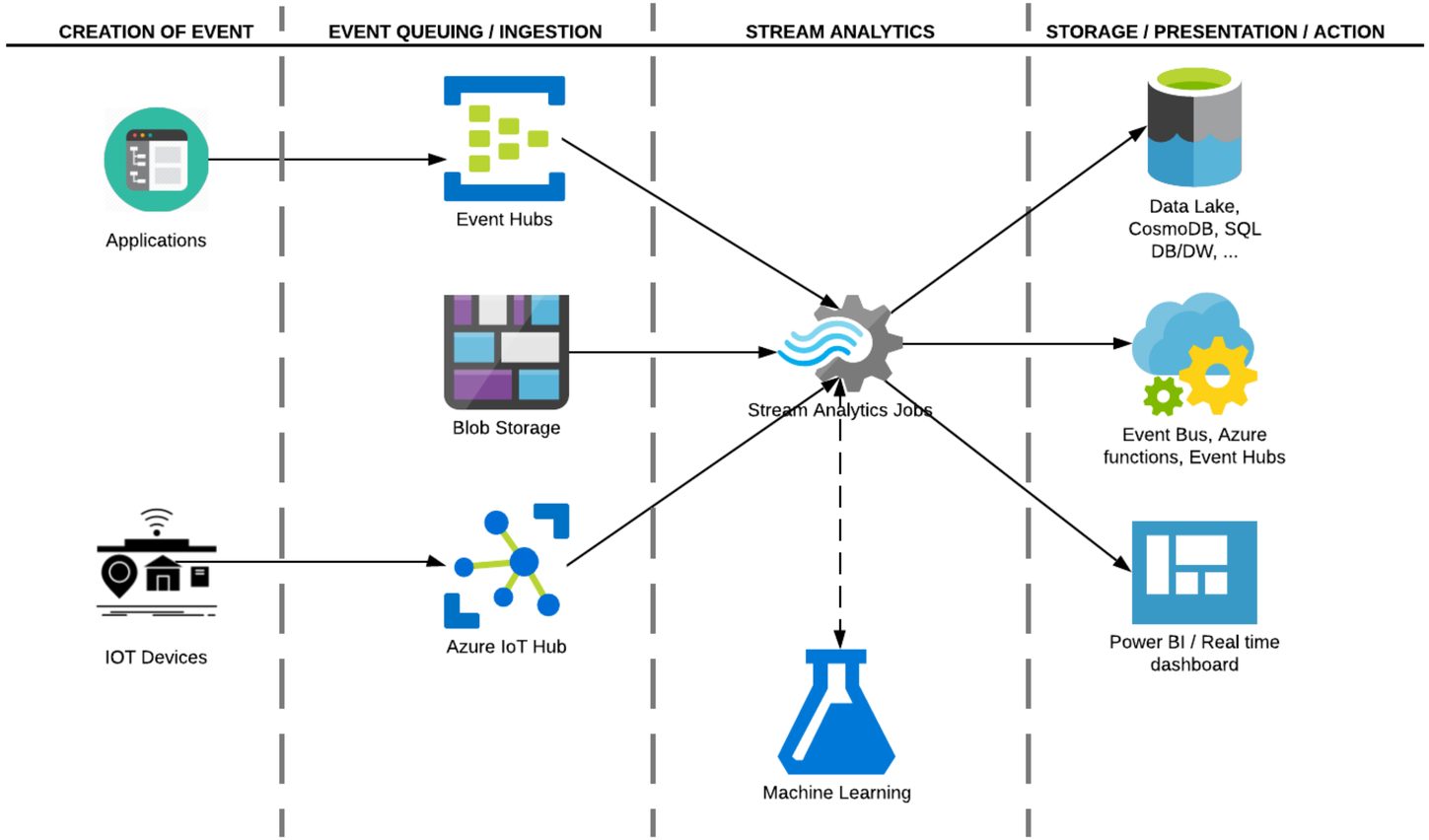
- **What is Azure Stream Analytics?**
  - **Real-time analytics** and **complex event-processing** engine
    - **Analyze and process** high volumes of **fast streaming data**
    - Can have multiple **simultaneously sources:**
      - **Sensors, clickstreams, social media feeds, applications, …**
- **Benefits**: Source/sink integration, SQL like query, Serverless, Scalable and fully managed.

Azure Stream Analytics

- **Why use Azure Stream Analytics?**
  - Reporting
  - Analyze logs
  - Trigger **actions/alerts** on certain data patterns
    - Remote monitoring and predictive maintenance
    - Fraud detection
  - Real time sales data
  - **Storing** transformed data for later use (regulations, batch, …)

Azure Stream Analytics

- **Needed components:** Input -> Query -> output



Azure Stream Analytics

- **Query language:**
  - **SAQL** (Stream Analytics Query Language)
  - **Powerfull** query language
    - Built for **data analysis**
    - Aggregate Functions, Analytic Functions, Array Functions, GeoSpatial Functions, Input Metadata Functions, Record Functions, Windowing Functions, Scalar Functions

Azure Stream Analytics

- **SAQL examples:**

```
    SELECT *
    INTO BlobOutput
    FROM IotHub
    HAVING Temperature > 25
```

```
WITH AnomalyDetectionStep AS
(
    SELECT
        EVENTENQUEUEDUTCTIME AS time,
        CAST(temperature AS float) AS temp,
        AnomalyDetection_SpikeAndDip(CAST(temperature AS float), 95, 120,
'spikesanddips')
            OVER(LIMIT DURATION(second, 120)) AS SpikeAndDipScores
    FROM input
)
SELECT
    time,
    temp,
    CAST(GetRecordPropertyValue(SpikeAndDipScores, 'Score') AS float) AS
    SpikeAndDipScore,
    CAST(GetRecordPropertyValue(SpikeAndDipScores, 'IsAnomaly') AS bigint) AS
    IsSpikeAndDipAnomaly
INTO output
FROM AnomalyDetectionStep
```

```
WITH sentiment AS (
SELECT text, sentiment1(text) as result
FROM datainput
)

SELECT text, result.[Score]
INTO datamloutput
FROM sentiment
```

DEMO

Azure Stream Analytics

Azure Stream Analytics

- Stream Analytics job
- Real-time data and filter messages (temperature > 25)
- Write output to blob storage
- Input generated by Raspberry Pi simulator

Advanced Terraform

Remote State

Remote State on Azure

- What is Terraform state?
  - By **default**, Terraform stores state **locally**
  - Terraform writes the state data to a **remote data store**
  - Remote state is a feature of **backends**

Remote State on Azure

- Benefits of remote state:
  - Working in a **team**
  - Keeping **sensitive information** off your local disk
  - **Remote** operations

Remote State on Azure

- **Locking** while running Terraform
  - Azure backend supports locking
  - Automatically locked
  - Prevents concurrent state operations
- **Encryption at rest** in Azure Blob Storage

DEMO

Remote State

Remote state on Azure

- Configure remote state using the Azure CLI
  - Resource group
  - Storage account
  - Storage container
  - Vault Key
- Configure Terraform to use this as remote state using the **backend**
- Spin up virtual machine and inspect the state file

Functions

Terraform Functions

- What are **Terraform functions**?
  - Vast amount of **built-in functions**
  - No support for user defined functions
  - Use “**terraform console**” to experiment
    - Examples:

```
$ terraform console
> max (10,20,30,2)
30
```

Terraform Functions

- Overview of all **Terraform functions**
    - https://www.terraform.io/docs/configuration/functions.html

Terraform Functions

- **Numeric** Functions
  - Examples:

```
$ terraform console
> max (10,20,30,2)
30
```

```
$ terraform console
> ceil(21.1)
22
```

Terraform Functions

- **String** Functions
  - Examples:

```
$ terraform console
join(", ", ["test", "training", "vm"])
test, training, vm
```

```
$ terraform console
> substr("Terraform", 1, 4)
erra
```

```
$ terraform console
> lower("TERRAform Training")
terraform training
```

Terraform Functions

- **Collection** Functions
  - Examples:

```
$ terraform console
> sort(["d", "a"])
[
  "a",
  "d"
]
```

```
$ terraform console
> lookup({a="test", b="terra"}, "a", "default")
test
> lookup({a="test", b="terra"}, "c", "default")
default
```

Terraform Functions

- **Encoding** Functions
  - Examples:

```
$ terraform console
> base64encode("Training")
VHJhaW5pbmc=
```

```
$ terraform console
> base64decode("VHJhaW5pbmc=")
Training
```

Terraform Functions

- **Filesystem** Functions
  - Examples:

```
$ echo test > file.txt
$ terraform console
> file("${path.module}/file.txt")
test
```

```
$ echo test > file.txt
$ terraform console
> fileexists("${path.module}/file.txt")
true
```

Terraform Functions

- **Date and Time** Functions
  - Examples:

```
$ terraform console
> formatdate("MMM DD, YYYY", "2020-01-11T00:00:00Z")
Jan 11, 2020
```

```
$ terraform console
> timeadd("2020-01-11T00:00:00Z", "35m")
2020-01-11T00:35:00Z
```

Terraform Functions

- **Hash and Crypto** Functions
  - Examples:

```
$ echo test > file.txt
$ terraform console
> md5(file("file.txt"))
d8e8fca2dc0f896fd7cb4cb0031ba249
```

Terraform Functions

- **IP Network** Functions
  - Examples:

```
$ terraform console
> cidrnetmask("10.0.0.0/16")
255.255.0.0
```

Terraform Functions

- **Type Conversion** Functions
  - Examples:

```
$ terraform console
> tomap({"traning" = 1, "terraform" = 2})
{
  "terraform" = 2
  "training" = 1
}
```

Demo

Terraform functions

Conditionals

Terraform Conditionals

- **Terraform Conditionals**?
  - If-Then-else
  - Introduced to get rid of some limitations
    - **Prior to v0.12** this was **only for primitive types**(aka not lists or maps)
    - **Since v0.12** this has been **resolved**
  - Format: **condition ? true_val : false_val**

Terraform Conditionals

- **Conditionally Omitted Arguments**
  - **Unset** a variable
  - “**Null**” value
  - Retain **default behavior**
    - **Avoid errors** when a variable is not set (useful in modules)
    - **Default** value will be used

```
variable "override_label" {
  type    = map
  default = null
}

resource "azurerm_network_security_group" "web" {
  name                = "webservers"
  location            = "West US"
  resource_group_name = azurerm_resource_group.demo.name
  tags                = var.override_label
 }
```

Demo

Terraform Conditionals

Demo

For loops

Demo

For each loops

AKS - Azure Kubernetes Service

AKS

- **Azure Kubernetes Service (AKS)** is Azure's fully managed Kubernetes offering
- Kubernetes is a **Container Orchestrator**, it allows you to run (docker) containers
- AKS **integrates** with all the other **Azure services**, so that you don't have to setup services like logging and networking yourself
- With terraform, you can easily set-up an AKS cluster
- In the first demo after this lecture, I'll show you how to setup a simple AKS cluster

AKS - deploying your app

- Azure has the capability to create a **CI/CD pipeline to build**, **test** and **deploy** your application on **Kubernetes**
- This is a separate service, called **Azure DevOps**
- Azure DevOps allows you to **integrate the Container Registry** (where the container images are stored) and **Kubernetes**
- Using your application code from git, you can **build** your docker container, **push** it to the container registry, and instruct the pipeline to **deploy** it on your Kubernetes cluster
- This all sounds really good, but **terraform support for Azure DevOps is lacking** at the time when I created this lecture - the demo to deploy the app will not be written completely in terraform until there is full terraform support (and at that point I'll redo the demo)

Demo

AKS

Demo

Deploying an application on AKS