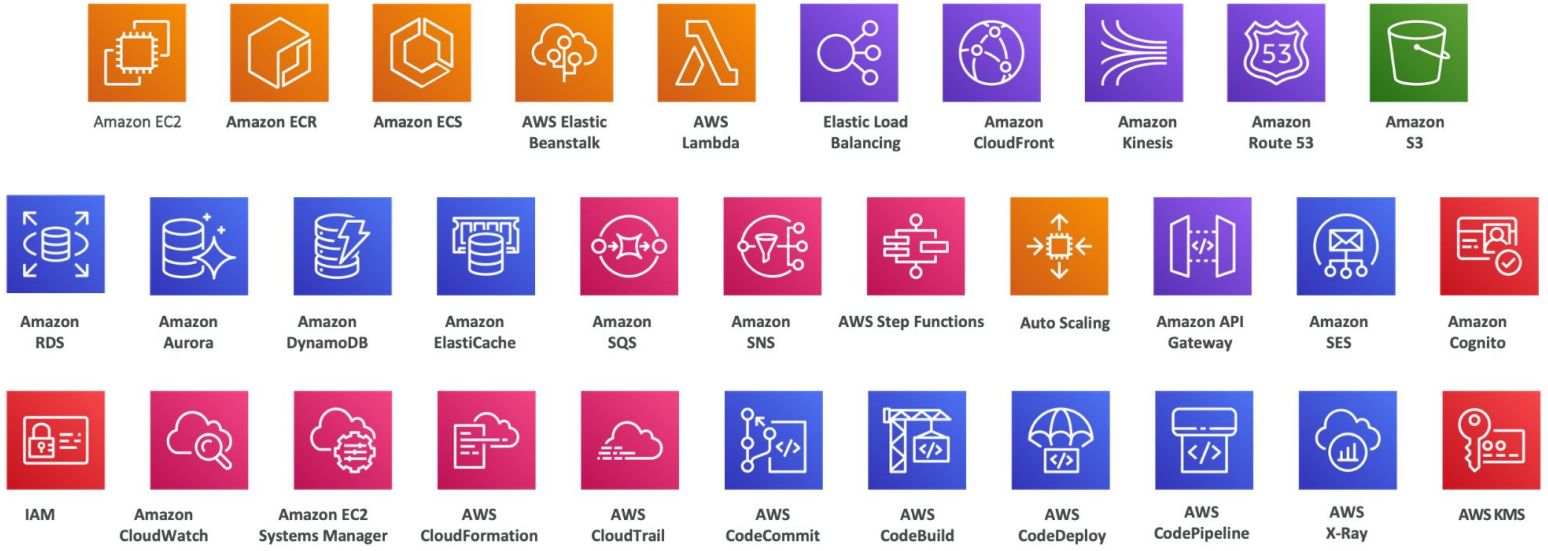
AWS Introduction

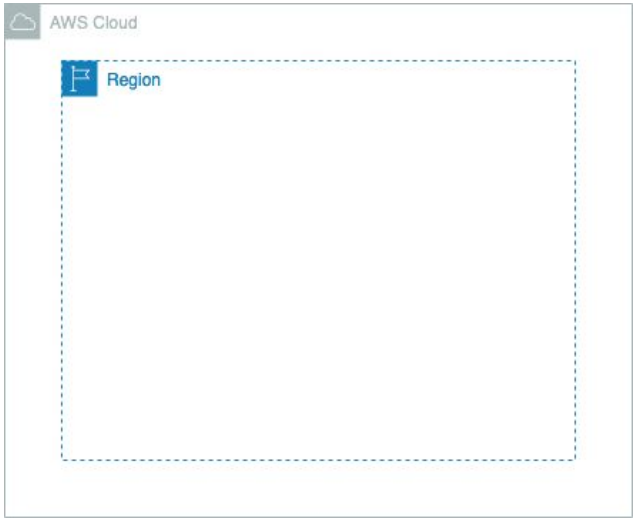
arconsis

AWS main components



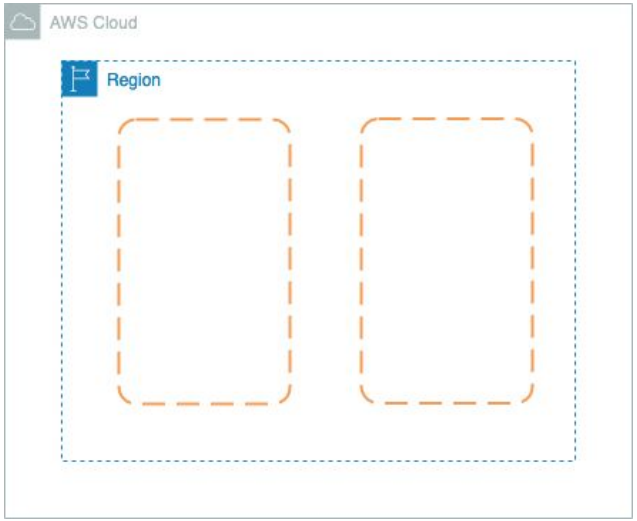
AWS Regions

- Designed to be **isolated** from other Amazon Regions
- Achieve the greatest possible **fault tolerance** and **stability**
- Most AWS Resources are **tied** to the Regions except some Global Services like Identity and Access Management (IAM)
- For example, we may want to launch instances in the EU to be near European customers or to meet legal requirements

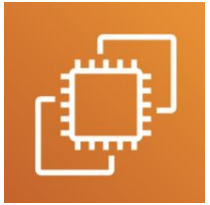


AWS Availability Zones (AZ)

- Availability Zones are multiple, **isolated locations** within each **Region**
- Represented by a Region code followed by a letter identifier; for example, `eu-central-1a`
- Consist of one or more discrete **data centers**, each with redundant power, networking, and connectivity
- Offer the ability to operate applications that are more highly available, fault tolerant, and scalable



Amazon EC2



- EC2 = Elastic Compute Cloud = Infrastructure as a Service
- You can use Amazon EC2 to **launch** as many or as few virtual servers as you need, configure security and networking, and manage storage
- Knowing EC2 is **fundamental** to understand how the Cloud works
- Operating System (**OS**): Linux, Windows or Mac OS
- How much compute power & cores (**CPU**)
- How much random-access memory (**RAM**)
- How much **storage** space
- **Network** card: speed of the card, Public IP address

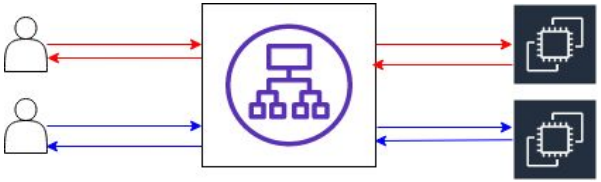
EC2 Types

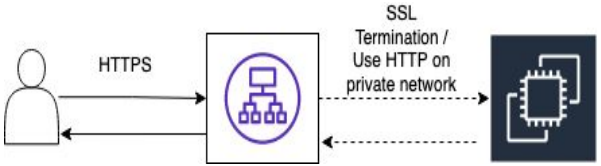
	Type	Description	Mnemonic
General Purpose	a1	Good for scale-out workloads, supported by Arm	**a** is for Arm processor – or as light as **A1** steak sauce
	t-family: t3, t3a, t2	Burstable, good for changing workloads	**t** is for **tiny** or **turbo**
	m-family: m6g, m5, m5a, m5n, m4	Balanced, good for consistent workloads	**m** is for **main** or happy **medium**
Compute Optimized	c-family: c5, c5n, c4	High ratio of compute to memory	**c** is for **compute**
Memory Optimized	r-family: r5, r5a, r5n, r4	Good for in-memory databases	**r** is for **RAM**
	x1-family: x1e, x1	Good for full in-memory applications	**x** is for **xtreme**
	High memory	Good for large in-memory databases	High memory is for... high memory.
	z1d	Both high compute and high memory	**z** is for **zippy**
Accelerated Computing	p-family: p3, p2	Good for graphics processing and other GPU uses	**p** is for **pictures**
	Inf1	Support machine learning inference applications	**Inf** is for **inference**
	g-family: g4, g3	Accelerate machine learning inference and graphics-intensive workloads	**g** is for **graphics**
	f1	Customizable hardware acceleration with field programmable gate arrays (FPGAs)	**f** is for **FPGA** or **feel** as in hardware
Storage Optimized	i-family: i3, i3en	SDD-backed, balance of compute and memory	**i** is for **IOPS**
	d2	Highest disk ratio	**d** is for **dense**
	h1	HDD-backed, balance of compute and memory	**H** is for **HDD**

EC2 Instance Types Comparison (and how to remember them) - ParkMyCloud

AWS ELB

- An ELB (EC2 Load Balancer) is a **managed load balancer**
- AWS takes care of upgrades, maintenance
- **Spreads load** across multiple downstream instances
- Exposes a s**ingle point of access** (DNS) to your application
- Does regular **health checks** to your instances
- **High availability** across zones
- **Separates** public traffic from private traffic
- Provide **SSL termination** (HTTPS) for your websites





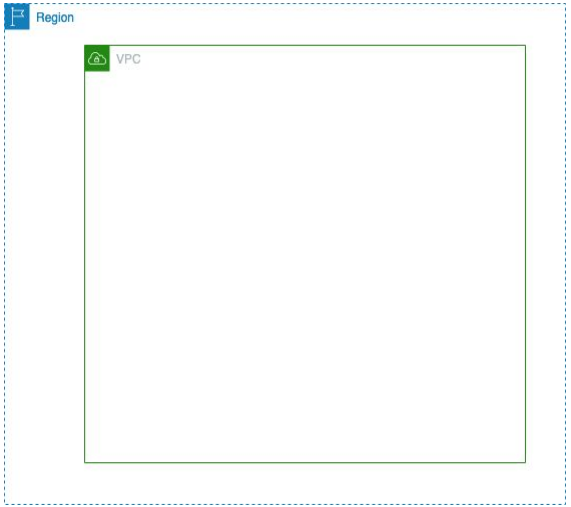
Types of load balancer on AWS

- **Classic Load Balancer** (v1 - old generation) – HTTP, HTTPS, TCP
- **Application Load Balancer** (v2 - new generation) – HTTP, HTTPS, WebSocket
- **Network Load Balancer** (v2 - new generation) – TCP, TLS & UDP

You can setup **internal** (private) or **external** (public) ELBs

AWS VPC

- VPC = **Virtual Private Cloud** to hold all of our AWS resources
- Restricts what sort of traffic, IP addresses and also the users that can access our instances
- VPC is **private**, only the Private IP ranges are allowed (10.0.0.0–10.255.255.255 / 172.16.0.0–172.31.255.255 / 192.168.0.0–192.168.255.255)
- Up to **5 per region** – soft limit
- A VPC's CIDR (Classless Inter-Domain Routing) should **not overlap** with your other networks

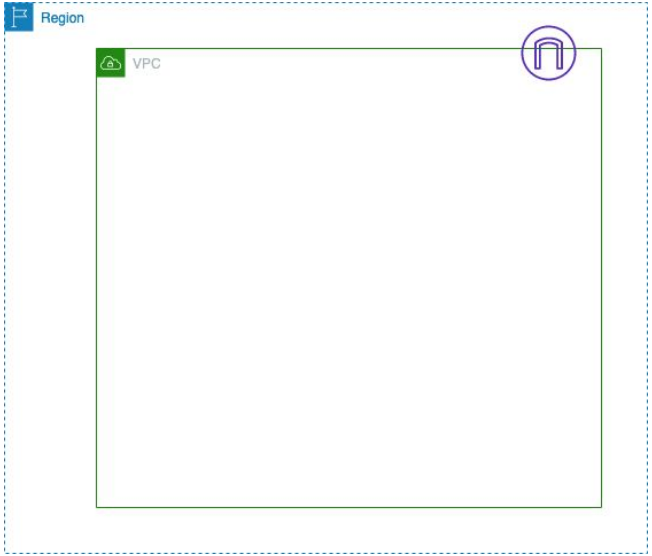


AWS VPC Components

- **Subnet:** A segment of a VPC's IP address range where you can place groups of isolated resources
- **Internet Gateway:** The Amazon VPC side of a connection to the public Internet
- **NAT Gateway:** Highly available, managed service for resources in a private subnet to access the Internet
- **Virtual private gateway:** The Amazon VPC side of a VPN connection
- **Peering Connection:** Route traffic via private IP addresses between two peered VPCs
- **VPC Endpoints:** Enables private connectivity to services hosted in AWS, from within your VPC
- **Egress-only Internet Gateway:** A stateful gateway to provide egress only access for IPv6 traffic from the VPC to the Internet

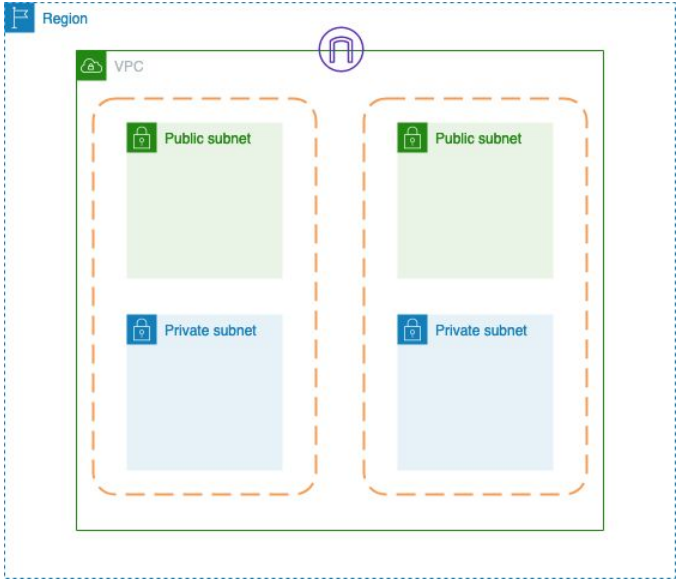
AWS Internet Gateways (IG)

- VPC are in a private network -> Can **not reach** internet
- IG **helps** our VPC instances connect with the **internet**
- Managed by AWS, scales **horizontally** and is **HA**
- One VPC can only be **attached** to **one** IGW and vice versa



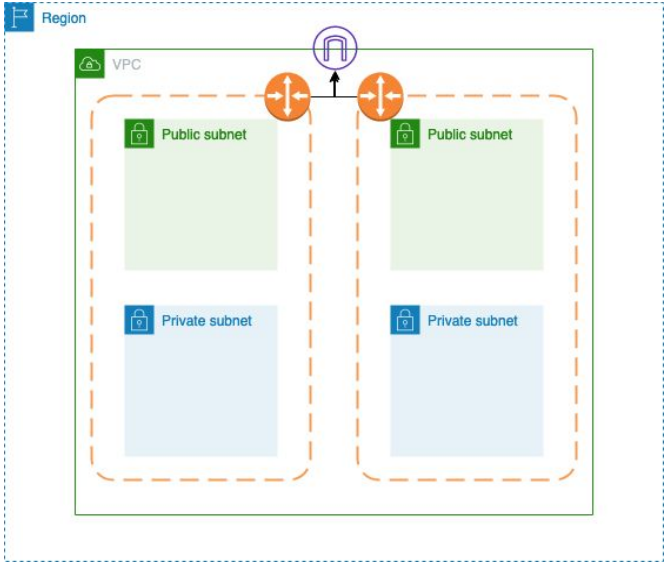
AWS Subnets

- Are containers within VPC that **segment** off a slice of the **CIDR block** you define in your VPC
- Subnets allow you to give **different access rules** and place resources in different containers where those rules should apply
- Is a Availability Zone resource
- Can be **public** (accessible from the internet) or **private** (**not** accessible from the internet)



AWS Route Tables

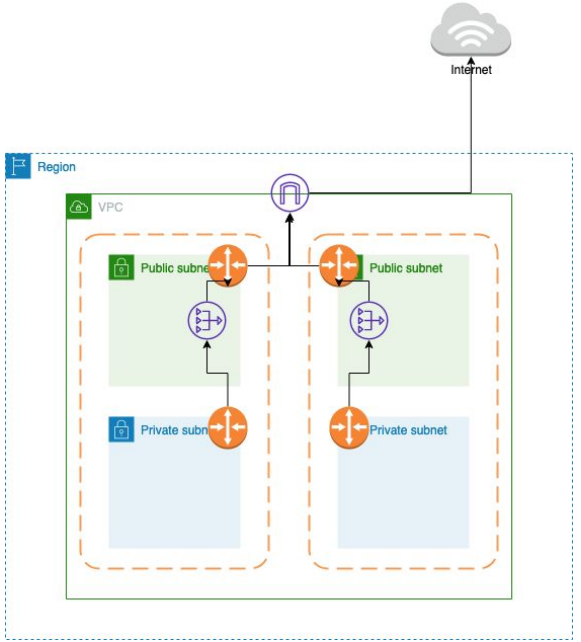
- Contains a **set of rules**, called routes, that are used to **determine** where network **traffic** from your subnet or gateway is directed
- Each **subnet** in your VPC must be **associated** with a route table, which controls the routing for the subnet (subnet route table)
- Each route in a table specifies a **destination** and a **target**
- For example, to enable a subnet to access the internet through an internet gateway, we can use the route table entry from the second image



Destination	Target
192.168.0.0/16	local
0.0.0.0/0	igw-087fc142cabecf6d8

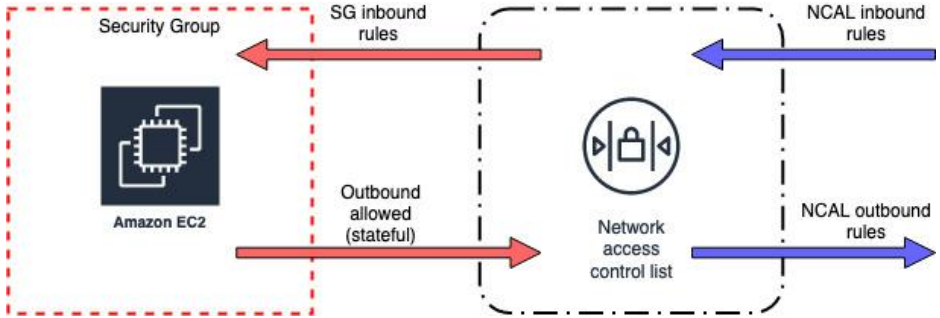
AWS NAT Gateway

- Allows instances in the **private** subnets to **connect** to the internet.
- Must be launched in a **public** subnet.
- **Managed** by AWS
- NAT is created in a **specific** AZ, uses an EIP
- 1 NAT per AZ to have **fault-tolerance** and **HA** (High Availability)
- Requires an IGW (**Private Subnet => NAT => IGW**)



Network ACLs

- NACL are like a **firewall** which control **traffic from and to subnet**
- Are placed on **subnet level**
- **Default** NACL allows everything outbound and everything inbound
- **One** NACL per Subnet
- **Deny** and **Allow** rules
- **Stateless**



AWS Security Groups

- They **control** how **traffic** is allowed into or out of our EC2 Instances.
- Security groups **only** contain **rules**
- Security groups rules can reference by **IP** or by **security group**
- **Stateful**: Changes in i**ncoming rules applied to outgoing rules**

Security group ID	Security group na...	VPC ID	Description
sg-01c6514886e02738b	alb-allow-http	vpc-03222a142e126c781	Allow HTTP access to ALB
sg-0a5f80739ed40184b	ec2-sg	vpc-03222a142e126c781	Allow access to ec2 only from ALB
sg-0d964290f4b09a593	default	vpc-03222a142e126c781	default VPC security group

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	Allow access http on port 80
HTTP	TCP	80	::/0	Allow access http on port 80

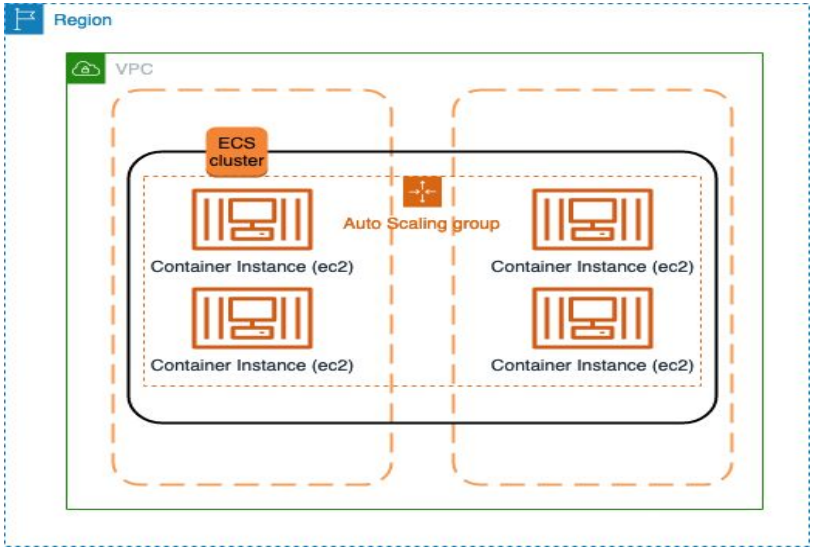
Type	Protocol	Port range	Source	Description - optional
All TCP	TCP	0 - 65535	sg-01c6514886e02738b / alb-allow-http	–

AWS NACLs vs SG

Security Group	NACL
Instance level	Subnet level
Stateful	Stateless
Allow rules only	Allow and Deny rules
All rules are evaluated before traffic is allowed	Rules are evaluated in the order specified
First layer of defense for egress traffic	First layer of defense for ingress traffic

AWS ECS

- ECS = Elastic Container Service
- Launch **Docker** containers on AWS
- Simplifies running containers in a HA manner across multiple Availability Zones within a Region
- Serverless with AWS Fargate



ECS Cluster

- Is Region specific
- Is a logical grouping of tasks and services
- Uses one or more EC2 Instances to run tasks
- EC2 instances of the cluster run the ECS agent
- The ECS agent registers the instance to the Cluster
- Serverless using AWS Fargate

ECS Task Definition

- A **JSON** file that **describes** one or more containers for ECS to run
- Can be thought of as a **blueprint for your application**
- **Docker image** to use with each container in your task
- **CPU** and **memory** to use with each task
- Which **ports** should be opened for your application
- What **data volumes** should be used with the containers in the task

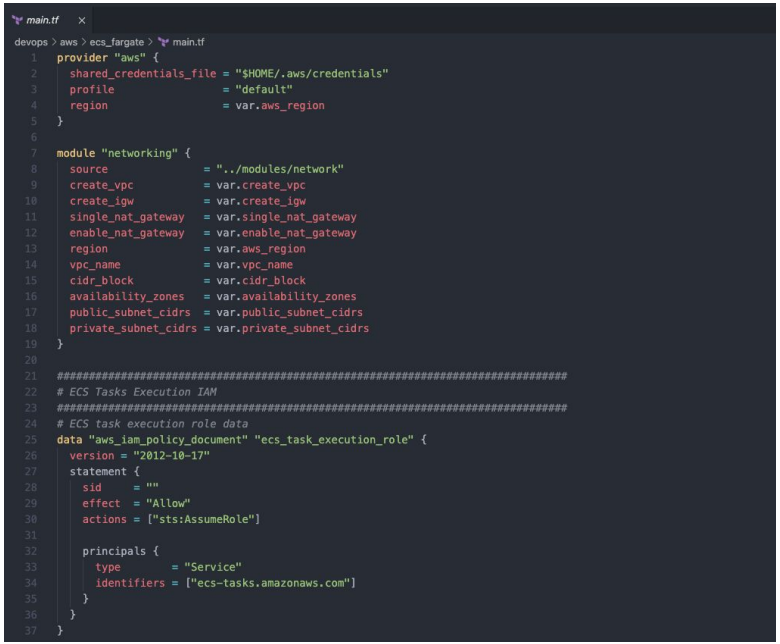
ECS Services

- Allows to **run and maintain** a specified number of tasks
- If any of the tasks **fails**, ECS launches another task in order to maintain the desired number of tasks in the service
- **Task placement strategies** and constraints to customise task placement decisions
- Three **deployment types**: rolling update, blue/green, and external
- Can be linked to an **ELB** (Load Balancer)

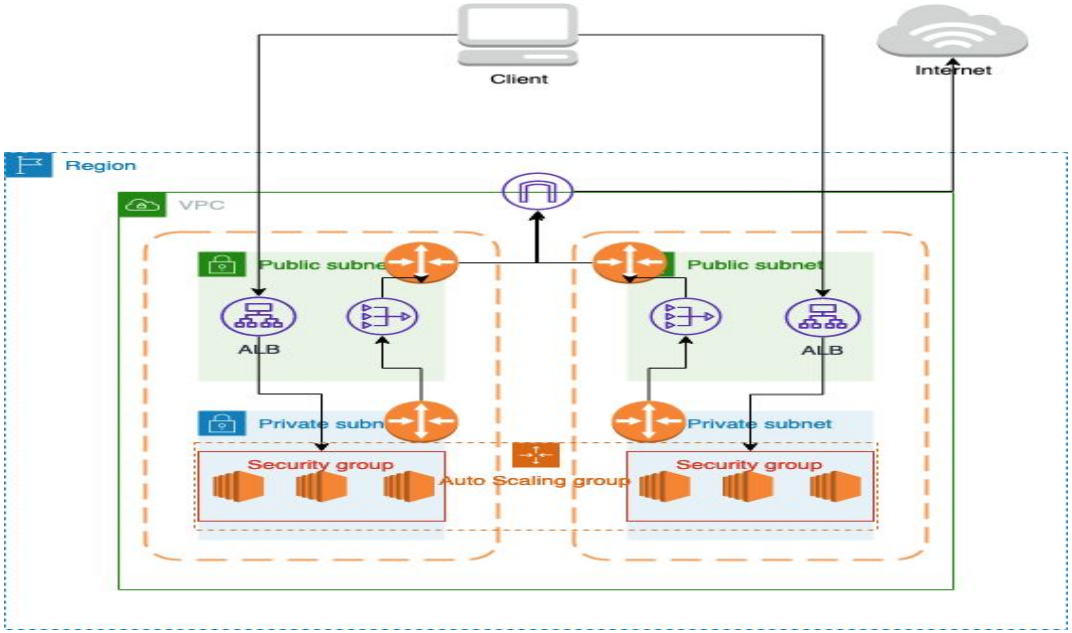
Terraform

- **Infrastructure as Code** (described using a high-level configuration syntax)
- Is a tool for **building**, **changing**, and **versioning** infrastructure safely and efficiently
- Configuration files describe to Terraform the components needed to run
- Generates an **execution plan** describing what it will do to reach the desired state
- Executes the plan to build the described infrastructure
- Determines what changed and creates **incremental** execution plans
- Can manage low-level components (compute instances, networking), as well as high-level components (DNS entries, SaaS features)

Terraform hands-on



AWS Monolith API design with fault-tolerance and HA




Thank you
Alexandros and Dimosthenis
arconsis