Tworzymy sieć pod aplikację “Generator Memów”

W tym zadaniu utworzysz niezbędne zasoby sieciowe pod aplikację.

Przygotowanie struktury projektu

1. Utwórz katalog (`katalog główny`), w którym będziesz trzymać wszystkie pliki używane na tym kursie (np. `aws-w-praktyce`). Jeśli masz już taki katalog, możesz pominąć ten krok.

2. Struktura katalogów będzie wyglądać tak:

```
\${Root-directory}
    \${Project}
        \${Component}
            \commands
            \parameters
            \templates
        \${Component}
            \commands
            \parameters
            \templates
        (...)
```

Dla aplikacji “Generator Memów” - na tym etapie projektu - będziesz mieć następującą strukturę:

```
\${Root-directory}
    \memes-generator
        \network
            \commands
            \parameters
            \templates
        \operations
            \commands
            \parameters
            \templates
```

W kolejnych etapach pojawią się kolejne komponenty, szablony i inne pliki, więc struktura projektu będzie wygladać mniej więcej tak:

```
\${Root-directory}
    \memes-generator
        \application
            \commands
            \parameters
            \templates
        \data
            \commands
```

```
        \parameters
        \scripts
        \templates
    \network
        \commands
        \parameters
        \templates
    \operations
        \commands
        \parameters
        \templates
    (...)
```

Co znajduje się w folderach

`commands` - przykładowe komendy, których możesz użyć do tworzenia stacków `parameters` - pliki `json` z parametrami dla danego stacka (z reguły dla danego typu środowiska - `dev`, `test`, `prod` - będzie odrębny plik z konfiguracją, a nazwa środowiska jest w nazwie pliku) `templates` - szablony do tworzenia infrastruktury

W późniejszych etapach projektu mogą pojawić się dodatkowe foldery (np. z kodem funkcji).

Jak używać plików z przykładowymi poleceniami

1. W terminalu wybierz katalog główny projektu.
2. Na samym początku ustaw zmienne `PROJECT`, `STAGE` i `REGION`.
3. Kolejne zmienne będziesz ustawiać już dla konkrektych stacków: `COMPONENT`, `STACK`, `TEMPLATE` i `PARAMETERS`.
4. Na podstawie tych zmiennych tworzone są kolejne, czyli `TEMPLATE_FILE` i `PARAM_FILE`
5. Odczytaj z pliku z parametrami wartości parametrów i zapisz je do zmiennej `PARAMS`
6. Utwórz, wyświetl, a następnie wykonaj polecenie `$deploy`.

Jeśli tworzysz kilka stacków, czynności z pkt. 3, 4 i 5 trzeba powtórzyć dla każdego stacka.

Stack 1 - bucket na logi

Na samym początku utworzymy bucket, do którego będą trafiać logi. Pliki do tej części znajdują się w folderze `operations`.

Pliki

Szablon: `memes-generator/operations/templates/log-bucket.yaml` Plik z parametrami dla środowiska `dev`: `memes-generator/operations/parameters/log-bucket-dev.json` Plik z przykładowymi poleceniami: `memes-generator/operations/commands/deploy-log-bucket.sh`

Tworzone zasoby

Utworzymy 3 zasoby:

1. Bucket na logi
2. Parametr SSM z nazwą bucketu na logi
3. Parametr SSM z ARN bucketu na logi

Utworzone paramtery SSM będą następnie użyte jako parametry wejściowe do kolejnych stacków.

Uwagi dotyczące zasobów

Access Control

Zwróć uwagę na property `AccessControl`:

```
  S3LogBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      AccessControl:  LogDeliveryWrite
      Tags:
        - Key: Name
          Value: !Sub ${Project}-${Stage}-${Component}-log-bucket
```

Ma ona wartość `LogDeliveryWrite` - co jest niezbędne, gdy chcemy pozwolić na zapisywanie logów z VPC lub z usługi Elastic Load Balancing.

Nazwa bucketu

Nazwa bucketu w szablonie nie jest podawana jawnie - usługa CloudFormation wygeneruje nazwę automatycznie.

Stack 2 - VPC

Pliki

Pliki do tej części znajdują się w folderze `network`.

Szablon: `memes-generator/network/templates/network.yaml` Plik z parametrami dla środowiska `dev`: `memes-generator/network/parameters/network-dev.json` Plik z przykładowymi poleceniami: `memes-generator/network/commands/deploy-network.sh`

Parametry wejściowe

Dla sieci najważniejszymi parametrami będą zakresy IP dla VPC oraz dla podsieci. Zakresy dla podsieci nie mogą się pokrywać.

Zwróć uwagę na `AllowedPattern` - jeśli parametr nie będzie miał odpowiedniego formatu, wdrożenie stacka nie powiedzie się. Używanie `AllowedPattern` pozwala usłudze CloudFormation dokonać walidacji parametrów jeszcze zanim zacznie się tworzenie zasobów. Stosowanie `AllowedPattern` i innych mechanizmów kontroli parametrów jest dobrą praktyką.

Tworzone zasoby

1. VPC - wirtualna sieć (`AWS::EC2::VPC`)
2. Flow log - log rejestrujący ruch w ramach VPC (`AWS::EC2::FlowLog`)
3. Subnet (Podsieć) - tworzymy ich aż 6: 2 publiczne, 2 prywatne i 2 na zasoby przechowujące dane (`AWS::EC2::Subnet`)
4. Internet Gateway - brama, przez którą ruch wchodzi i wychodzi do Internetu (`AWS::EC2::InternetGateway`)
5. Gateway Attachment - połączenie utworzonej Internet Gateway z konkretną siecią VPC (`AWS::EC2::VPCGatewayAttachment`)
6. Tablice routingu: 1 dla podsieci publicznych oraz po 1 dla każdej podsieci niepublicznej (w sumie 4 - `AWS::EC2::RouteTable`)
7. Route - czyli wpis do tablicy routingu `AWS::EC2::Route`
8. Route Table Association - czyli przypisanie danej tablicy routingu do konkretnej podsieci (`AWS::EC2::SubnetRouteTableAssociation`)
9. Parametry SSM zawierające ID, nazwy lub ARN zasobów. Parametry te bedą będą następnie użyte jako parametry wejściowe do kolejnych stacków.

Uwagi do zasobów

DependsOn

Zwróć uwagę, że dla zasobu `DefaultPublicRoute` dodane jest `DependsOn`:

```
DefaultPublicRoute:
  Type: AWS::EC2::Route
  DependsOn:
    - AttachGateway
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway
```

Jest tak dlatego, że utworzenie tego wpisu do tablicy wymaga, aby wcześniej została utworzona `Internet Gateway` i przypisana do naszej `VPC`. Jednocześnie między zasobem `DefaultPublicRoute` a `AttachGateway` nie ma jawnego odniesienia. Bez `DependsOn` usługa CloudFormation mogłaby rozpocząć tworzenie wpisów zanim zostałyby spełnione wszystkie warunki do powstania wpisu - spowodowałoby to błąd i wycofanie tworzenia stacka lub zmian w stacku.

MapPublicIpOnLaunch

`MapPublicOnLaunch` określa, czy zasoby tworzone w danej podsieci będą miały automatycznie przypisany publiczny adres IP. Porównaj wartości tej property dla 2 podsieci - publicznej oraz niepublicznej:

```
  PublicSubnetA:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone: !Select [ 0 , !GetAZs '' ]
      VpcId: !Ref Vpc
      CidrBlock: !Ref PublicSubnetACidr
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${Project}-${Stage}-${Component}-public-subnet-a

  DataSubnetA:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone: !Select [ 0 , !GetAZs '' ]
      VpcId: !Ref Vpc
      CidrBlock: !Ref DataSubnetACidr
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${Project}-${Stage}-${Component}-data-subnet-a
```

Funkcja !Select

```
  DataSubnetA:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone: !Select [ 0 , !GetAZs '' ]
      VpcId: !Ref Vpc
      CidrBlock: !Ref DataSubnetACidr
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${Project}-${Stage}-${Component}-data-subnet-a
```

Za pomocą tej funkcji możemy automatycznie wybrać `Availability Zone` dla danej podsieci z listy dostępnych AZ dla danego regionu. Cyfry `0`, `1` itd. są kolejnymi indeksami na liście dostępnych AZ (zaczynając od 0).

Stack 3 - Security Groups

Pliki

Pliki do tej części znajdują się w folderze `network`.

Szablon: `memes-generator/network/templates/security-groups.yaml` Plik z parametrami dla środowiska dev: `memes-generator/network/parameters/security-groups-dev.json` Plik z przykładowymi poleceniami: `memes-generator/network/commands/deploy-security-groups.sh`

Tworzone zasoby

1. Security Group:
   - `JumpHostSecurityGroup` - grupa, w której znajdzie się jumphost
   - `AlbSecurityGroup` - grupa, w której znajdzie się load balancer
   - `ApplicationSecurityGroup` - grupa dla instancji EC2 z backendem aplikacji
   - `DatabaseSecurityGroup` - grupa dla bazy danych
2. Parametry SSM

Uwagi do parametrów i zasobów

AWS::SSM::Parameter::Value

Zwróć uwagę na typ parametru `VpcId`:

```
VpcId:
    Description: Reference of the VpcId from the SSM
    Type: AWS::SSM::Parameter::Value<String>
```

Typ parametru wskazuje, że wartość tego parametru będzie pobrana z SSM Parameter Store (i będzie ona typu `String`)

Parametry ApplicationPort i DatabasePort

Reguły w SecurityGroups wymagają podawania numerów portów. Aplikacje i bazy danych działają na różnych portach. Jeśli podajemy nr portu w parametrze wejściowym (zamiast bezpośrednio przy `property` zasobu), możemy używać tego samego szablonu i plików z różnymi zestawami parametrów do tworzenia stacków dla wielu różnych typów aplikacji i baz danych.

```
  DatabasePort:
    Description: Port for Database
    Type: String

  DatabaseSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: SecurityGroup for Database
      GroupName: !Sub ${Project}-${Stage}-${Component}-database-sg
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: !Ref DatabasePort
        ToPort: !Ref DatabasePort
        SourceSecurityGroupId: !Ref ApplicationSecurityGroup
      - IpProtocol: tcp
```

```
      FromPort: !Ref DatabasePort
      ToPort: !Ref DatabasePort
      SourceSecurityGroupId: !Ref JumpHostSecurityGroup
    VpcId: !Ref VpcId
    Tags:
      - Key: Name
        Value: !Sub ${Project}-${Stage}-${Component}-database-sg
```

Z kolei dla grupy `JumpHostSecurityGroup` znamy z góry zestaw portów do połączeń SSH, więc tam numer portu pojawia się obok property:

```
JumpHostSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: SecurityGroup for JumpHost
    GroupName: !Sub ${Project}-${Stage}-${Component}-jumphost-sg
    SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: 22
      ToPort: 22
      CidrIp: !Ref JumpHostAllowedCidrIpBlock
    VpcId: !Ref VpcId
    Tags:
      - Key: Name
        Value: !Sub ${Project}-${Stage}-${Component}-jumphost-sg
```

Bloki CIDR vs ID innej Security Group w regułach

Do zasobów z grupy `AlbSecurityGroup` (czyli do load balancera) będziemy chcieli dopuścić ruch spoza AWS, więc przy definiowaniu zródła ruchu wchodzącego użyta została property `CidrIp` i konkretny blok CIDR:

```
AlbSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: SecurityGroup for Alb
    GroupName: !Sub ${Project}-${Stage}-${Component}-alb-sg
    SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: 80
      ToPort: 80
      CidrIp: 0.0.0.0/0
    - IpProtocol: tcp
      FromPort: 443
      ToPort: 443
      CidrIp: 0.0.0.0/0
    VpcId: !Ref VpcId
    Tags:
      - Key: Name
        Value: !Sub ${Project}-${Stage}-${Component}-alb-sg
```

Natomiast do zasobów w `ApplicationSecurityGroup` dopuszczamy ruch z zasobów w `AlbSecurityGroup` i `JumpHostSecurityGroup` - podajemy więc ID tych grup (za pomocą funkcji `!Ref`):

```
  ApplicationSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: SecurityGroup for Application
      GroupName: !Sub ${Project}-${Stage}-${Component}-application-sg
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: !Ref ApplicationPort
        ToPort: !Ref ApplicationPort
        SourceSecurityGroupId: !Ref AlbSecurityGroup
      - IpProtocol: tcp
        FromPort: !Ref ApplicationPort
        ToPort: !Ref ApplicationPort
        SourceSecurityGroupId: !Ref JumpHostSecurityGroup
      VpcId: !Ref VpcId
      Tags:
        - Key: Name
          Value: !Sub ${Project}-${Stage}-${Component}-application-sg
```

Stack 4 - NAT Gateway

Pliki

Pliki do tej części znajdują się w folderze `network`.

Szablon: `memes-generator/network/templates/nat-gateway.yaml` Plik z parametrami dla środowiska `dev`:

`memes-generator/network/parameters/nat-gateway-a-dev.json` `memes-generator/network/parameters/nat-gateway-b-dev.json`

Plik z przykładowymi poleceniami: `memes-generator/network/commands/deploy-nat-gateway.sh`

Uwaga: potrzebujemy 2 pliki z parametrami, gdyż - docelowo - w projekcie będą utworzone 2 NAT Gateway w 2 odrębnych Availability Zone.

Parametry wejściowe

1. `PublicSubnet` - określa, w której podsieci publicznej znajdzie się NAT Gateway
2. `PrivateRouteTable` - określa, w której tablicy routingu (przypisanej do prywatnej podsieci) będzie umieszczony wpis pozwalający na ruch wychodzący do Internetu.

Tworzone zasoby

1. NAT Gateway
2. Stały adres IP (wymagany przez NAT Gateway)

3. Wpis do tablicy routingu umożliwiający ruch z podsieci prywatnej przez NAT Gateway do Internetu (po drodze jest jeszcze Internet Gateway)

Uwagi do zasobów i parametrów

Zwróć uwagę, jak przekazujemy w parametrach wejściowych ID odpowiedniej podsieci i tablicy routingu dla danej NAT Gateway:

Fragment pliku z parametrami:

```
    {
        "ParameterKey": "PublicSubnet",
        "ParameterValue": "/memes-generator/dev/network/public-subnet-
a/subnet-id"
    },
    {
        "ParameterKey": "PrivateRouteTable",
        "ParameterValue": "/memes-generator/dev/network/private-route-table-
a/rt-id"
    }
```

Są to nazwy parametrów SSM utworzonych w ramach stacka tworzącego sieć, podsieci i tablice routingu.

Fragment pliku z szablonem defniującym parametry SSM:

```
  PublicSubnetAIdParam:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
      Description: !Sub Stores ${Project}-${Stage}-${Component}-
PublicSubnetA Id
      Tier: Standard
      Name: !Sub /${Project}/${Stage}/${Component}/public-subnet-a/subnet-id
      Value: !Ref PublicSubnetA
      Tags:
        Name: !Sub ${Project}-${Stage}-${Component}-public-subnet-a-id

  PrivateRouteTableA:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref Vpc
      Tags:
        - Key: Name
          Value: !Sub ${Project}-${Stage}-${Component}-private-route-table-a
```

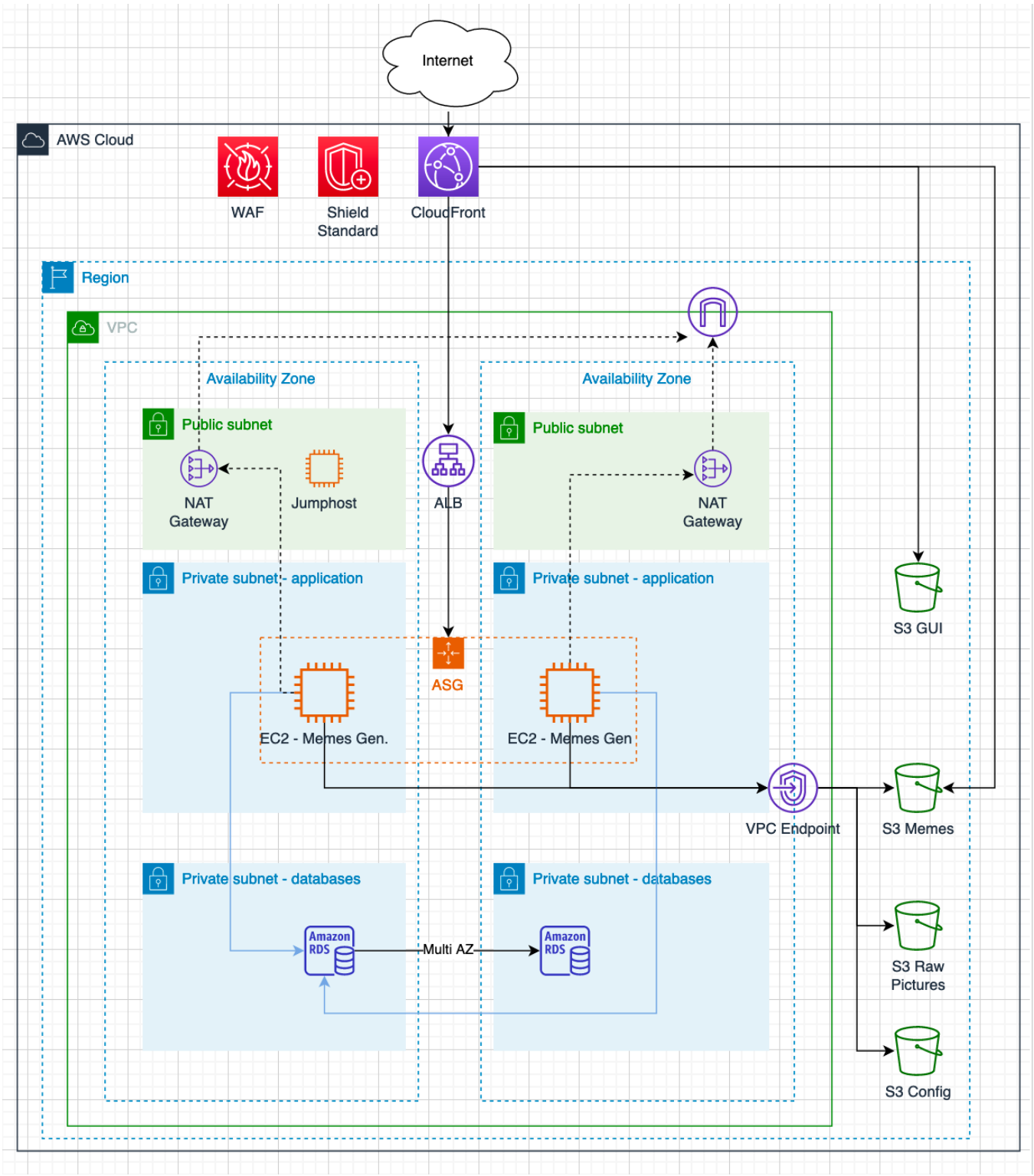
`${Project}, ${Stage} i ${Component}` zostały w pliku z parametrami zastąpione wartościami podanymi w parametrach wejściowych stacka `network`.

Po przetestowaniu tworzenia stacków z NAT Gateway usuń je - do czasu, aż będą potrzebne. Unikniesz niepotrzebnych kosztów, które w przypadku NAT Gateway są dość wysokie

Co po utworzeniu stacków?

1. Obejrzyj w konsoli AWS utworzone zasoby - wpisz w wyszukiwarkę `VPC`. Obejrzyj, co jest w sekcjach `Your VPCs`, `Subnets`, `Route Tables`, `NAT Gateways`
2. Zagadka: dlaczego wpis do tablicy routingu dotyczący NAT Gateway został utworzony dopiero w ostatnim stacku? Co by się stało, gdyby utworzyć ostatni stack bez tego wpisu, a następnie zaktualizować stack 3 o ten wpis?

Diagram


Internet
AWS Cloud
WAF
Shield Standard
CloudFront
Region
VPC
Availability Zone
Availability Zone
Public subnet
Public subnet
NAT Gateway
Jumphost
ALB
NAT Gateway
Private subnet - application
Private subnet - application
S3 GUI
ASG
EC2 - Memes Gen.
EC2 - Memes Gen
VPC Endpoint
S3 Memes
Private subnet - databases
Private subnet - databases
Amazon RDS
Multi AZ
Amazon RDS
S3 Raw Pictures
S3 Config