AWS-pilviarkkitehtuurin suunnittelu ja toteutus

Marianne Äikäs, AB6258, HTK22S1

Lopputyö
ICT-infrastruktuuri pilvialustalla HT00BN69-3003, Juha-Tapio Teno
5.5.2024
Tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
University of Applied Sciences

Sisältö

1 Johdanto 3

2 Osa 1: Pilviarkkitehtuurin suunnittelu 3

2.1 Vaihe 1: Arkkitehtuurikaavio 3

2.2 Vaihe 2: CloudFormation-templaatit ympäristön luomiseksi 4

2.2.1 VPC (Virtual Private Network) 4

2.2.2 Internet Gateway ja Gateway Attachment 5

2.2.3 Subnets (Aliverkot) 5

2.2.4 Routing tables (Reititystaulut) 6

2.2.5 Security Group (InstanceSecurityGroup) 7

2.2.6 EFS (Elastic File System) 8

2.2.7 Launch Template 9

2.2.8 AutoScaling ja CloudWatch Alarm 12

2.2.9 Application Load Balancer 13

2.2.10 Outputs 14

3 Osa 2: Pilviarkkitehtuurin toteutus ja reflektio 14

3.1 Toteutetun pilviarkkitehtuurin monimutkaisuus 14

3.2 CloudFormation-templaattien käyttö ympäristön luomiseen 15

3.3 Arkkitehtuurikaavio 15

3.4 Pohdinta käytetyistä palveluista 16

Kuviot

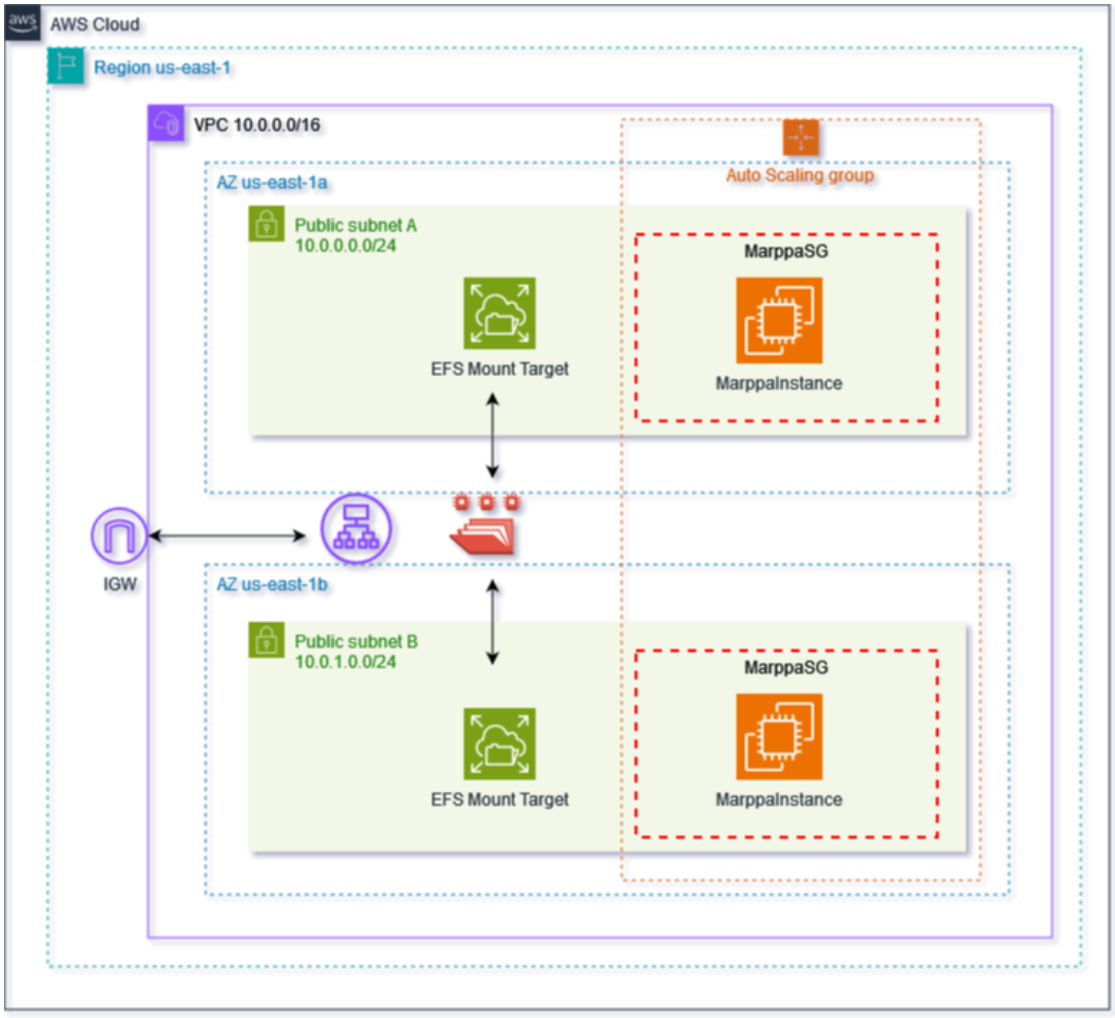
Kuvio 1. Arkkitehtuurikaavio toteutettu draw.io -sovelluksella. 3

1 Johdanto

Tässä loppuprojektissa minun tehtäväni on suunnitella ja toteuttaa pilviarkkitehtuuri Amazon Web Servicesin (AWS) avulla. Projekti jakautuu kahteen pääosaan: suunnitteluun ja toteutukseen. Pilviarkkitehdin työssä on oleellista muodostaa kokonaiskäsitys rakennettavasta pilviympäristöstä, ja pohtia mitä palveluita kannattaa hyödyntää ympäristön rakentamisessa. Loppuprojektissa työ toteutetaan AWS:n Academy Learner Lab -ympäristössä. Olen rakentanut koko lopputukoksen IaC-menetelmällä (Infrastucture as a Code), jotta mukailisin parhaiden periaatteiden nopeutta ja edullisuutta.

2 Osa 1: Pilviarkkitehtuurin suunnittelu

2.1 Vaihe 1: Arkkitehtuurikaavio



Kuvio 1. Arkkitehtuurikaavio toteutettu draw.io -sovelluksella.

2.2 Vaihe 2: CloudFormation-templaatit ympäristön luomiseksi

Tätä kyseistä tehtävää varten halusin luoda yhden kokonaisen CloudFormation-templaten, jotta infrastruktuurin pystyttäminen olisi mahdollisimman suoraviivaista ja helppoa opettajalle, joka tarkistaa tämän tehtävän. Normaalissa tilanteessa olisin rakentanut komponentin eri osiin:

1. Ensimmäisenä verkkokonfiguraatio
2. EC2-komponentit ja Security Groupit
3. Loadbalanceri sekä Auto Scaling Group asetukset

Kun teemme erillisinä ”layereinä” infraskutuurin koodin, se kestää paremmin päivityksiä ja bugeja, kun päivitettävät osat eivät kuulu tiettyjen osien devaamiseen. Tämä käytäntö on myös tietoturvallisempaa ja poistaa tarpeen ”kovakoodata” tietoja templateihin. Layereiden hyödyntäminen mahdollistaa CI/CD julkaisuputken hyödyntämisen. YAML-tiedostoihin tulisi tehdä Output-osio, jossa käytetään Export-toimintoa, jolloin näiden tietojen hyödyntäminen onnistuu eri stackien välillä.

2.2.1 VPC (Virtual Private Network)

```
VPC:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: 10.0.0.0/16
    EnableDnsSupport: yes
    EnableDnsHostnames: yes
    Tags:
      - Key: Name
        Value: MarppaVPC
```

Kaikki komponentit tarvitsevat verkon, johon ne rakentuvat. Tässä tulee käytäntöön VPC. **VPC** on virtuaalinen verkko, jonka avulla käyttäjät voivat eristää ja hallita resurssejaan pilvessä. Se mahdollistaa mukautettujen verkkoympäristöjen luomisen, kuten IP-osoitteiden valinnan, aliverkkojen määrittelyn ja verkkoliikenteen hallinnan. VPC tarjoaa turvallisen ja eristetyn ympäristön AWS-palveluille ja resursseille.

2.2.2 Internet Gateway ja Gateway Attachment

```
#Creating Internet Gateway
  InternetGateway:
    Type: 'AWS::EC2::InternetGateway'
    Properties:
      Tags:
        - Key: Name
          Value: MarppaIGW
#Attaching the ITG to the VPC
  InternetGatewayAttachment:
    Type: 'AWS::EC2::VPCGatewayAttachment'
    Properties:
      VpcId: !Ref VPC
      InternetGatewayId: !Ref InternetGateway
```

Internet Gateway mahdollistaa yhteyden muodostamisen VPC:n ja julkisen Internetin välillä. IGW mahdollistaa resurssien, kuten EC2-instanssien, pääsyn Internetiin ja Internetistä tulevan liikenteen reitittämisen VPC:hen.

Gateway Attachment on käytännössä linkki VPC:n ja Internet Gatewayn välillä ja se varmistaa, että VPC:lle osoitetut Internet-liikenteen pyynnöt kulkevat oikein ja vastaavasti vastaukset palautuvat takaisin VPC:hen.

2.2.3 Subnets (Aliverkot)

```
#Creating Public Subnet 1
  PublicSubnet1:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: us-east-1a
      CidrBlock: 10.0.0.0/24
      MapPublicIpOnLaunch: yes
      Tags:
        - Key: Name
          Value: MarppaPublicSubnet1AZ1
#Creating Public Subnet 2
  PublicSubnet2:
    Type: 'AWS::EC2::Subnet'
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: us-east-1b
      CidrBlock: 10.0.1.0/24
```

```
      MapPublicIpOnLaunch: yes
      Tags:
        - Key: Name
          Value: MarppaPublicSubnet2AZ2
```

Tässä tehtävässä loin VPC-alueelleni kaksi julkista aliverkkoa. Kummatkin aliverkot ovat erillisillä Availability Zoneilla. (Availability Zone voidaan käsittää olevan ”fyysinen konesali”, jossa palvelimet sijaitsevat, eli fyysinen alue.) Olen tehnyt kaksi julkista aliverkkoa siksi koska sijoitan EC2-instanssit näille, joten niiden tulee olla julkisia. Tehtävässä minulla ei ollut tarvetta yksityiselle aliverkolle, koska en käsittele esimerkiksi tietokantapalvelimia infrastruktuurirakenteessani. Verkot ovat erillisillä AZ-alueilla, jotta rakennelmani olisi vikasietoisempi.

Subnet (aliverkko) on alue, joka käsittää osan VPC:n IP-osoiteavaruudesta ja sijoitetaan tiettyyn fyysiseen alueeseen (AZ) tai vyöhykkeeseen (Region). Aliverkot voivat olla joko julkisia tai yksityisiä, riippuen niiden asetuksista ja liitettyjen palvelujen saatavuudesta Internetiin.

2.2.4 Routing tables (Reititystaulut)

```
#Creating a Public Route Table
  PublicRouteTable:
    Type: 'AWS::EC2::RouteTable'
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: MarppaPublicSubnetRouteTable
#Configuring Public Route
  PublicRoute:
    Type: 'AWS::EC2::Route'
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway
#Associating Subnet 1 and Route Table
  PublicSubnet1RouteTableAssociation:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1
#Associating Subnet 2 and Route Table
  PublicSubnet2RouteTableAssociation:
    Type: 'AWS::EC2::SubnetRouteTableAssociation'
    Properties:
```

```
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2
```

Aikaisemmin luodut aliverkot tarvitsevat reititystauluja, jotta resurssien välinen ja yhteys Internetiin toimii. Reititystauluissa määritellään miten verkkoliikenne ohjataan: Ne ovat ohjeita verkkolaitteille, kuten reitittimille tai kytkimille. Reititystauluille tulee kertoa, mitä aliverkkoja ne ohjaavat ja tämä tapahtuu Association-osassa.

2.2.5 Security Group (InstanceSecurityGroup)

```
#Creating Security Group
  InstanceSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupName: MarppaSG
      GroupDescription: Enable SSH access and HTTP
      VpcId: !Ref VPC
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
```

Security Groupit ovat virtuaalisia palomuureja, jotka toimivat verkkotason suodattimina mm. EC2-instansseissa ja VPC-ympäristöissä. Niiden tarkoitus on valvoa liikennettä ryhmittämällä yhteen sallitut ja estettävät verkkoyhteydet instansseille niiden IP-osoitteiden ja porttien perusteella. Security Groupit voivat olla joko sisääntulon (Ingress) tai ulostulon (Egress) suuntaisia ja mahdollistavat tarkkaan määritellyn pääsyn instansseille.

Olen luonut Security Groupin Instansseille, jossa sallitaan kaikki Internetistä (0.0.0.0/0) sisään tuleva liikenne porteista 22 ja 80. Portti 22 on oletusportti SSH-yhteydelle ja portti 80 on oletus http-yhteydelle, johon yleisesti asennetaan webpalvelin.

2.2.6 EFS (Elastic File System)

```
###### Regional EFS configuration
  FileSystem:
    Type: AWS::EFS::FileSystem
    Properties:
      Encrypted: true
      FileSystemTags:
        - Key: Name
          Value: MarppaEFS
      PerformanceMode: generalPurpose
      ThroughputMode: bursting
  MountTarget1:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId: !Ref FileSystem
      SubnetId: !Ref PublicSubnet1
      SecurityGroups:
        - !Ref MountTargetSecurityGroup
  MountTarget2:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId: !Ref FileSystem
      SubnetId: !Ref PublicSubnet2
      SecurityGroups:
        - !Ref MountTargetSecurityGroup
  MountTargetSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allow NFS access to EFS from within the VPC
      VpcId: !Ref VPC
      GroupName: Marppa-EFS-SG
      SecurityGroupIngress:
        - IpProtocol: "tcp"
          FromPort: 2049
          ToPort: 2049
          CidrIp: 10.0.0.0/16
      SecurityGroupEgress:
        - IpProtocol: -1
          FromPort: 0
          ToPort: 0
          CidrIp: 0.0.0.0/0
```

EFS on pilvitiedostojärjestelmä, joka tarjoaa skaalautuvaa ratkaisua tallennukseen. Verkkopohjainen tiedostojärjestelmä mahdollistaa useiden EC2-instanssien pääsyn yhteiseen tiedostojärjestelmään

samanaikaisesti. EFS automatisoi tallennuksen kapasiteetin hallinnan ja skaalautuvuuden, mikä tekee siitä ihanteellisen ratkaisun dynaamisiin ja jaettuihin sovelluksiin, kuten web-sovelluksiin tai tiedostojen jakamiseen.

Regionaalisella tiedostojärjestelmällä tarkoitetaan sitä, että se on saavutettavissa alueen (region) sisällä eri aliverkoista. Tätä varten tarvitaan Mount Targeteja, liitospisteitä, joiden avulla EC2-instanssit osaavat ottaa yhteyttä jaettuun EFS:iin. Kun EC2-instanssi on liitetty mount targetiin, se voi käsitellä tiedostoja EFS:stä kuten paikallisia tiedostoja. Olen luonut kumpaakin aliverkkoani varten mount targetin, sillä kummassakin aliverkossani on EC2-instansseja.

Lopuksi olen myös luonut erillisen Security Groupin (MountTargetSecurityGroup), jonka tarkoituksena on mahdollistaa EFS:n toiminta VPC-verkossa. Se sallii sisään tulevaa liikennettä VPC-verkosta porttiin 2049, joka on oletusportti NFS:lle. Ulos suuntautuva liikenne on sallittu kaikkialle Internetiin (0.0.0.0/0).

2.2.7 Launch Template

```
#Configuring launch template
  LaunchTemplate:
    Type: 'AWS::EC2::LaunchTemplate'
    Properties:
      LaunchTemplateName: MarppaTemplate
      LaunchTemplateData:
        NetworkInterfaces:
          - DeviceIndex: 0
            AssociatePublicIpAddress: true
            Groups:
              - !Ref InstanceSecurityGroup
        ImageId: !Ref AMI
        InstanceType: !Ref InstanceType
        KeyName: vockey
        TagSpecifications:
          - ResourceType: "instance"
            Tags:
            - Key: "Name"
              Value: "MarppaInstance"
        UserData:
          Fn::Base64:
            Fn::Join:
              - ""
              - - "#!/bin/bash -x\n"
```

```
                - "export LC_CTYPE=en_US.UTF-8\n"
                - "export LC_ALL=en_US.UTF-8\n"
                - "yum update -y\n"
                - "amazon-linux-extras install epel -y\n"
                - "yum install -y httpd stress curl nfs-utils\n"
                - "EC2_REGION="  # Set up a variable to store the AWS region
               - Ref: "AWS::Region"  # Reference to the AWS region where the instance
is launched
                - "\n"
                  - "DIR_TGT=/var/www/html//\n"  # Define the target directory for
mounting EFS
                 - "EFS_FILE_SYSTEM_ID="  # Set up a variable to store the EFS file
system ID
                - Ref: "FileSystem"  # Reference to the AWS EFS file system ID
                - "\n"
                - "mkdir -p $DIR_TGT\n"  # Create the target directory if it doesn't
exist
                  - "DIR_SRC=$EFS_FILE_SYSTEM_ID.efs.$EC2_REGION.amazonaws.com\n"  #
Define the source directory for EFS
                                            -    "mount    -t    nfs4    -o    nfs-
vers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2            $DIR_SRC:/
$DIR_TGT\n"  # Mount the EFS file system to the target directory with NFS version
4.1 options


                  #Placing files for the installed httpd (Apache) Webserver
                - "cd /var/www/html/\n"
                        -  "wget  https://github.com/marppaiks/aws/archive/refs/he-
ads/main.zip\n"  # Download the zip
                - "unzip main.zip\n"  # Unzip the downloaded file
                 - "cp -r aws-main/* /var/www/html\n"  # Copy files from extracted
directory to the mounted directory
                   - "rm -rf aws-main main.zip\n"  # Remove unnecessary files and
directories
                 - "systemctl enable httpd\n"  # Enable Apache HTTP server to start
on boot
                - "systemctl start httpd\n"  # Start Apache HTTP server
```

Launch Template on EC2-instanssien resurssi, jossa määritellään sen asetukset. Se on malli tai pohja, jonka perusteella EC2-instansseja pystytetään. Olen käyttänyt Launchtemplatea, jotta saisin infrastruktuurini automatisoitua skaalauksen puitteissa.

Lyhyesti valitsemastani konfiguraatiosta:

- **AssociatePublicIpAddress: true**
  - Instanssi saa julkisen ipv4-osoitteen. Tämän avulla instanssiin on pääsy myös management consolesta ip-osoitteella. Tätä ei välttämättä tarvitsisi, sillä minulla on käytössä Loadbalancer, jolla on DNS-osoite.
- **ImageId: !Ref AMI**
  - Olen määrittänyt YAML-koodissani parametreihin oletusimagen, johon instanssit käynnistyvät. Viittaan tässä sen arvoon.

```
  AMI:
    Description: Launch EC2 instance from this AMI
    Type: AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>
    Default: /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2
```

  - Tarkoittaa siis, että käytössä on Amazon Linux.
- **InstanceType: !Ref InstanceType**
  - Olen määrittänyt YAML-koodissani parametreihin oletusarvon EC2-intanssin koolle/perheelle ja viittaan tässä sen arvoon.

```
  InstanceType:
    Description: EC2 instance family/size, e.g. t2.large
    Type: String
    Default: t2.micro
```

  - Tarkoittaa siis sitä, että EC2-instanssi on kokoa t2.micro.

- ***Koko**: T2.micro on pieni kokoinen instanssi, joka tarjoaa kohtuullisen laskentatehon ja muistin kevyisiin sovelluksiin ja testaukseen.*
- ***Käyttö**: Se soveltuu hyvin esimerkiksi kehitys- ja testaustyökaluille, pienille verkkosivustoille, ja muihin vähäisiä resursseja vaativiin käyttötarkoituksiin.*
- ***T2-perheen piirteet**: T2-instances kuuluvat AWS:n "T2-perheeseen", mikä tarkoittaa, että niiden suorituskyky perustuu burstaavaan CPU-tehoon, jossa instanssi voi väliaikaisesti käyttää enemmän laskentaresursseja, jos tarve vaatii.*
- ***Vapaa käyttö**: T2.micro-instanseilla on vapaa käyttötarjous, mikä tarkoittaa, että AWS tarjoaa tietyn määrän tuntikäyttöä ilmaiseksi tiettyjen ehtojen täyttyessä.*
- ***Rajoitukset**: T2.micro-instanseissa on rajoituksia, kuten rajoitettu verkkosuorituskyky ja tallennustila, mikä tulee ottaa huomioon suunnitellessa instanssien käyttöä.*

- **UserData script**
  - Lyhyesti: Instanssin käynnistyessä User Data scripti suorittaa bash-komentoina EFS:n mountaamisen hakemistoon /var/www/html ja sen jälkeen asentaa Apache httpd palveluun oman webbisivuni. Tarkemmat kuvaukset löytyvät itse scriptistä.

2.2.8 AutoScaling ja CloudWatch Alarm

```
AutoScalingGroup:
    Type: AWS::AutoScaling::AutoScalingGroup
    DeletionPolicy: Delete
    Properties:
      MinSize: '2'
      MaxSize: '4'
      DesiredCapacity: '2'
      LaunchTemplate:
        LaunchTemplateId: !Ref LaunchTemplate
        Version: !GetAtt LaunchTemplate.LatestVersionNumber
      HealthCheckType: ELB
      VPCZoneIdentifier:
        - !Ref PublicSubnet1
        - !Ref PublicSubnet2
      TargetGroupARNs:
        - !Ref ALBTargetGroups
ScalingPolicy:
  Type: 'AWS::AutoScaling::ScalingPolicy'
  Properties:
    AdjustmentType: ChangeInCapacity
    AutoScalingGroupName: !Ref AutoScalingGroup
    ScalingAdjustment: '1'
CloudWatchAlarm:
  Type: 'AWS::CloudWatch::Alarm'
  Properties:
    EvaluationPeriods: '1'
    Statistic: Average
    Threshold: '50'
    AlarmDescription: Alarm if CPU higher than 50%
    Period: '60'
    AlarmActions:
      - !Ref ScalingPolicy
    Namespace: AWS/EC2
    Dimensions:
      - Name: AutoScalingGroupName
        Value:
          Ref: AutoScalingGroup
    ComparisonOperator: GreaterThanThreshold
    MetricName: CPUUtilization
```

AutoScalingGroup määrittää automaattisen skaalausryhmän, joka hallinnoi EC2-instanssien määrää dynaamisesti. Templatessa määritän sen, että se käyttää aiemmin määritettyä Launchtemplatea EC2-instanssien luomiseksi, sekä minimäärä olemassa olevia instansseja on 2. Maksimirajan asetin 4:ään. Olen asettanut CloudWatchiin menevän hälytyksen, mikäli instanssien CPU-kapasiteetin

käyttö ylittää 50%. (Tässä tapauksessa pelkkä hälytys tulee CloudWatchiin, mutta toimintoja ei suoriteta, koska en ole määrittänyt toimintoa koodissa. Toiminto voisi laukaista sen, että AutoScalingGroup luo lisää instansseja.)

2.2.9 Application Load Balancer

```
ALBTargetGroups:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: !Sub MarppaTG
    VpcId: !Ref VPC
    TargetType: instance
    HealthCheckPort: 80
    Port: 80
    Protocol: HTTP
ALB:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Type: application
    Scheme: internet-facing
    SecurityGroups:
    - !Ref InstanceSecurityGroup
    Subnets:
    - !Ref PublicSubnet1
    - !Ref PublicSubnet2
    Tags:
      - Key: Name
        Value: MarppaALB
ALBlistener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    Protocol: HTTP
    Port: 80
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref ALBTargetGroups
    LoadBalancerArn: !Ref ALB
```

Loadbalancer ohjaa liikennettä EC2-instanssien välillä riippuen niiden reitityssäännöistä ja eheydestä. Se parantaa sovellusten käytettävyyttä, luotettavuutta ja suorituskykyä siten, että jos jokin instanssi häviää tai muuten on epäkunnossa, se osaa ohjata liikenteen eheälle ja toimivalle instanssille. Se myös jakaa liikenteen tasaisesti olemassa olevien instanssien välillä, joten surffailukokemus on saumaton.

Loadbalancereita on useanlaisia, mutta olen ottanut itse käyttöön Application Load Balancerin siksi koska minulla on sovellus, joka käyttää http-yhteyttä. Internet-facing LB:lla on julkinen DNS-osoite, jonka avulla käyttäjät voivat muodostaa yhteyden webpalveluun.

Target group tarkoittaa ryhmää, jota Loadbalancer haistelee sen varalta, että sen täytyy tehdä toimia. Olen asettanut niin, että luodut EC2-instanssit menevät automaattisesti target groupiin ja niistä tarkastellaan portista 80 kulkevaa liikennettä. Loadbalancer suorittaa ”health checkejä” instansseihin todetakseen niiden tilaa.

2.2.10 Outputs

```
Outputs:

  LoadBalancerDNS:
    Description: DNS address of the Load Balancer
    Value: !GetAtt ALB.DNSName
```

Kun stacki ajetaan ja se on suoritettu, sen tiedoissa on saatavilla outputs-osio. Olen asettanut niin, että sinne tuodaan Loadbalancerin luoma DNS-osoite, jolla pääsee katselemaan webbisivua. Helpottaa opettajan työtä, kun ei tarvitse lähteä seikkailemaan management consolessa.

3 Osa 2: Pilviarkkitehtuurin toteutus ja reflektio

3.1 Toteutetun pilviarkkitehtuurin monimutkaisuus

Oma arkkitehtuurini tuli lopulta suhteellisen monimutkaiseksi. Resurssien luominen tulee varmistaa niin, että ne tapahtuvat oikeassa järjestyksessä.

Haasteita, joita kohtasin oli mm:

- Tallennustilatyypin valitseminen automaattisesti skaalautuvaan ympäristöön.
    - Alun perin valitsemani EBS ei soveltunut automaattisesti skaalautuvaan ympäristöön, sekä EFS-ratkaisu on rahallisesti edullisempi.
- Osa resursseista oli hankala luoda YAML-templaattiin.
- YAML-formatointi tuotti suurta päänvaivaa.
- Learner Lab -ympäristössä puuttuvia oikeuksia (esim. CodeCommit ei ollut käytössä).

Well-Architected Frameworkin osalta lähdin suunnittelemaan arkkitehtuuria siten, että yritin tehdä kaiken mahdollisimman parhaiden käytäntöjen mukaan kuin mahdollista. Kyseessä on kuitenkin labraympäristö, rajalliset oikeudet ja resurssit, sekä sivustoni realiteetti kävijämäärässä ei todennäköisesti tulisi koskaan olemaan kovin iso. Tästä huolimatta halusin rakentaa arkkitehtuurin niin, että sen luomisesta on minulle hyötyä oppimisen kannalta.

1. **Operational Excellence:** IaC helpottaa ympäristön hallintaa ja toistettavuutta.
2. **Security**: Security Groupeilla on rajoitettu liikennettä ja varmistettu, että vain tarvittavat portit ovat avoimia. EFS käyttää salattua tiedonsiirtoa.
3. **Reliability:** Kaksi julkista aliverkkoa eri alueilla ja loadbalancer varmistavat korkean käytettävyyden ja vikasietoisuuden. Auto Scaling takaa resurssien riittävyyden ja vähentää ylikuormituksen riskiä.
4. **Performance Efficiency**: Loadbalancer ja Auto Scaling mahdollistaa dynaamisen resurssien hallinnan ja skaalautuvuuden → optimaalinen suorituskyky ja resurssien käyttö. EFS skaalautuu automaattisesti tarpeen mukaan.
5. **Cost Optimization:** Aikaisemmissa pilareissa mainitut resurssit auttavat optimoimaan kustannukset ja välttämään resurssien ylimääräistä käyttöä. Lisäksi CloudFormation template mahdollistaa ympäristön helpon hallinnan ja kustannusten seurannan.
6. **Sustainability:** ELB ja Autoscaling optimoivat resurssien käyttöä, joka vähentää ylimääräisiä ja turhia energiakustannuksia.

3.2 CloudFormation-templaattien käyttö ympäristön luomiseen

CloudFormation templaten käyttäminen nopeutti merkittävästi ympäristön pystyttämistä ja sen virhetilanteiden tutkimista ja korjaamista. Tietty oma aikansa meni, että sai templaten täysin toimivaksi, mutta nyt kun se on valmis niin pystytys on nopeaa. Templateja rakentaessa sai myös paljon paremman käsityksen siitä miksi ja miten eri resurssit toimivat ja liittyvät toisiinsa. Tätä samaa käsitystä ei välttämättä saa, jos vain kliksuttelee ohjauspaneelissa. Tosin jotkin resurssit olivat YAML:in kannalta sen verran kryptisiä, että piti käydä ohjauspaneelin kautta katsomassa asetuksia, että ymmärsi mitä syötettä oltiin vailla.

3.3 Arkkitehtuurikaavio

Katsoin mallia kursseilla tehdyistä labroista arkkitehtuurikaavion luomiseen. Kaavio auttaa ymmärtämään resurssien välisiä suhteita ja selkeyttää sitä, mitä kaikkea on käytössä.

3.4 Pohdinta käytetyistä palveluista

Tarkoitukseni oli alun perinkin hyödyntää aikaisemmin tekemääni webbisivua tämän työn kanssa. Alkuperäinen suunnitelma piti sisällään sen, että sivustolla olevasta yhteydenottolomakkeesta lähtisi triggeri (Lambda?), joka laukaisisi SNS-palvelun ja syötetyt tiedot tulisivat minulle sähköpostiin. En kuitenkaan ole niin koodiguru, että olisin onnistunut tässä ja vaihdoin ideaani alkuyritysten jälkeen.

Halusin tehdä arkkitehtuurin vikasietoiseksi ja automaattisesti skaalautuvaksi.

Tästä syystä VPC-alueellani on kaksi aliverkkoa. Aliverkot ovat eri AZ-alueilla vikasietoisuuden vuoksi. Kummassakin aliverkossa pyörii instanssit, joille on asennettu Apache httpd-palvelu. Palvelussa lepää portfoliosivustoni. Instanssit käyttävät jaettua tiedostojärjestelmää (EFS), joka mountautuu aina instanssin bootissa. Instanssit kuuluvat auto scaling groupiin AZ-alueiden ylitse. Auto scaling yhdistyy loadbalanceriin, joka jakaa liikennettä instansseille. Auto scaling hoitaa instanssien pystyttämisen launch templatesta, jos tarve vaatii. Loadbalancer toimii yhteydessä Internet Gatewayn kanssa, josta pyynnöt tulevat julkisiin verkkoihin.

Sivustoni pakattu tiedosto hakeutuu Githubista, omasta repositoriostani, joka on asetettu julkiseksi. Tämä ei tietenkään oikeasti olisi kovin ideaalia. Repositorioon tulisi luoda avaimet ja muut killuttimet, jotta se ei olisi julkisesti saatavilla. Tämän tehtävän kannalta näin kuitenkin on.

Tehtävä oli hauska toteuttaa, vaikka muutama ärräpää pääsikin templaten kanssa painiessa. Opin runsaasti palveluista ja niiden yhteyksistä! Ei harmita, että tuli kokeiltua erilaisia vaihtoehtoja.