

תוכן עניינים

2.....	הצעת הפרויקט.....
5.....	טופס הצהרת.....
6.....	אישור מקום וה坦נסות.....
7.....	רקע/מבוא - הכרת הארגון וסביבה העבודה.....
8.....	סיכום דוחות ופגישות.....
8.....	דוח שבועי 1.....
9.....	דוח שבועי 2.....
10.....	דוח שבועי 3.....
11.....	דוח שבועי 4.....
12.....	הציגת המשימות.....
12.....	1. מבוא ורקע לפרויקט.....
13.....	2. תכנון וארQUITקטורה המערכת.....
13.....	3. שלב 1-3: בניית התשתייה הבסיסית.....
24.....	4. שלבים 4-5: Load Balancing-Application Layer.....
27.....	5. שלב 6: Secret Management.....
29.....	6. שלבים 7-8: CI/CD Automation-GitOps.....
33.....	7. תחילת הפרישה המלא.....
33.....	8. ניטור-Observability.....
34.....	9. אבטחה-Compliance.....
34.....	10. טיפול בעוות נפוצות-Troubleshooting.....
35.....	11. מסקנות וכיווני פיתוח עתידיים.....
36.....	נו ^פ חים.....
43.....	12. סיכום.....
44.....	מקורותביבליוגרפיה.....
45.....	מצגת הפרויקט.....

רקע/מבוא - הכרת הארגון וסביבת העבודה

בינת תקשורת מחשבים הינה חברה מובילה בתחום פתרונות מתקדמים בתחום ה-IT, התקשות, הענן ובטיחת המידע, ופועלת מול מגוון רחב של לקוחות במגזרים הציבורי, הביטחוני והעסק' בישראל. החברה מספקת שירותי מקצוע לאקצה – החל משלב התכנון והאפיון, דרך הקמה וההטמעה, ועד לניהול ותחזוקה שוטפת של מערכות ותשתיות. צוותי החברה מתחמם ביישום טכנולוגיות מתקדמות ובהתקמת פתרונות ייעודיים לדרישות הלוקח, תוך שמירה על סטנדרטים גבוהים של ביצועים ובבטיחה.

חטיבת **DevOps** בbynת אחראית על פיתוח והטמעת תהליכי אוטומציה, ניהול תשתיות ענן מודרניות ושיפור מתודולוגיות פיתוח ופריסת בארגונים. החטיבה פועלת בסביבה טכנולוגית עדכנית, תוך שימוש בכלים ובמתודולוגיות מעולמות **Agile**-**Cloud**, **DevOps**, המישת פתרונות המבוססים על Kubernetes, Kubernete, ניהול תצורה, תשתיות כהוד (CaaS) וכל CI/CD מתקדמים.

הפרויקט הנוכחי תוכנן במטרה לספק ללקוח פתרון מלא ומקייף העונה על צרכי הארגונים והטכנולוגיות. הדרישת המרכזית היינה הטמעת ארכיטקטורת **Zero Trust Network**, המאפשרת בקרת גישה קפדנית המבוססת על זהות, מצב המכשיר והקשר הגישה, תוך שימוש מרבי של שטח החשיפה ומינעת גישה בלתי מורשית למשאים רגילים. הפתרון כולל מנגנוני ניהול הרשות מתקדמים והגנה רב-שכבותית.

בנוסף, הוגדרה דרישת להקמת סביבת **Kubernetes** בענן **AWS**, שתאפשר גמישות תפעולית, ניהול משאביים דינמי ויכולות אוטומציה מלאות לפריסת אפליקציות ותשתיות. סביבת Kubernetes נבנתה עם מנגנון ניהול סודות מאובטח (Secret Management), תמייה מלאה במתודולוגיית **GitOps** לניהול גרסאות מבוקר, ושליטה מלאה בתהליכי עדכון והחזרת גרסה במקרה של תקלות.

תהליך ההקמה כלל תכנון ארכיטקטורה מאובטחת מבוססת עקרונות **Infrastructure as Code**. באמצעות כלים כגון Terraform ו-Helm והטמעת מנגנוני CI/CD לשיפור מהירות ואיכות ההפצת הפתרון נבנה כך שיספק **zmינות גבוהה** (High Availability), יתמוך בדרישות תאימות רגולטוריות ויאפשר תחזוקה ותפעול יעילים למשך זמן, עם יכולת הרחבת עתידית בהתאם לצרכי הארגון.

ויקומי דו"חות ופתרונות

דו"ח שבועי 1 – הכנות תשתיית AWS ובסיס ZTN

מטרות: להקים את התשתייה בunken ולהתחל בהגדרת חיבור רשות מאובטחים.

תכלות:

AWS Environment Setup .1

- הגדרת ארכיטקטורת תשתיות: VPC, Subnets, Routing Tables, Internet Gateway, NAT Gateway.

כתיבת מודולים ב-Terraform לפריסה של:

- VPC ותתי-ירשות
- Security Groups ו-Routing בסיסים
- EKS Cluster בסיס לתקמת

Secure Connectivity via Zero Trust Network .2

- בחירת ספק Twingate והבנת הדרישות.
- פריסת ZTN Connector ב-VPC.
- קונפיגורציה ראשונית של מדיניות גישה מאובטחת (Access Policies).

3. תיעוד שבועי

- תרשימי רשות (VPC/Subnets/Connector)
- קבצי Terraform בסיסים
- דוגמה למדיניות NTZ בסיסית

תוצרים:

- EKS Skeleton Terraform modules בסיסים לפריסת VPC ו-NTZ
- חיבור ZTN ראשוני לעבודה

דוח שבועי 2 – פריסת EKS וkonfiguracija בסיסית

מטרות: להקים ולהגדיר את EKS עם ניהול Nodes ותשתיות בסיסית ל-ingress.

תכליה:

Kubernetes (EKS) Configuration .1

- Terraform EKS cluster-לBootstrap
- הגדרת(Cluster Access) kubectl, AWS IAM
- konfiguracija של Autoscaling Node Groups ו-ingress

Ingress and Load Balancer .2

- nginx Ingress Controller
- הגדרת Load Balancer בסיסי
- konfiguracija של Route DNS ב-53 אם דרוש

3. תיעוד שבועי

- תרשימים עם Cluster
- דוגמה לקובץ config kubectl
- תרשימים Ingress + Load Balancer

תוצריים:

- Node Groups עובד עם Cluster
- Load Balancer Ingress Controller בסיסי ו-ingress
- גיש נגיש בצורה מאובטחת דרך ZTN Cluster

דוח שבועי 3 – פרישת אפליקציות וניהול סודות

מטרות: לפרוס אפליקציות בצורה מאובטחת ולנהל סודות בצורה אוטומטית.

תכליה:

Application Deployments .1

- תבניות Helm עבור האפליקציות

- קונפיגורציה של Deployment, Service, Ingress

Secrets Management .2

- בחירת כלי לניהול סודות (External Secrets Operator או AWS Secrets Manager)

- אוטומציה של הזרקת סודות ל-Kubernetes

- יישום מדיניות סיבוב סודות (Rotation Policy)

3. תיעוד שבועי

- דוגמאות Helm Charts

- תרשימים זרימת סודות

- מדיניות סיבוב והזרקה

תוצריים:

- אפליקציות פרוסות דרך Helm
- סודות מוגדרים ומסודרים עם רוטציה אוטומטית

דוח שבועי 4 – GitOps, אוטומציה ותיעוד סופי

מטרות: להפוך את כל הסביבה לאוטומטית, מבוקרת ומוסדרת ב-GitOps.

תכלות:

GitOps with Argo CD .1

- התקנת Argo CD ב-EKS
- ייצור Argo CD applications לכל פרויקט/אפליקציה
- חיבור ל-source of truth Git repository

Automation & Documentation .2

- אחסון Helm ו-Terraform בקוד Git version-controlled
- אוטומציה של פריסת תשתיית אפליקציות דרך CI/CD
- תיעוד סופי

Infrastructure design ■

Access instructions ■

Secrets strategy ■

Argo CD app structure ■

3. תוצרים סופיים

- סביבה פרוסה ומואבטחת עם Zero Trust
- Argo CD מלא עם GitOps
- תיעוד מסודר וברור

הציגת המשימות

Cloud Infrastructure Automation with GitOps

פרויקט סיום - אוטומציה של תשתיות ענן עם שיטות GitOps

1. מבוא ורקע לפרויקט

1.1 תיאור כללי

בעולם הפיתוח המודרני, ארגונים רבים מאמצים גישות DevOps וענן ציבורי כדי להאיץ את תהליכי הפיתוח והפרישה. פרויקט זה מתמקד בبنיה תשתיות ענן מתקדמת ומأובטחת על גבי AWS, תוך שימוש של **Infrastructure as Code**, Kubernetes, Zero Trust Network, GitOps ו-CloudWatch Metrics.

המטרה העיקרית היא ייצירת סביבת פיתוח וprd קפיצה מאובטחת ואוטומטית שמאפשרת לצוותי הפיתוח לפתח ולנהל אפליקציות בצורה יעילה ובוטיחה.

1.2 מטרות הפרויקט

- **אבטחה מתקדמת:** יישום עקרונות Zero Trust Network לגישה מאובטחת למשאים
- **אוטומציה מלאה:** הקמת pipeline CI/CD GitOps מלא עם AWS Lambda ו-AWS Secrets Manager
- **ניהול סודות מתקדם:** יישום Secrets Management עם AWS KMS
- **קונטינריזציה:** פריסת אפליקציות באמצעות Docker容器 ו-Kubernetes
- **Terraform Infrastructure as Code:** ניהול התשתיות באמצעות Terraform

1.3 ארכיטקטורה כללית

הפתרון מתבסס על ארכיטקטורת microservices המופעלת על Elastic Kubernetes Service, עם גישה מאובטחת דרך Zero Trust Network ואוטומציה מלאה באמצעות GitOps.

2. תכנון וארQUITטורת המערכת

2.1 רכיבי התשתיות העיקריים

2.1.1 רשת ואבטחה

- **VPC:** רשת פרטית מבודדת עם subnets רבים
- **Security Groups:** חוקי firewall ברמת הרשת
- **Zero Trust Network - Twingate:** גישה מאובטחת למשאים פרטיים
- **NAT Gateway:** גישה אינטראנט יוצאת לרכיבים פרטיים

Container Orchestration- Compute 2.1.2

- **EKS Cluster:** AWS Kubernetes ב
- **EC2 Node Groups:** EC2: קבוצות nodes עם Auto Scaling
- **EC2 Connector Instance:** מכונה ייעודית עבור Twingate Connector

2.1.3 רכיבי פירסה ואוטומציה

- **Application Load Balancer:** חלוקת עומס וחשיפת אפליקציות
- **NGINX Ingress Controller:** ניתוב תעבורה בתוך הקלוסטר
- **ArgoCD:** אוטומציה של הקלוסטר עם gitflow - GitOps continuous deployment
- **Helm Charts:** ניהול deployment של אפליקציות

3. שלב 3-1: בניית התשתיות הבסיסית

3.1 הקמת סביבת AWS (משימה 1)

3.1.1 ארכיטקטורת הרשת

בשלב זה מוקמת התשתיות הבסיסית של AWS באמצעות Terraform modules המספקים:

:Networking-ו VPC

- **(10.0.0.0/16** עם CIDR block מתאים (לדוגמה: NAT Gateway-ו Load Balancer Public subnet •
- **EC2 connector-ו EKS nodes** Private subnets •
- **Internet Gateway** לגישת אינטרנט •
- **private subnets NAT Gateway** יוצאת מ- •
- **subnet מתאימות לכל Route Tables** •

:Security Groups

- עם חוקים מתאימים EKS cluster security group
- Node groups security groups
- Twingate עם פתיחות נדרשות EC2 connector security group

:Terraform קוד דוגמת

```
#Creating main VPC.
resource "aws_vpc" "main" {
  cidr_block          = var.vpc_cidr
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "${var.project_prefix}-vpc"
  }
}

# Creating Internet Gateway for the VPC.
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "${var.project_prefix}-igw"
  }
}
```

```
# Creating Internet Gateway for the VPC.
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "${var.project_prefix}-igw"
  }
}
```

```
# Creating public subnets.
resource "aws_subnet" "public" {
    count           = length(var.public_subnet_cidrs)
    vpc_id          = aws_vpc.main.id
    cidr_block      = var.public_subnet_cidrs[count.index]
    map_public_ip_on_launch = true
    availability_zone = var.availability_zones[count.index]

    tags = {
        Name = "${var.project_prefix}-public-${count.index}"
    }
}
```

```
# Creating private subnets.
resource "aws_subnet" "private" {
    count           = length(var.private_subnet_cidrs)
    vpc_id          = aws_vpc.main.id
    cidr_block      = var.private_subnet_cidrs[count.index]
    availability_zone = var.availability_zones[count.index]
    map_public_ip_on_launch = false

    tags = {
        Name = "${var.project_prefix}-private-${count.index}"
    }
}
```

```
# Creating route table
resource "aws_route_table" "public" {
    vpc_id = aws_vpc.main.id

    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.igw.id
    }

    tags = {
        Name = "${var.project_prefix}-rt-public"
    }
}

# Associating public subnets with the public route table.
resource "aws_route_table_association" "public" {
    count          = length(aws_subnet.public)
    subnet_id     = aws_subnet.public[count.index].id
    route_table_id = aws_route_table.public.id
}

# Creating NAT Gateway for private subnets.
resource "aws_eip" "nat_eip" {
    domain = "vpc"

    tags = {
        Name = "${var.project_prefix}-nat-eip"
    }
}
```

```
# Creating NAT Gateway for private subnets.
resource "aws_eip" "nat_eip" {
  domain = "vpc"

  tags = {
    Name = "${var.project_prefix}-nat-eip"
  }
}

# Creating NAT Gateway resource.
resource "aws_nat_gateway" "nat" {
  allocation_id = aws_eip.nat_eip.id
  subnet_id     = aws_subnet.public[0].id
  depends_on    = [aws_internet_gateway.igw]

  tags = {
    Name = "${var.project_prefix}-nat-gw"
  }
}
```

```
# Creating route table for private subnets.
resource "aws_route_table" "private" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block      = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.nat.id
  }

  tags = {
    Name = "${var.project_prefix}-rt-private"
  }
}

# Associating private subnets with the private route table.
resource "aws_route_table_association" "private" {
  count          = length(aws_subnet.private)
  subnet_id     = aws_subnet.private[count.index].id
  route_table_id = aws_route_table.private.id
}
```

```

module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "20.37.1"

  cluster_name      = var.eks_cluster_name
  cluster_version   = var.cluster_version
  vpc_id            = var.vpc_id
  subnet_ids        = var.subnet_ids
  enable_irsa       = true

  cluster_endpoint_private_access = true

  eks_managed_node_groups = {
    menahem-einav_ng = {
      min_size      = var.node_group_min_size
      desired_size   = var.node_group_desired_size
      max_size      = var.node_group_max_size
      instance_types = var.node_group_instance_types
      subnet_ids     = var.subnet_ids
      key_name       = "menahem-einav"
      lifecycle = {
        create_before_destroy = true
      }
    }
  }
  tags = {
    Environment = var.environment
    Project     = var.project_name
  }
}

```

3.2 יישום Zero Trust Network - Twingate (משימה 2)

3.2.1 עקרונות Zero Trust

Zero Trust Network מבוסס על העיקרון "Never Trust, Always Verify". בפרויקט זה, Twingate משמש כפתרון ZTN המספק:

- **אימות מתמיד:** כל גישה דורשת אימות וחד פעמי
- **גישה מינימלית:** משתמשים מקבלים גישה רק למשאים הנדרשים להם
- **הצפנה מקצה לקצה:** כל התעבורה מוצפנת

בפרט, על ה ec2 יש twingate connector שדרכו אפשר להתחבר למשאים ברשת הפתוחה (זה נעשה עם twingate client). בפרט, ה connector מיועד כדי להתחבר ל eks (לדוגמא עם kubectl). מיועד בשביל התקשורת בטוחה של הצוות devops. (במקרה שלנו ה twingate)

Twingate Connector 3.2.2

הacketן על private subnet EC2 instance Twingate Connector הפנימיים:

תהליך התקנה:

1. ייצרת private subnet EC2 instance ב-
2. התקנת Twingate Connector Docker באמצעות Twingate Management Console-Connector
3. רישום ה Twingate Management Console-Connector ב-Connector EKS
4. הגדרת Access Policies עבור משאבי EKS

דוגמת קונפיגורציה עם Terraform :

```

```
EC2 Twingate Connector Instance

Latest ami
data "aws_ami" "ubuntu" {
 most_recent = true

 filter {
 name = "name"
 values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
 }

 filter {
 name = "virtualization-type"
 values = ["hvm"]
 }
 owners = ["099720109477"] # Canonical
}

Remote Network
resource "twingate_remote_network" "this" {
 name = var.twingate_remote_network_name
}

Connector
resource "twingate_connector" "this" {
 name = var.twingate_connector_name
 remote_network_id = twingate_remote_network.this.id
}

Tokens
resource "twingate_connector_tokens" "this" {
 connector_id = twingate_connector.this.id
}

Security Group
resource "aws_security_group" "twingate_sg" {
 name = "${var.project_prefix}-twingate-sg"
 description = "Allow Twingate outbound access"
 vpc_id = var.vpc_id

 egress {
 from_port = 0
 to_port = 0
 protocol = "-1"
 cidr_blocks = ["0.0.0.0/0"]
 }

 ingress {
 from_port = 22
 to_port = 22
 protocol = "tcp"
 cidr_blocks = ["0.0.0.0/0"]
 }

 tags = [
 Name = "${var.project_prefix}-twingate-sg"
]
}
```

```

```

# -----
# Template File (outside)
# -----
data "template_file" "twingate_user_data" {

  depends_on = [twingate_connector_tokens.this]
  template = file("../scripts/twingate-run-connector.sh")

  vars = {
    TWINGATE_ACCESS_TOKEN  = twingate_connector_tokens.this.access_token
    TWINGATE_REFRESH_TOKEN = twingate_connector_tokens.this.refresh_token
    TWINGATE_NETWORK       = var.twingate_network
    TWINGATE_LABEL         = var.twingate_connector_name
  }
}

# -----
# EC2 Instance
# -----
resource "aws_instance" "twingate_connector" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"
  subnet_id     = var.private_subnet_ids[0]
  vpc_security_group_ids = [aws_security_group.twingate_sg.id]
  key_name      = "menahem-einav"

  user_data = base64encode(data.template_file.twingate_user_data.rendered)
  user_data_replace_on_change = false
  depends_on = [data.template_file.twingate_user_data]

  tags = {
    Name = "${var.project_prefix}-twingate-ec2"
  }
  lifecycle {
    create_before_destroy = true
  }
}

# Resource: EKS API internal access
resource "twingate_resource" "eks_api" {
  name          = "${var.project_prefix}-eks-api"
  address       = "10.0.0.0/8"
  remote_network_id = twingate_remote_network.this.id
  lifecycle {
    ignore_changes = [
      access_group
    ]
  }
}

resource "aws_security_group_rule" "allow_twingate_to_eks_https" [
  type          = "ingress"
  from_port     = 443
  to_port       = 443
  protocol      = "tcp"
  security_group_id = var.eks_security_group_id
  source_security_group_id = aws_security_group.twingate_sg.id
]
}

```

הקובץ הוא סקrypt אתחול (*user data script*) שמריץ ב-EC2 את תהליך ההתקנה (Access/Refresh Tokens), באמצעות פרטיה הגישה של Twingate Connector Docker container. ראה בתוכף Twingate API קיבל מה-Terraform.

3.3 הקדמה למשימות 3, 4 ו 5

משימות 3, 4 ו 5 זה למעשה האפליקציה שהמשתמשים יכולים לגשת אליה. בשביל שהמשתמש יוכל לדבר עם האפליקציה אנו בונים Load Balancer או ingress cluster. המשתמש מתקשר עם LB והוא מתקשר עם האפליקציה שבתוך הcluster. האפליקציה נשמרת בdocker container docker image אנו שומרים בhub .helm chart את ה deployment של האפליקציה אנו עושים עם .helm values את ה docker tag אנו מעדכנים דרך ה .helm values.

הקוד של האפליקציה (וגם המרחב github repo) נשמר בcharts manifests.private. כתע נניח שצורת הפיתוח רצים לדחוף שינוי באפליקציה. אם נרצה שהשינוי יבוצע בcluster, נctrkr לעשות את הפעולות הבאות.

1. לבנות את האפליקציה. זה אומר לבנות tag חדש (ופועלות נוספות בהתאם לאפליקציה).
2. לדחוף את התג החדש לhub docker.
3. לעדכן את המרחב helm chart values (בgithub repo) להשתמש בתג החדש.
4. לחת את המערך המעודכן מתוך github ולבנות עמו זה upgrade app בcluster. זהו! העדכון הושלם בהצלחה.

כמובן שלא נרצה לעשות את זה ידנית כל פעם. לכן נרצה לעשות תהליכי אוטומציה לזה. כאן נכנס לתמונה משימות 6, 7 ו 8 (Automation).

אנו משתמשים בgithub actions לצורך הבא; כאשר צוות הפיתוח עשו git push לאפליקציה אז הworkflow יבנה את docker image עם תג חדש (מודגר לפי תאריך שעה ושנה), ידחוף לhub docker עם התג החדש (זה נעשה באמצעות שמירת token בgithub secrets). יעדכן את הקובץ helm/app/values.yaml עם התג החדש.

זהו, פעולות 1, 2 ו 3 הושלמו. כתע נשר לעדכן את הcluster, גם באופן אוטומציה כמו כן. לצורך כך משתמש בargo cd. אני יזכיר שככל המニアטוט ישב על private repo. אז בשבייל שargo יכול למשוך את המニアטוט helm/app/values.yaml. את token נשמר בcluster secrets. אם נעשה את זה בצוורה ישירה, יוכל לעשות זאת בשני דרכיהם:

1. שירות. לדוגמה עם kubectl create secret token token_name --dry-run --export --apply.
2. לעשות token.secret.yaml, לשמר את token שם ואז לעשות לקובץ apply. דרך הראשונה לא מספיק טובה לנו, כי נרצה לשמור את token (וכן את שאר הסודות) בצוורה מסודרת. הדרך השנייה גם לא מספיק טובה, מכיוון שאנו שומרים את token בקובץ yaml (בbase 64) והקובץ יושב בrepo github יוציא שכל מי שיש גישה לrepo יוכל לראות את כל הסודות ששמרנו בקובץ yaml (אפילו זה repo פרטי, זה לא דרך בטוחה לשמר סודות).

הפתרון לפROYיקט זה הוא AWS Secrets Manager. AWS Secrets Manager עובד כך: יש secret name (לפעמים נקרא key), ולו יש רשימה של זוגות key-value.

לדוגמא, נניח שהאפליקציהutzיה משתמשת ב data base data base ולצורך התחברות נדרש:

1. USER_NAME
2. PASSWORD
3. HOST
4. NAME
5. TOKEN

از נוכל ליצור secret בשם DB וכל אחד מהרשימה יהיה key, והערך שלו יהיה value (לדוגמא, key זה root ו value זה USER_NAME).

במקרה שלנו זה יראה כך:

secret name: project-name

key: url	value: github-url
key: username	value: github-username
key: password	value: github-argo-token

בשביל ש argocd יוכל להשתמש בהזה, נדרש למשוך את כל המידע הזה ל cluster אשר הוא עושים עם אובייקט של kubernetes שנקרא external-secrets.io/v1. external-secrets.io/v1

בסוף דבר argocd יודע לדבר עם kubernetes secret וכן מה שבפועל קורה זה שה secrets Manager AWS Secrets Manager יוצר external-secrets secret מותאם ואז argocd משתמש בהזה כדי להתחבר לרפזיטורי הפרט.

זהו, כתעת ש argocd יכול לגשת לרפזיטורי; argocd מושך את values.yaml המעודכן ומעדכן את ה helm chart (האפליקציה).
וຍימנו גם את שלב 4 באוטומציה שלנו.

3.4 הקמת EKS Cluster (משימה 3)

EKS Cluster Configuration 3.4.1

מקום עם הגדרות הבאות:

:Cluster Features

- Kubernetes גרסה יציבה (1.29)
- Twingate דרך Private endpoint access

:Node Groups

- instance type t3.medium
- (min: 1, max: 3, desired: 2) מוגדר Auto Scaling

EKS Bootstrap Terraform 3.4.2

```

module "eks" {
  source  = "../../modules/eks"
  vpc_id    = module.network.vpc_id
  subnet_ids = module.network.private_subnet_ids

  eks_cluster_name      = var.eks_cluster_name
  cluster_version       = var.cluster_version

  node_group_desired_size = var.node_group_desired_size
  node_group_max_size     = var.node_group_max_size
  node_group_min_size     = var.node_group_min_size
  node_group_instance_types = var.node_group_instance_types

  environment           = var.environment
  project_name          = var.project_prefix
}

```

4. שלבים Load Balancing-Application Layer :4-5

(משימה 4) Load Balancer-Ingress 4.1

4.1.1 ארכיטקטורת התעבורה

זרימת התעבורה מהמשתמש לאפליקציה:

Internet → LB → NGINX Ingress → Service → Pod

רכיבים עיקריים:

- **Load Balancer**: נקודת הכניסה החיצונית
- **NGINX Ingress Controller**: מ途וב תעבורה בתוך הקלוסטר
- **Kubernetes Services**: חיבור pods פנימית

NGINX Ingress Controller Deployment 4.1.2

```

controller:
  progressDeadlineSeconds: 600
  ingressClassResource:
    name: nginx
    enabled: true
    default: true
  service:
    type: LoadBalancer
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
    service.beta.kubernetes.io/aws-load-balancer-tags: "Environment=dev,Owner=mendel-einav"

# this is how to add ingress-nginx

# helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
# helm repo update

# helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx \
#   --namespace ingress-nginx \
#   --create-namespace \
#   -f values-ingress-controller.yaml

```

4.2 פרישת אפליקציות (משימה 5)

4.2.1 אסטרטגיית Container Management

אפליקציות מנהלות C-containernized microservices

Container Registry Strategy

- Docker Hub Docker images •
- timestamp אוטומטי בהתקבוס על Image tagging •
- לאופטימיזציה Multi-stage builds •
- images של Security scanning •

Helm Charts Structure 4.2.2

```

app/
  └── Chart.yaml
  └── templates
    ├── deployment.yaml
    ├── ingress.yaml
    └── service.yaml
  └── values.yaml

```

זה דוגמא למבנה פשוט, אבל יכול להיות בtemplates עוד קבצים.

Application Helm Template Example 4.2.3

FILE: templates/ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ .Values.ingress.name | default (printf "ingress-%s" .Values.deployment.container.name) }}
  namespace: {{ .Values.namespace }}
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: {{ .Values.ingress.ingressClassName }}
  rules:
    - {{- if .Values.ingress.host }}
      host: {{ .Values.ingress.host }}
      {{- end }}
      http:
        paths:
          - path: {{ .Values.ingress.path }}
            pathType: Prefix
            backend:
              service:
                name: {{ .Values.service.name | default (printf "service-%s" .Values.deployment.container.name) }}
                port:
                  number: {{ .Values.service.port }}
```

FILE: templates/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.deployment.name }}
  namespace: {{ .Values.namespace }}
  labels:
    app: hello-selector
spec:
  replicas: {{ .Values.deployment.replicas }}
  selector:
    matchLabels:
      app: hello-selector
  template:
    metadata:
      labels:
        app: hello-selector
    spec:
      containers:
        - name: {{ .Values.deployment.container.name }}
          image: {{ .Values.deployment.container.image }}
          imagePullPolicy: Always
          ports:
            - containerPort: {{ .Values.deployment.container.port }}
```

FILE: templates/service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.service.name | default (printf "service-%s" .Values.deployment.container.name) }}
  namespace: {{ .Values.namespace }}
spec:
  selector:
    app: hello-selector
  ports:
    - protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: {{ .Values.deployment.container.port }}
```

5. שלב 6 : Secrets Management

AWS Secrets Manager Integration 5.1

5.1.1 AWS Secrets Manager Integration

הפתרון לניהול סודות מtabased על שכבות אבטחה מרובות:

AWS Secrets Manager → External Secrets → K8s Secret → Application/Argocd

יתרונות הגישה:

- **הפרדת סודות מקוד:** אין secrets בrepository secrets_rotation AWS Secrets Manager
- **Automatic Rotation:** תומך AWS Secrets Manager

AWS Secrets Manager Structure 5.1.2

secret name: project-name

key: url	value: github-url
key: username	value: github-username
key: password	value: github-argo-token

External Secrets Operator 5.2

Configuration-ו External Secrets Installation 5.2.1

הפקודות הבאות הם לצורך התקנת ה-ExternalSecrets :cluster

helm repo add external-secrets https://charts.external-secrets.io

```
helm install external-secrets \
external-secrets/external-secrets \
-n external-secrets \
```

```
--create-namespace \
.ExternalSecret | ClusterSecretStore |
```

בשביל למשוך את הסודות, אנו צריכים ClusterSecretStore או ExternalSecret.

מגדר איך ומאייפה לקבל את הסודות (AWS Secrets Manager, עם פרטי הגישה). ClusterSecretStore: מגדיר איזה סוד להביא ויכן ליצור את הסוד בתוך קוברניטו. ExternalSecret:

```
##### FILE: external-secrets/cluster-secret-store.yaml #####
```

```
apiVersion: external-secrets.io/v1
kind: ClusterSecretStore
metadata:
  name: argo-cluster-secret-store
spec:
  provider:
    aws:
      service: SecretsManager
      region: eu-central-1
      auth:
        secretRef:
          accessKeyIDSecretRef:
            name: access-secret
            key: access-key
            namespace: external-secrets
          secretAccessKeySecretRef:
            name: access-secret
            key: secret-access-key
            namespace: external-secrets
```

```
##### FILE: external-secrets/ExternalSecret.yaml #####
```

```

apiVersion: external-secrets.io/v1
kind: ExternalSecret
metadata:
  name: argo-extenal-secret
  namespace: external-secrets
spec:
  refreshInterval: 72h
  secretStoreRef:
    name: argo-secret-store
    kind: SecretStore
  target:
    name: kube-argo-secret-value
    creationPolicy: Owner
  dataFrom:
    - extract:
        key: rickroll-project-menahem-einav

```

6. שלבים 7-8 : CI/CD Automation- GitOps

(7) ArgoCD עם GitOps 6.1

Principles ו ArgoCD Architecture 6.1.1

:ArgoCD מימוש את עקרונות GitOps

:Core Principles

- **Git as Single Source of Truth**
- **Declarative Configuration**
- **Automated Sync**
- **Observability**

Configuration ו ArgoCD Installation 6.1.2

:cluster נדרה הקיימת כדי argocd

```

kubectl create namespace argocd
helm install argocd argo/argo-cd -n argocd --create-namespaces

```

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: app-name
  namespace: argocd
  labels:
    app.kubernetes.io/managed-by: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default

  source:
    repoURL: 'https://github.com/your-repo-url.git'
    targetRevision: your-target-branch
    path: helm/app
    helm:
      valueFiles:
        - values.yaml

  destination:
    server: https://kubernetes.default.svc
    namespace: app-ns

  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
```

Private Repository Access Configuration 6.1.3

```
# external-secrets/argo-repo-secret.yaml
apiVersion: external-secrets.io/v1
kind: ExternalSecret
metadata:
  name: me-repo-secret
  namespace: argocd
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: argo-cluster-secret-store
    kind: ClusterSecretStore

  target:
    name: me-repo-secret
    creationPolicy: Owner
    template:
      metadata:
        labels:
          argocd.argoproj.io/secret-type: repository
      type: Opaque
      data:
        url: "{{ index . \"github-url\" }}"
        username: "{{ index . \"github-username\" }}"
        password: "{{ index . \"github-argo-token\" }}"

    dataFrom:
      - extract:
          key: your-aws-secrets-manager-secret-name
```

(משימה 8) GitHub Actions עם CI/CD Pipeline 6.2

6.2.1 תזכורת ל 3.3 הקדמה למשימות 3 4 ו 5

ב 3.3 דיברנו על המקרה שצווות הפיתוח דוחפים שינוי לאפליקציה.

אמרנו שנצרך לעשות אוטומציה לשילבים הבאים

1. לבנות את האפליקציה. זה אומר לבנות docker image tag חדש (ופועלות נוספות בהתאם לאפליקציה).

2. לדוחוף את התג החדש ל docker hub.

3. לעדכן את ה helm chart values (ב github repo) להשתמש בתג החדש.

4. לחתוך את ה values המעודכן מתוך github וילעשות עם זה upgrade ב app helm cluster.

נשים לב ששאτ שלב 4 כבר השלמנו עם AWS Secrets Manager ו argocd

נשאר לנו להשלים את שלבים 1 2 ו 3.

GitHub Actions Workflow 6.2.2

FILE: .github/workflows/deploy.yaml

```

name: Build and Deploy to EKS

on:
  push:
    paths:
      - 'website/**'
  branches:
    - your-target-branch
  workflow_dispatch:

permissions:
  contents: write

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4
        with:
          token: ${{ secrets.GITHUB_TOKEN }}
          fetch-depth: 0

      - name: Get current date
        id: date
        run: echo "today=$(date +'%Y-%m-%d-%H-%M-%S')" >> "$GITHUB_OUTPUT"

      - name: Login to DockerHub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and push Docker image
        env:
          IMAGE_NAME: your-image-name
          IMAGE_TAG: ${{ steps.date.outputs.today }}
        run:
          docker build -t $IMAGE_NAME:$IMAGE_TAG ./website
          docker push $IMAGE_NAME:$IMAGE_TAG

      - name: Update image tag in values.yaml
        run:
          sed -i "s/tag::*/tag: \"${{ steps.date.outputs.today }}\"/" ./helm/app/values.yaml

      - name: Commit and push updated values.yaml
        env:
          GH_PAT: ${{ secrets.GH_PAT }}
          IMAGE_TAG: ${{ steps.date.outputs.today }}
        run:
          git config user.name "github-actions[bot]"
          git config user.email "github-actions[bot]@users.noreply.github.com"
          git add ./helm/app/values.yaml
          git commit -m "Update image tag to $IMAGE_TAG [skip ci]" || echo "No changes to commit"
          git remote set-url origin https://x-access-token:$GH_PAT@github.com/${{ github.repository }}
          git push origin HEAD:your-target-branch

```

זהו! סימנו את כל שלבי האוטומציה.

7. תהליכי הפרisa המלא

Development Workflow 7.1

graph TD

```

A[Developer Push] --> B[GitHub Actions Triggered]
B --> C[Run Tests & Build]
C --> D[Build Docker Image]
D --> E[Push to Docker Hub]
E --> F[Update Helm Values]
F --> G[Commit Changes]
G --> H[ArgoCD Detects Changes]
H --> I[Pull Updated Manifests]
I --> J[Deploy to Kubernetes]
J --> K[Application Updated]

```

Deployment Stages 7.2

Publish-& Build :1-3 - GitHub Actions 7.2.1

unit tests, integration tests : Code Testing	.1
בנית image עם tag חדש : Docker Build	.2
העלאת Docker Hub- image ל- : Registry Push	.3
עדכון values.yaml עם tag חדש : Helm Update	.4

GitOps Sync :4 - ArgoCD 7.2.2

Git repository Change Detection: ArgoCD	.1
משיכת המניפסטים המעודכנים : Manifest Pull	.2
AWS-n secrets Secret Resolution: External Secrets	.3
פריסת האפליקציה המעודכנת : Application Deployment	.4

8. ניטור ו-Observability

Monitoring Stack 8.1

ArgoCD Dashboard 8.1.1

- סטטוס sync של כל האפליקציות
- היסטוריית deployments
- resources Health status
- Rollback capabilities

9. אבטחה ו-Compliance

Security Best Practices 9.1

Network Security 9.1.1

- כל ה compute resources ברשת פרטית: **Private Subnets**
- גישה מאומתת דרך Twingate בלבד: **Zero Trust**

Secrets Security 9.1.2

- AWS Secrets Manager secrets: **No Secrets in Code**
- מדיניות הסיברים: **Rotation**

10. טיפול בעוות נפוצות ו-Troubleshooting

10.1 בעוות נפוצות

ArgoCD Sync Issues 10.1.1

```
# Check ArgoCD application status
kubectl get applications -n argocd

# View detailed application info
argocd app get my-app

# Force refresh
aargocd app refresh my-app

# Manual sync
argocd app sync my-app
```

External Secrets Issues 10.1.2

```
#Check External Secrets status
kubectl get externalsecrets

# View External Secrets events
kubectl describe externalsecret project-secrets

# Validate AWS permissions
aws secretsmanager describe-secret --secret-id project-name
```

11. מסקנות וכיוני פיתוח עתידיים

11.1 הישgi הפרויקט

הפרויקט הצליח להציג את המטרות הבאות:

- אוטומציה מלאה של תהליכי deployment
- אבטחה מתקדמת עם Zero Trust Network
- ניהול secrets מ慷慨י עם AWS Secrets Manager
- ArgoCD workflow
- Terraform עם Infrastructure as Code

11.2 וכיוני פיתוח עתידיים

11.2.1 שיפורים טכנולוגיים

- Grafana-ו Prometheus: הוספת Observability
- multi-region deployment: Disaster Recovery
- OPA Gatekeeper: Policy as Code

11.2.2 שיפורים תהליכיים

- Blue/Green deployments-ו Progressive Delivery: Canary
- testing frameworks עם Automated Testing: Integration
- Compliance Automation: SAST/DAST integration

נומחים

נספח א: קישורים ומקורות

תיעוד רשמי

- [AWS EKS Documentation](#) •
- [Terraform AWS Provider](#) •
- [ArgoCD Documentation](#) •
- [External Secrets Operator](#) •
- [Twingate Documentation](#) •

Best Practices Guides

- [EKS Best Practices Guide](#) •
- [Kubernetes Security Best Practices](#) •
- [GitOps Principles](#) •

נספח ב: סקורייפטים

main.tf

```

provider "aws" {
  region    = var.aws_region
  access_key = var.aws_access_key_id
  secret_key = var.aws_secret_access_key
}

provider "twingate" {
  api_token = var.twingate_api_token
  network   = var.twingate_network
}

provider "kubernetes" {
  host          = data.aws_eks_cluster.this.endpoint
  cluster_ca_certificate = base64decode(data.aws_eks_cluster.this.certificate_authority[0].data)
  token         = data.aws_eks_cluster_auth.this.token
}

data "aws_eks_cluster" "this" {
  name = module.eks.cluster_name
}

data "aws_eks_cluster_auth" "this" {
  name = module.eks.cluster_name
}

terraform {
  required_version = ">=1.5"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 5.16.0"
    }

    twingate = {
      source  = "twingate/twingate"
      version = "~>3.3.2"
    }
  }
}

```

```

# MODULE: Network / VPC
module "network" {
  source          = "../../modules/network"
  vpc_cidr       = var.vpc_cidr
  public_subnet_cidrs = var.public_subnet_cidrs
  private_subnet_cidrs = var.private_subnet_cidrs
  availability_zones = var.availability_zones
  project_prefix   = var.project_prefix
}

# MODULE: EKS Cluster
module "eks" {
  source          = "../../modules/eks"
  vpc_id         = module.network.vpc_id
  subnet_ids     = module.network.private_subnet_ids

  eks_cluster_name = var.eks_cluster_name
  cluster_version  = var.cluster_version

  node_group_desired_size = var.node_group_desired_size
  node_group_max_size     = var.node_group_max_size
  node_group_min_size     = var.node_group_min_size
  node_group_instance_types = var.node_group_instance_types

  environment      = var.environment
  project_name     = var.project_prefix
}

# MODULE: Twingate Connector
module "twingate" {
  source          = "../../modules/twingate"
  project_prefix  = var.project_prefix
  vpc_id         = module.network.vpc_id
  private_subnet_ids = module.network.private_subnet_ids
  public_subnet_ids = module.network.public_subnet_ids
  twingate_network = var.twingate_network
  twingate_remote_network_name = var.twingate_remote_network_name
  twingate_connector_name = var.twingate_connector_name
  twingate_api_token   = var.twingate_api_token
  eks_security_group_id = module.eks.cluster_security_group_id
}

```

```

# IAM role for External Secrets Operator
resource "aws_iam_policy" "eso_READONLY_policy" {
  name      = "${var.project_prefix}-Externals-ReadOnlyPolicy"
  description = "Read-only access to one specific secret for ESO"
  policy = jsonencode({
    Version = "2012-10-17",
    Statement: [
      {
        Sid: "SecretsManagerReadSpecificSecretOnly",
        Effect: "Allow",
        Action: [
          "secretsmanager:GetSecretValue",
          "secretsmanager:DescribeSecret",
          "secretsmanager>ListSecretVersionIds"
        ],
        Resource: "arn:aws:secretsmanager:eu-central-1:314525640319:secret:rickroll-project-menahem-einav-QP2Y08"
      },
      {
        Sid: "KMSReadOnly",
        Effect: "Allow",
        Action: [
          "kms:DescribeKey",
          "kms>ListAliases",
          "kms>ListKeys"
        ],
        Resource: "*"
      }
    ]
  })
}

```

twingate-run-connector.sh

```
#!/bin/bash
set -euxo pipefail

echo "[INFO] Updating and installing Docker..." | tee /var/log/twingate.log

apt-get update | tee -a /var/log/twingate.log
apt-get install -y docker.io | tee -a /var/log/twingate.log

echo "[INFO] Enabling and starting Docker..." | tee -a /var/log/twingate.log
systemctl enable docker
systemctl start docker

echo "[INFO] Waiting for Docker to be ready..." | tee -a /var/log/twingate.log
until docker info > /dev/null 2>&1; do
| sleep 1
done

echo "[INFO] Running Twingate Connector..." | tee -a /var/log/twingate.log

docker run -d \
--sysctl net.ipv4.ping_group_range="0 2147483647" \
--env TWINGATE_NETWORK="${TWINGATE_NETWORK}" \
--env TWINGATE_ACCESS_TOKEN="${TWINGATE_ACCESS_TOKEN}" \
--env TWINGATE_REFRESH_TOKEN="${TWINGATE_REFRESH_TOKEN}" \
--env TWINGATE_LABEL_HOSTNAME="${TWINGATE_LABEL}" \
--env TWINGATE_LABEL_DEPLOYED_BY="docker" \
--name "twingate-pragmatic-griffin" \
--restart=unless-stopped \
--pull=always \
twingate/connector:1
```

AWS Lambda:

סקריפט זה הינו פונקציית AWS Lambda שモפעלת ע"י Scheduler ומיועדת להפעיל או לעצור מашבים בסביבת AWS. היא מזזה את מופע ה-EC2 עם תג מסוים ומפעילה או עוצרת אותם בהתאם למצב (mode), וגם מעודכנת את הגדרות ה-Scaling של קבוצות ה-Node ב-EKS כדי להגדיל את המশאים באופן אוטומטי.

```

import boto3
import os

REGION = os.environ.get("AWS_REGION", "eu-central-1")
MODE = os.environ.get("MODE", "stop").lower()

DEFAULT_CLUSTER = "menahem-einav_cluster"

EKS_MIN_SIZE = int(os.environ.get("EKS_MIN_SIZE", "1"))
EKS_MAX_SIZE = int(os.environ.get("EKS_MAX_SIZE", "2"))
EKS_DESIRED_SIZE = int(os.environ.get("EKS_DESIRED_SIZE", "1"))

ec2 = boto3.client('ec2', region_name=REGION)
eks = boto3.client('eks', region_name=REGION)

def lambda_handler(event, context):
    mode = event.get('mode') or MODE
    print(f"Running in '{mode}' mode...")

    ec2_state_filter = 'running' if mode == 'stop' else 'stopped'
    instances = ec2.describe_instances(Filters=[
        {'Name': 'tag:Name', 'Values': ['menahem-einav-twingate-ec2']},
        {'Name': 'instance-state-name', 'Values': [ec2_state_filter]}
    ])
    instance_ids = [i['InstanceId'] for r in instances['Reservations'] for i in r['Instances']]

    if instance_ids:
        if mode == 'stop':
            print(f"Stopping EC2 instances: {instance_ids}")
            ec2.stop_instances(InstanceIds=instance_ids)
        else:
            print(f"Starting EC2 instances: {instance_ids}")
            ec2.start_instances(InstanceIds=instance_ids)
    else:
        print(f"No EC2 instances to {mode}.")

    clusters_event = event.get("clusters")
    if isinstance(clusters_event, list) and clusters_event:
        cluster_list = clusters_event
    else:
        clusters_env = os.environ.get("CLUSTERS", DEFAULT_CLUSTER)
        cluster_list = [c.strip() for c in clusters_env.split(",") if c.strip()]

    for cluster in cluster_list:
        try:
            nodegroups = eks.list_nodegroups(clusterName=cluster)['nodegroups']
            for ng in nodegroups:
                config = {'minSize': 0, 'maxSize': 1, 'desiredSize': 0} if mode == 'stop' else {
                    'minSize': EKS_MIN_SIZE,
                    'maxSize': EKS_MAX_SIZE,
                    'desiredSize': EKS_DESIRED_SIZE,
                }
                print(f"Updating EKS node group: {cluster}/{ng} -> {config}")
                eks.update_nodegroup_config(
                    clusterName=cluster,
                    nodegroupName=ng,
                    scalingConfig=config
                )
        except Exception as e:
            print(f"Failed to update cluster {cluster}: {e}")

    return {'status': f'{mode} completed'}

```

aws-key-rotation:

תוכנית זאת נועדה לאתר את כל משתמשי IAM ב-AWS, לבדוק את גיל מפתחות הגישה שלהם, ולזהות מפתחות ישנים מעל מגבלות הימים שהוגדרה.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include "strlist.h"

#define DAYS_LIMIT 90

StrList get_all_usernames() {
    StrList usernames = StrList_init(8);

    FILE *fp = popen("aws iam list-users --query 'Users[*].UserName' --output text", "r");
    if (!fp) {
        perror("popen list-users");
        return usernames; // empty list on error
    }
    char *line = NULL;
    size_t len = 0;
    ssize_t read;

    // Using POSIX getline to read lines (allocates buffer automatically)
    while ((read = getline(&line, &len, fp)) != -1) {
        // Tokenize line by tab '\t' and newline '\n'
        char *token = strtok(line, "\t\n");
        while (token) {
            StrList_add(&usernames, token);
            token = strtok(NULL, "\t\n");
        }
    }
    free(line);
    pclose(fp);

    return usernames;
}

int key_age_days(const char* create_date_iso) {
    char cmd[256], output[64];
    sprintf(cmd, sizeof(cmd), "echo $(( $(date +%%s) - $(date -d '%s' +%%s)) / 86400 ))", create_date_iso);

    FILE* fp = popen(cmd, "r");
    if (!fp) return -1;
    if (fgets(output, sizeof(output), fp) == NULL) {
        pclose(fp);
        return -1;
    }
    pclose(fp);

    return atoi(output);
}

StrList get_old_keys(const char* username) {
    char cmd[512];
    sprintf(cmd, sizeof(cmd), "aws iam list-access-keys --user-name %s --query 'AccessKeyMetadata[*].[AccessKeyId,CreateDate]' --output text", username);

    FILE* fp = popen(cmd, "r");
    StrList old_keys = StrList_init(4);
    if (!fp) {
        perror("Failed to list access keys");
        return old_keys;
    }
    char *line = NULL;
    size_t len = 0;
    ssize_t read;

    while ((read = getline(&line, &len, fp)) != -1) {
        char access_key[64], create_date[64];
        if (sscanf(line, "%63s %63s", access_key, create_date) == 2) {
            int age = key_age_days(create_date);
            if (age >= DAYS_LIMIT) {
                StrList_add(&old_keys, access_key);
            }
        }
    }
    free(line);
    pclose(fp);
    return old_keys;
}
```

```

int create_new_key(const char* username, char* new_access_key, char* new_secret_key) {
    char cmd[512];
    snprintf(cmd, sizeof(cmd), "aws iam create-access-key --user-name %s --query 'AccessKeyId,SecretAccessKey' --output text", username);

    FILE* fp = fopen(cmd, "r");
    if (!fp) {
        perror("Failed to create new access key");
        return -1;
    }

    if (fscanf(fp, "%63s %127s", new_access_key, new_secret_key) != 2) {
        fprintf(stderr, "Failed to parse new access key\n");
        pclose(fp);
        return -1;
    }

    pclose(fp);
    return 0;
}

void deactivate_keys(const char* username, StrList* keys) {
    char cmd[512];
    for (int i = 0; i < keys->length; i++) {
        const char* key = keys->items[i];
        snprintf(cmd, sizeof(cmd), "aws iam update-access-key --user-name %s --access-key-id %s --status Inactive", username, key);
        int rc = system(cmd);
        if (rc == 0) {
            printf("Deactivated old key %s\n", key);
        } else {
            fprintf(stderr, "Failed to deactivate key %s\n", key);
        }
    }
}

void rotate_keys_for_user_test(const char* username) {
    StrList old_keys = get_old_keys(username);

    if (old_keys.length == 0) {
        printf("[TEST] No keys older than %d days for user %s\n", DAYS_LIMIT, username);
        StrList_free(&old_keys);
        return;
    }

    printf("[TEST] Would create new access key for user %s\n", username);
    printf("[TEST] Would deactivate keys:\n");
    for (int i = 0; i < old_keys.length; i++) {
        printf(" %s\n", old_keys.items[i]);
    }

    StrList_free(&old_keys);
}

```

במצב test היא מזיגה אילו פעולות הio מתבצעות — יצירת מפתח חדש והשבחת המפתחות הקיימים — תוך שימוש ב-AWS CLI להרצת הפקודות.

```

int main() {
    StrList users = get_all_usernames();

    printf("Found %d user(s):\n", users.length);
    for (int i = 0; i < users.length; ++i) {
        printf(" - %s\n", users.items[i]);
        rotate_keys_for_user_test(users.items[i]);
    }

    StrList_free(&users);
    return 0;
}

```

בתוכנית C זאת, השתמשתי ב `#include "strlist.h"` זה ה header file .

```

#ifndef STRLIST_H
#define STRLIST_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char** items;
    int length;
    int capacity;
} StrList;

StrList StrList_init(int capacity) {
    StrList list;
    list.items = (char**)malloc(capacity * sizeof(char*));
    list.length = 0;
    list.capacity = capacity;
    return list;
}

void StrList_free(StrList* list) {
    for (int i = 0; i < list->length; i++) {
        free(list->items[i]);
    }
    free(list->items);
    list->items = NULL;
    list->length = 0;
    list->capacity = 0;
}

void StrList_add(StrList* list, const char* item) {
    if (list->length == list->capacity) {
        list->capacity = list->capacity > 0 ? list->capacity * 2 : 4;
        char** new_items = (char**)realloc(list->items, list->capacity * sizeof(char*));
        if (!new_items) {
            perror("realloc");
            exit(EXIT_FAILURE);
        }
        list->items = new_items;
    }
    list->items[list->length++] = strdup(item);
}

char* StrList_get(StrList* list, int index) {
    if (index < 0 || index >= list->length) {
        return NULL;
    }
    return list->items[index];
}

void StrList_print(StrList* list) {
    for (int i = 0; i < list->length; ++i) {
        printf("%d: %s\n", i, list->items[i]);
    }
}

#endif // STRLIST_H

```

12. סיכום

12.1 סיכום הפרויקט

פרויקט זה הציגIMPLEMENTATION מקיף של פתרון cloud-native מודרני הכלול:

הישגים טכנולוגיים:

- אוטומציה מלאה של תהליכי CI/CD עם GitOps
- יישום Zero Trust Network לאבטחה מתקדמת
- ניהול secrets מאובטח עם AWS Secrets Manager
- תשתיות מודולרית וscalable עם Terraform
- Kubernetes/EKS עם Container orchestration

יתרונות עסקיים:

- זמן deployment מהיר ואמין
- אבטחה ברמה enterprise
- חיסכון בעליות תפעול
- מדרגות גבואה
- אמינות ושמירת SLA

12.2 רפלקציה על התהילה

הפרויקט חשף אתגרים מעוניינים בתחוםים הבאים:

- הצורך באבטחה רב-שכבותית Secrets Management
- יישום Zero Trust בסביבת cloud Network Security
- איזון בין גמישות לבטיחות Automation

12.3 המלצות להמשך

לטוווח קצר:

- יישום automated testing בpipelines
- הוספת alerting- monitoring- מתקדמים
- שיפור runbooks- documentation

לטוווח ארוך:

- מעבר ל multi-region deployment
- יישום service mesh Istio/Linkerd
- הוספת advanced security scanning
- יישום chaos engineering

מקורות ביבליוגרפיה

- Amazon Web Services.** (2024). *Amazon EKS User Guide*. .1
/https://docs.aws.amazon.com/eks
- HashiCorp.** (2024). *Terraform AWS Provider Documentation*. .2
/https://registry.terraform.io/providers/hashicorp/aws
- Argo Project.** (2024). *ArgoCD Documentation*. https://argo-cd.readthedocs.io .3
- /External Secrets Operator.** (2024). *ESO Documentation*. https://external-secrets.io .4
- Twingate.** (2024). *Zero Trust Network Access Documentation*. .5
/https://docs.twingate.com
- Cloud Native Computing Foundation.** (2024). *Kubernetes Documentation*. .6
/https://kubernetes.io/docs
- /Helm.** (2024). *Helm Documentation*. https://helm.sh/docs .7
- GitHub.** (2024). *GitHub Actions Documentation*. https://docs.github.com/en/actions .8
- /Docker Inc.** (2024). *Docker Documentation*. https://docs.docker.com .9
- AWS.** (2024). *AWS EKS Best Practices Guide*. .10
/https://aws.github.io/aws-eks-best-practices

הפרויקט הוקם במסגרת קורס פרקטיקום בהנדסת תוכנה - התמחות בטכנולוגיות ענן ו-DevOps

מצגת הפרויקט

פרויקטיקום

טכני תובנה-DevOps

מנחם שפרבר



תיאור החברה ולקוחותיה

בינת תקשורת מחשבים הינה חברת מובילה באספקת פתרונות IT, תקשורת, ענן וابטחת מידע למגזרים הציבורי, הביטחוני, והעסקי בישראל. חטיבת DevOps בינת אחראית על תכנון והטמעה של תהליכי אוטומציה, ניהול תשתיות ענן ו-IT מודרני, ושיפור רציפות הפיתוח והפרישה עברור מגוון רחב של לקוחות.

לבינה צוות מקצועי וдинמי, הפועל בסביבה טכנולוגית מתקדמת ומשלב עבודה עם כלים וmethodologies עדכניות מעולמות ה-DevOps, Cloud, Agile.

רקע לפרויקט ודרישות הליקוח

הליקוח פנה לחברת ביינט במטרה לבצע שדרוג מהותי לאבטחת הגישה לתשתיות הארגוניות, וזאת לאור סיכון אבטחה שהתגלו בשימוש בפתרונות ה-VPN הקיים. הדרישת המרכזית הייתה להטמעת פתרון Zero Trust Network (ZTN) אשר יאפשר בקרת גישה קפדרנית ומובוססת זהות, לצד צמצום מרבי של שטח החשיפה.

בנוסף, הליקוח הציביע עד להקמת סביבת Kubernetes בענן AWS, שתספק יכולות ניהול מתקדמות, גמישות תפעולית, ואוטומציה מלאה של תהליכי פרישה ותחזוקה של אפליקציות ותשתיות. הפרויקט כלל אלינטגרציה של מגנוני ניהול סודות מאובטחים, לצד יישום מתודולוגיית GitOps, על מנת להבטיח פרישה וניהול גרסאות בצורה עקבית, מאובטחת ויעילה.

шиיקולי תכנון

מודולריות – בניית המערבתvr בך שביל רביב פועל בנפרד, מה שמקל על תחזוקה ושדרוגים.

אבטחה מתקדמת – שילוב מגנוני ההגנה חמימים למניעת גישה לא מורשית ואיומי סייבר.

אוטומציה מלאה – צמצום עבודה ידנית באמצעות תהליכי פרישה וניהול אוטומטיים.

יכולת התרחבות עתידית – תכנון המערבתvr בך שתוכל לגודל ולהתאים לצרכים חדשים בקצב.



```
# MODULE: Network / VPC
module "network" {
  source          = "../../modules/network"
  vpc_cidr        = var.vpc_cidr
  public_subnet_cids = var.public_subnet_cids
  private_subnet_cids = var.private_subnet_cids
  availability_zones = var.availability_zones
  project_prefix   = var.project_prefix
}
```

Kubernetes (EKS) Setup
,Terraform באמצעות EKS בBootstrap
הגדרת גישה לקלאסטר עם **kubectl**
הקמת **Node Groups** ו奧托索キילינג



```
# MODULE: EKS Cluster
module "eks" {
  source          = "../../modules/eks"
  vpc_id          = module.network.vpc_id
  subnet_ids      = module.network.private_subnet_ids

  eks_cluster_name = var.eks_cluster_name
  cluster_version = var.cluster_version

  node_group_desired_size = var.node_group_desired_size
  node_group_max_size     = var.node_group_max_size
  node_group_min_size     = var.node_group_min_size
  node_group_instance_types = var.node_group_instance_types

  environment
  project_name        = var.environment
  prefix               = var.project_prefix
}
```

AWS Environment Setup
הגדרת ארכיטקטורת רשת: **VPC**, תתי-רשת,
בתיבת מודולית Terraform להקמה אוטומטית
הקמת **Security Groups** והגדרת EKS Cluster



Zero Trust Networking (Twingate)
הטמעת ZTN בפתרון Twingate
הפעלת Connector בתוך-VPC
קביעת מדיניות גישה מאובטחת



```
# MODULE: Twingate Connector
module "twingate" {
  source          = "../../modules/twingate"
  project_prefix   = var.project_prefix
  vpc_id          = module.network.vpc_id
  private_subnet_ids = module.network.private_subnet_ids
  public_subnet_ids = module.network.public_subnet_ids
  turing_gateway_ip = var.turing_gateway_ip
  twingate_remote_network_name = var.twingate_remote_network_name
  twingate_connector_name = var.twingate_connector_name
  twingate_api_token = var.twingate_api_token
  eks_security_group_id = module.eks.cluster.security_group_id
}
```



GitOps with Argo CD

התקנת Argo CD על הקלאסטר

Argo Applications

הגדרת Source of Truth ב-Argo CD

```
# git checkout -b devops-deploy-pilot
name: Build and Deploy to GKE
apiVersion: argoproj.io/v1alpha1
kind: Application
spec:
  route:
    path: "/api"
    target:
      type: Deployment
      name: hello-world
  replicas: 1
  triggers:
    - type: Git
      workflowDispatch:
        ref: main
        annotations:
          permissions:
            contents: write
        jobs:
          - build-and-deploy
            run: echo "Hello world from Argo CD"
            steps:
              - name: Checkout code
                uses: actions/checkout@v2
              - name: Set environment variables
                uses: actions/configure-prereqs@v1
                with:
                  secrets:
                    DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
                    DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}
              - name: Build and push Docker image
                image: gcr.io/distroless/base
                args:
                  - --tag
                  - ${{ env.GITHUB_REF }}-${{ env.GITHUB_SHA }}-values.yaml
                env:
                  DOCKER_REGISTRY: gcr.io
                  DOCKER_IMAGE_NAME: ${{ env.GITHUB_REF }}-${{ env.GITHUB_SHA }}-values.yaml
                  DOCKER_TAG: ${{ env.GITHUB_REF }}-${{ env.GITHUB_SHA }}-values.yaml
              - name: Push image tag to values.yaml
                run: |
                  sed -i 's/\${{ secrets.GITHUB_TOKEN }}/\${{ secrets.GITHUB_TOKEN }}/' ./values.yaml
  secrets:
    - name: secrets
      provider: GitHub
      token:
        type: OAuthToken
        value: ${{ secrets.GITHUB_TOKEN }}
```

כלים וטכנולוגיות




Secrets Management

AWS Secrets Manager: בהירט מנהל סודות: הדרמת סודות לאפליקציה בצוות אוטומטי

הגדרת מדיניות ריענון סודות

```
# external-secrets/secret-store.yaml
apiVersion: external-secrets.io/v1
kind: ClusterSecretStore
metadata:
  name: argo-cluster-secret-store
spec:
  provider:
    aws:
      service: SecretsManager
      region: eu-central-1
      auth:
        secretRef:
          accessKeyId: DSecretRef:
            name: access-secret
            key: access-key
            namespace: external-secrets
          secretAccessKeyRef:
            name: access-secret
            key: secret-access-key
            namespace: external-secrets
  annotations:
    external-beta.kubernetes.io/mw-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
    service.beta.kubernetes.io/mw-load-balancer-tags: "Environment=dev,cluster=demo-clus"
```

Ingress and Load Balancer

AWS ALB Ingress in NGINX או NGINX Controller

התקנת NGINX Controller והיפוך שירותים עם DNS, עדכון Route 53-בDNS

```
# values-ingress-controller.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: values-ingress-controller
  namespace: kube-system
  labels:
    app: values-ingress-controller
    chart: values-ingress-controller-0.1.0
    release: values-ingress-controller
    component: controller
  annotations:
    configmap.kubernetes.io/immutable: "true"
  progressDeadlineSeconds: 0
  ingressClassResource:
    name: values-ingress-controller
    enabled: true
    default: true
    service:
      type: LoadBalancer
      annotations:
        external-beta.kubernetes.io/mw-load-balancer-type: "nlb"
        service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
        service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
        service.beta.kubernetes.io/mw-load-balancer-tags: "Environment=dev,cluster=demo-clus"
```

This is how to add Ingress rule

```
# helm repo add ingress-inginx https://kubernetes.github.io/ingress-inginx
# helm upgrade --install ingress-inginx ingress-inginx/ingress-inginx

# kubectl get svc
# kubectl patch svc ingress-inginx -p '{"spec": {"externalTrafficPolicy": "Cluster", "loadBalancerIP": "192.168.1.100", "ports": [{"port": 80, "targetPort": 80}], "type": "LoadBalancer"} }'
# kubectl get svc
# kubectl get svc --selector=app=values-ingress-controller
# kubectl get svc --selector=app=values-ingress-controller -n kube-system
# kubectl get svc --selector=app=values-ingress-controller -n kube-system -n kube-system
```



Application Deployment

שימוש ב-Helm לייצור Templates, יצירה קבצי מנופס:

Deployment, Service, Ingress

```
# Chart.yaml
apiVersion: v2
name: hello-counter
description: A Helm chart for Kubernetes test app
type: application
version: 0.1.0
appVersion: "1.23.0"
```

Automation & Documentation

שמירת קוד Git-ב-Helm-ו-Terraform CI/CD אוטומציה של הפרישה באמצעות Automation & Documentation

GitHub

```
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.deployment.name }}
  namespace: {{ .values.namespace }}
  labels:
    app: hello-selector
spec:
  replicas: {{ .Values.deployment.replicas }}
  selector:
    matchLabels:
      app: hello-selector
  template:
    metadata:
      labels:
        app: hello-selector
    spec:
      containers:
        - name: {{ .Values.deployment.container.name }}
          image: {{ .Values.deployment.container.image }}
          restartPolicy: Always
          ports:
            - containerPort: {{ .Values.deployment.container.port }}
          envFrom:
            - secretRef:
                name: {{ .Values.deployment.container.envFromSecretName }}
          volumeMounts:
            - name: secrets-store-inline
              mountPath: /mnt/secrets-store
              readOnly: true
          volumes:
            - name: secrets-store-inline
              ebs:
                driver: secrets-store.csi.k8s.io
                readOnly: true
                volumeAttributes:
                  secretProviderClass: aws-secret
```

```
# values.yaml
namespace: app
deployment:
  name: hello-counter
  replicas: {{ .Values.deployment.replicas }}
  containers:
    name: counter
    image: mysql:5.7
    port: 40
    environment:
      database: visitor_counter
      service: service-hello-counter
      port: 3306
  ingress:
    name: ingress-hello
    ingressClassName: nginx
    path: /
    pathType: Prefix
  mysql:
    enabled: true
    name: mysql
    image: mysql:8.0
    port: 3306
    rootPassword: rootpassword
    database: visitor_counter
    volume:
      type: persistentVolumeClaim
      claimName: mysql-pvc
      hostPath: /var/lib/mysql
  initdb:
    enabled: true
    script:
      CREATE DATABASE IF NOT EXISTS visitor_counter;
      USE visitor_counter;
      CREATE TABLE IF NOT EXISTS users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        username VARCHAR(50) UNIQUE NOT NULL,
        password VARCHAR(255) NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      );
```

יחסים גומליין עיקריים בין מרכיבי המערכת

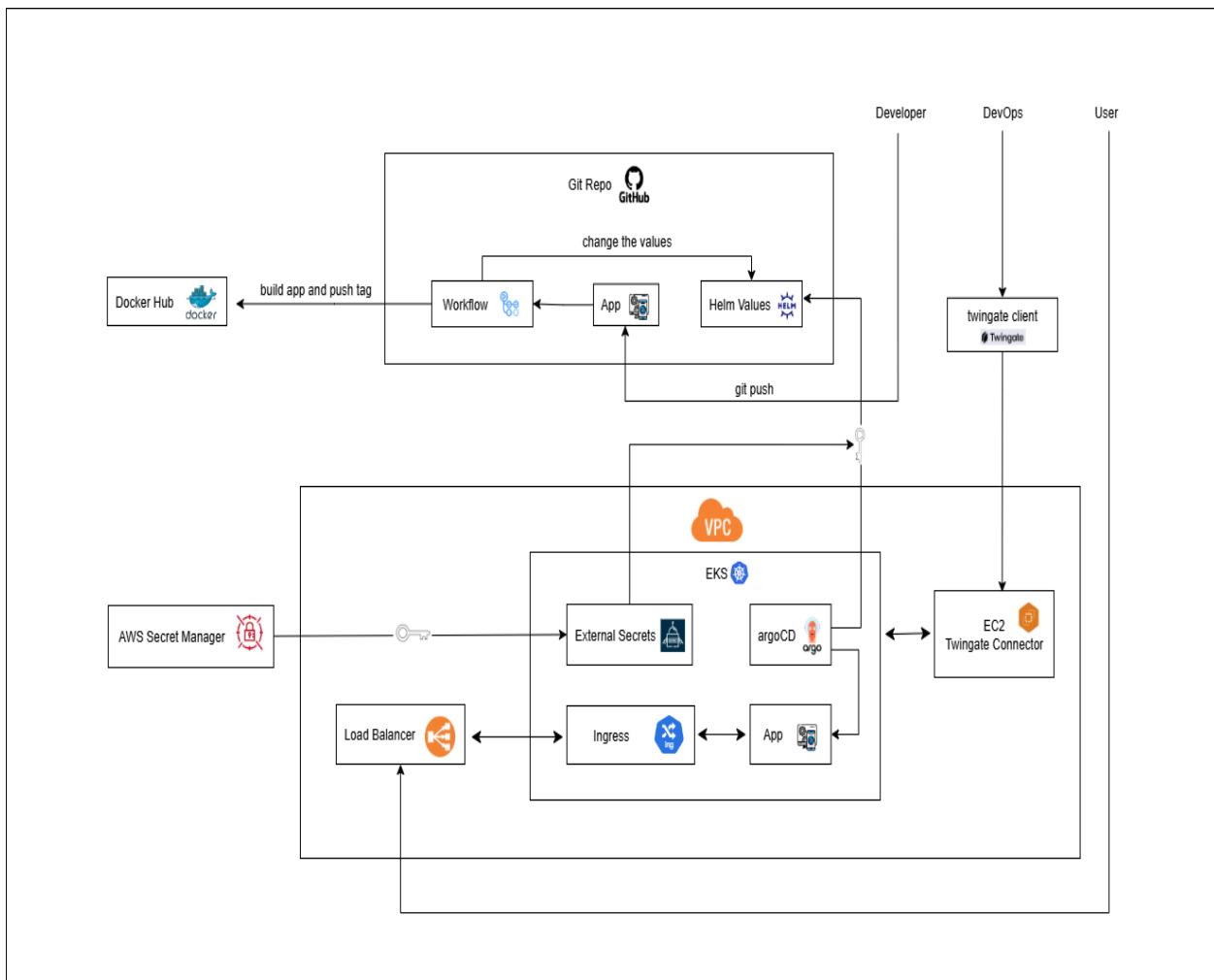
אפשר גישה מבוקרת רק למורשים ומעביר תובורה מוצפנת ל-EKS דרך KonNECTOR בתוך ה-VPC.

אחראי על כל הקמה תשתיתית – Terraform VPC, Subnets, Security Groups, EKS Argo CD מושך את הגדרות הפרישה מתיק Git וمعدכן את הקלאוסטר אוטומטית.

Helm משמש לפרישת אפליקציות בגרפים מודולריים – כולל שירותים, Secrets, Ingress וכו'.

AWS Secrets Manager מספק סודות בצורה מאובטחת לקלאסטר דרך שירות RSA.

ארכיטקטורה ותהליכי זרימה



סיכום מרכבי הפרויקט

במסגרת הפרויקט נבנתה תשתיית ענן מתקדמת ב-AWS באמצעות Terraform, שבכללה ייצור VPC תתי-רשתות, קבוצות אבטחה ו-EKS לניהול קוברנטייס. הושם פתרון Zero Trust Network באמצעות Twingate, על מנת לספק גישה מאובטחת ומובוססת זהות לכל הרכיבים. בקורס ריבבי ניהול מרכזים כמו Load Balancer-ו-Ingress Controller, לצד פתרונות לניהול סודות בצורה מאובטחת באמצעות AWS Secrets Manager או External Secrets Operator. הalice הפירוש והעדרון התבצעו בגישה CD עם Argo, Helm לשילוב管理 תבניות האפליקציות, וכל זאת מתחת למערך אוטומציה המאפשר שליטה, ניטור והתרחבות עתידית בצורה פשוטה ויעילה.



כיווני פיתוח עתידיים

לטוווח קצר

Grafana-ו-Prometheus עם Observability
הטמעת CI/CD Pipeline ב-Automated Testing
שדרוג-ו-Monitoring Alerting
Runbooks-ו-Documentation
шиיפור Policy as Code (OPA Gatekeeper)
יישום (OPA Gatekeeper)
Progressive Delivery (Canary / Blue-Green)

לטוווח ארוך

Disaster Recovery Multi-Region Deployment
הטמעת Service Mesh (Istio / Linkerd)
(Advanced Security Scanning)
יישום Chaos Engineering לשיפור שרידות