

## תוכן עניינים

2.....	הצעת הפרויקט
5.....	טופס הצהרת
6.....	אישור מקום ההתנסות
7.....	רקע/מבוא - הכרת הארגון וסביבת העבודה
8.....	סיכומי דו"חות ופגישות
8.....	דוח שבועי 1
9.....	דוח שבועי 2
10.....	דוח שבועי 3
11.....	דוח שבועי 4
12.....	הצגת המשימות
12.....	1. מבוא ורקע לפרויקט
13.....	2. תכנון וארכיטקטורת המערכת
13.....	3. שלב 1-3: בניית התשתית הבסיסית
24.....	4. שלבים 4-5: Load Balancing ו- Application Layer
27.....	5. שלב 6: Secret Managment
29.....	6. שלבים 7-8: GitOps ו- CI/CD Automation
33.....	7. תהליך הפריסה המלא
33.....	8. ניתור ו- Observability
34.....	9. אבטחה ו- Compliance
34.....	10. טיפול בבעיות נפוצות ו- Troubleshooting
35.....	11. מסקנות וכיווני פיתוח עתידיים
36.....	נספחים
43.....	12. סיכום
44.....	מקורות ביבליוגרפיה
45.....	מצגת הפרויקט

## רקע/מבוא - הכרת הארגון וסביבת העבודה

---

**בינת תקשורת מחשבים** הינה חברה מובילה במתן פתרונות מתקדמים בתחומי ה-IT, התקשורת, הענן ואבטחת המידע, ופועלת מול מגוון רחב של לקוחות במגזרים הציבורי, הביטחוני והעסקי בישראל. החברה מספקת שירותים מקצה לקצה – החל משלב התכנון והאפיון, דרך ההקמה וההטמעה, ועד לניהול ותחזוקה שוטפת של מערכות ותשתיות. צוותי החברה מתמחים ביישום טכנולוגיות מתקדמות ובהתאמת פתרונות ייעודיים לדרישות הלקוח, תוך שמירה על סטנדרטים גבוהים של ביצועים ואבטחה.

חטיבת ה-**DevOps** בבינת אחראית על פיתוח והטמעת תהליכי אוטומציה, ניהול תשתיות ענן מודרניות ושיפור מתודולוגיות פיתוח ופריסה בארגונים. החטיבה פועלת בסביבה טכנולוגית עדכנית, תוך שימוש בכלים ובמתודולוגיות מעולמות ה-**Cloud**, **DevOps** ו-**Agile**, ומיישמת פתרונות המבוססים על Kubernetes, ניהול תצורה, תשתיות כקוד (IaC) וכלי CI/CD מתקדמים.

הפרויקט הנוכחי תוכנן במטרה לספק ללקוח פתרון מלא ומקיף העונה על צרכיו הארגוניים והטכנולוגיים. הדרישה המרכזית הייתה הטמעת ארכיטקטורת **Zero Trust Network**, המאפשרת בקרת גישה קפדנית המבוססת על זהות, מצב המכשיר והקשר הגישה, תוך צמצום מרבי של שטח החשיפה ומניעת גישה בלתי מורשית למשאבים רגישים. הפתרון כלל מנגנוני ניהול הרשאות מתקדמים והגנה רב-שכבתית.

בנוסף, הוגדרה דרישה להקמת סביבת **Kubernetes** בענן **AWS**, שתאפשר גמישות תפעולית, ניהול משאבים דינמי ויכולות אוטומציה מלאות לפריסת אפליקציות ותשתיות. סביבת ה-Kubernetes נבנתה עם מנגנון ניהול סודות מאובטח (Secret Management), תמיכה 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 ההפצות. הפתרון נבנה כך שיספק זמינות גבוהה (High Availability), יתמוך בדרישות תאימות רגולטוריות ויאפשר תחזוקה ותפעול יעילים לאורך זמן, עם יכולת הרחבה עתידית בהתאם לצרכי הארגון.

# סיכומי דו"חות ופגישות

## דוח שבועי 1 – הכנת תשתית AWS ובסיס ZTN

מטרות: להקים את התשתית בענן ולהתחיל בהגדרת חיבורי רשת מאובטחים.

תכולה:

### 1. AWS Environment Setup

○ הגדרת ארכיטקטורת תשתית: VPC, Subnets, Routing Tables, Internet Gateway, NAT Gateway.

○ כתיבת מודולים ב-Terraform לפריסה של:

■ VPC ותתי-רשתות

■ Routing ו-Security Groups בסיסיים

■ בסיס להקמת EKS Cluster

### 2. Secure Connectivity via Zero Trust Network

○ בחירת ספק Twingate והבנת הדרישות.

○ פריסת ZTN Connector ב-VPC.

○ קונפיגורציה ראשונית של מדיניות גישה מאובטחת (Access Policies).

### 3. תיעוד שבועי

○ תרשימי רשת (VPC/Subnets/Connector)

○ קבצי Terraform בסיסיים

○ דוגמה למדיניות ZTN בסיסית

תוצרים:

● Terraform modules בסיסיים לפריסת VPC ו-EKS Skeleton

● חיבור ZTN ראשוני לעבודה

## דוח שבועי 2 – פריסת EKS וקונפיגורציה בסיסית

**מטרות:** להקים ולהגדיר את EKS עם ניהול Nodes ותשתית בסיסית ל-Ingress.

**תכולה:**

### 1. Kubernetes (EKS) Configuration

- Bootstrap ל-EKS cluster באמצעות Terraform
- הגדרת AWS IAM, kubectl (Cluster Access)
- קונפיגורציה של Node Groups ו-Autoscaling

### 2. Ingress and Load Balancer

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

### 3. תיעוד שבועי

- תרשים Cluster עם Node Groups
- דוגמה לקובץ kubectl config
- תרשים Ingress + Load Balancer

**תוצרים:**

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

## דוח שבועי 3 – פריסת אפליקציות וניהול סודות

**מטרות:** לפרוס אפליקציות בצורה מאובטחת ולנהל סודות בצורה אוטומטית.

**תכולה:**

### 1. Application Deployments

- תבניות Helm עבור האפליקציות
- קונפיגורציה של Deployment, Service, Ingress

### 2. Secrets Management

- בחירת כלי לניהול סודות (AWS Secrets Manager או External Secrets Operator)
- אוטומציה של הזרקת סודות ל-Kubernetes
- יישום מדיניות סיבוב סודות (Rotation Policy)

### 3. תיעוד שבועי

- דוגמאות Helm Charts
- תרשים זרימת סודות
- מדיניות סיבוב והזרקה

**תוצרים:**

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

## דוח שבועי 4 – GitOps, אוטומציה ותיעוד סופי

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

**תכולה:**

### 1. GitOps with Argo CD

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

### 2. Automation & Documentation

- אחסון Terraform ו-Helm בקוד Git version-controlled
- אוטומציה של פריסת תשתית ואפליקציות דרך CI/CD
- תיעוד סופי:

- Infrastructure design
- Access instructions
- Secrets strategy
- Argo CD app structure

### 3. תוצרים סופיים

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

## הצגת 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 והפריסה. פרויקט זה מתמקד בבניית תשתית ענן מתקדמת ומאובטחת על גבי AWS, תוך שילוב של טכנולוגיות מתקדמות כגון GitOps, Zero Trust Network, Kubernetes, ו-Infrastructure as Code.

המטרה העיקרית היא יצירת סביבת פיתוח ופרודקציה מאובטחת ואוטומטית שמאפשרת לצוותי הפיתוח לפרוס ולנהל אפליקציות בצורה יעילה ובטוחה.

#### 1.2 מטרות הפרויקט

- **אבטחה מתקדמת:** יישום עקרונות Zero Trust Network לגישה מאובטחת למשאבים
- **אוטומציה מלאה:** הקמת pipeline CI/CD מלא עם GitOps
- **ניהול סודות מתקדם:** יישום Secrets Management עם AWS Secrets Manager
- **קונטיינריזציה:** פריסת אפליקציות באמצעות Kubernetes
- **Infrastructure as Code:** ניהול התשתית 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 וארכיטקטורת המערכת

### 2.1 רכיבי התשתית העיקריים

#### 2.1.1 רשת ואבטחה

- **VPC**: רשת פרטית מבודדת עם subnets מרובים
- **Security Groups**: חוקי firewall ברמת הרשת
- **Zero Trust Network - Twingate**: גישה מאובטחת למשאבים פרטיים
- **NAT Gateway**: גישת אינטרנט יוצאת לרכיבים פרטיים

#### 2.1.2 Container Orchestration-ו Compute

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

#### 2.1.3 רכיבי פריסה ואוטומציה

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

## 3. שלב 1-3: בניית התשתית הבסיסית

### 3.1 הקמת סביבת AWS (משימה 1)

#### 3.1.1 ארכיטקטורת הרשת

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

#### VPC ו-Networking:

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



**:Security Groups**

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

**דוגמת קוד Terraform:**

```
#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_irs        = 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 למשאבים הנדרשים להם
- **הצפנה מקצה לקצה:** כל התעבורה מוצפנת

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

### 3.2.2 הקמת Twingate Connector

Twingate Connector מותקן על EC2 instance ב-private subnet ומאפשר גישה מאובטחת למשאבים הפנימיים:

#### תהליך ההתקנה:

1. יצירת EC2 instance ב-private subnet
2. התקנת Twingate Connector באמצעות Docker
3. רישום ה-Connector ב-Twingate Management Console
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 `twingate-run-connector.sh` הוא סקריפט שתחול (user data script) שמריץ ב-EC2 את תהליך ההתקנה וההרצה של Twingate Connector בתוך Docker container, באמצעות פרטי הגישה (Access/Refresh Tokens) ש-Twingate API קיבל מה-Terraform. ראה בנוסף ב.



### 3.3 הקדמה למשימות 4 ו 5

משימות 4 ו 5 זה למעשה האפליקציה שהמשתמשים יכולים לגשת אליה. בשביל שהמשתמש יוכל לדבר עם האפליקציה או בונים ingress ו Load Balancer ב kubernetes cluster. המשתמש מתקשר עם ה LB וה LB מתקשר עם האפליקציה שבתוך ה cluster דרך ה ingress. האפליקציה נשמרת ב docker container. את ה docker image או שומרים ב docker hub. את ה deployment של האפליקציה או עושים עם helm chart. את ה docker tag או מעדכנים דרך ה helm values.

הקוד של האפליקציה (וגם ה Helm charts manifests) נשמר ב private github repo.

- כעת נניח שצוות הפיתוח רוצים לדחוף שינוי באפליקציה. אם נרצה שהשינוי יתבצע ב cluster, נצטרך לעשות את הפעולות הבאות.
1. לבנות את האפליקציה. זה אומר לבנות docker image tag חדש (ופעולות נוספות בהתאם לאפליקציה).
  2. לדחוף את התג החדש ל docker hub.
  3. לעדכן את ה helm chart values (ב github repo) להשתמש בתג החדש.
  4. לקחת את ה values המעודכן מתוך github ולעשות עם זה upgrade ל app helm ב kubernetes cluster.
- זהו! העדכון הושלם בהצלחה.

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

אנו משתמשים ב github actions לצורך הבא; כאשר צוות הפיתוח עשו git push לאפליקציה אז ה workflow יבנה את האפליקציה, יבנה את ה docker image עם תג חדש (מוגדר לפי תאריך שעה ושניה), ידחוף ל docker hub עם התג החדש (זה נעשה באמצעות שמירת token ב github secrets), יעדכן את הקובץ helm/app/values.yaml עם התג החדש.

- זהו, פעולות 1 ו 2 ו 3 הושלמו.
- כעת נשאר לעדכן את ה cluster, גם באוטומציה כמובן. לצורך כך נשתמש ב argocd.
- אני יזכיר שכל המניפסט יושב על private repo. אז בשביל ש argo יוכל למשוך את ה helm/app/values.yaml הוא צריך להשתמש ב token.
- את ה token נשמור ב kubernetes secret.
- אם נעשה את זה בצורה ישירה, נוכל לעשות זאת בשני דרכים;
1. ישירות. לדוגמא עם kubectl create secret
  2. לעשות secret.yaml, לשמור את ה token שם ואז לעשות לקובץ apply.
- הדרך הראשונה לא מספיק טובה לנו, כי נרצה לשמור את ה token (וכן את שאר הסודות) בצורה מסודרת. הדרך השנייה גם לא מספיק טובה, מכיוון שאנו שומרים את ה token בקובץ yml (ב base 64) והקובץ יושב ב github repo יוצא שלכל מי שיש גישה ל repo יוכל לראות את כל הסודות ששמרנו בקבצי yml (אפילו שזה repo פרטי, זה לא דרך בטוחה לשמור סודות).

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

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

1. USER\_NAME
2. PASSWORD
3. HOST
4. NAME
5. TOKEN

אז נוכל ליצור secret בשם DB וכל אחד מהרשימה יהיה key, והערך שלו יהיה value (לדוגמא, key זה USER\_NAME ו value זה root).

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 בזה, נצטרך למשוך את כל המידע הזה ל cluster את זה אנו עושים עם אובייקט של kubernetes שנקרא external-secrets.io/v1. בסופו של דבר argocd יודע לדבר עם kubernetes secret ולכן מה שבפועל קורה זה שה external-secrets מושך את המידע מ AWS Secrets Manager, יוצר kubernetes secret מתאים ואז argocd משתמש בזה כדי להתחבר לרפוזיטורי הפרטי.

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

## 3.4 הקמת EKS Cluster (משימה 3)

### EKS Cluster Configuration 3.4.1

EKS Cluster מוקם עם הגדרות הבאות:

#### :Cluster Features

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

#### :Node Groups

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

### EKS ל 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 4-5: Load Balancing ו-Application Layer

### 4.1 Ingress ו-Load Balancer (משימה 4)

#### 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**: ניתוב תעבורה בתוך הקלסטר
- **Kubernetes Services**: חשיפת 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-nginx-controller.yaml

```

## 4.2 פריסת אפליקציות (משימה 5)

### 4.2.1 Container Management

אפליקציות מנוהלות כ-containerized microservices:

#### :Container Registry Strategy

- Docker images נשמרים ב-Docker Hub
- Image tagging אוטומטי בהתבסס על timestamp
- Multi-stage builds לאופטימיזציה
- Security scanning של images

### 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 ב templates 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 ארכיטקטורת Secrets Management

הפתרון לניהול סודות מתבסס על שכבות אבטחה מרובות:

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

יתרונות הגישה:

- הפרדת סודות מקוד: אין secrets ב repository
- Rotation אוטומטית: AWS Secrets Manager תומך ב rotation

#### 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

#### 5.2.1 Configuration- External Secrets Installation

הפקודות הבאות הם לצורך התקנת ExternalSecrets ל cluster:

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

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

--create-namespace \

בשביל למשוך את הסודות, אנו צריכים ExternalSecret ו ClusterSecretStore.

ClusterSecretStore: מגדיר איך ומאיפה לקבל את הסודות (AWS Secrets Manager, עם פרטי הגישה).  
ExternalSecret: מגדיר איזה סוד להביא וכיצד ליצור את הסוד בתוך קוברנטיס.

##### 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-external-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: GitOps ו- CI/CD Automation

### 6.1 GitOps עם ArgoCD (משימה 7)

#### Principles ו- ArgoCD Architecture 6.1.1

ArgoCD מיישם את עקרונות GitOps:

#### :Core Principles

- **Git as Single Source of Truth**: כל deployment מוגדר בGit
- **Declarative Configuration**: תיאור מצב רצוי, לא פקודות
- **Automated Sync**: סנכרון אוטומטי בין Git למצב בקלסטר
- **Observability**: ניטור מתמיד של drift detection

#### Configuration ו- ArgoCD Installation 6.1.2

הפקודות הבאות הם לצורך התקנת argocd ל cluster:

```

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

```



```
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
```

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

### 6.2.1 תזכורת ל 3.3 הקדמה למשימות 3 ו 4 ו 5

- ב 3.3 דיברנו על המקרה שצוות הפיתוח דוחפים שינוי לאפליקציה. אמרנו שנצטרך לעשות אוטומציה לשלבים הבאים
1. לבנות את האפליקציה. זה אומר לבנות docker image tag חדש (ופעולות נוספות בהתאם לאפליקציה).
2. לדחוף את התג החדש ל docker hub.
3. לעדכן את ה helm chart values (ב github repo) להשתמש בתג החדש.
4. לקחת את ה values המעודכן מתוך github ולעשות עם זה upgrade ל app helm ב kubernetes cluster.
- נשים לב שאת שלב 4 כבר השלמנו עם argocd ו AWS Secrets Manager.
- נשאר לנו להשלים את שלבים 1 ו 2 ו 3.

### 6.2.2 GitHub Actions Workflow

##### FILE: .github/workflows/deploy.yaml #####

```

name: Build and Deploy to EKS

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

```

וזהו! סיימנו את כל שלבי האוטומציה.

## 7. תהליך הפריסה 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

#### GitHub Actions 7.2.1 - שלב 1-3: Build ו-Publish

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

#### ArgoCD 7.2.2 - שלב 4: GitOps Sync

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

## 8. ניטור ו-Observability

### Monitoring Stack 8.1

#### ArgoCD Dashboard 8.1.1

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

## 9. אבטחה ו-Compliance

### Security Best Practices 9.1

#### Network Security 9.1.1

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

#### Secrets Security 9.1.2

- **No Secrets in Code**: כל ה secrets ב-AWS Secrets Manager
- **Rotation**: מדיניות rotation של secrets

## 10. טיפול בבעיות נפוצות ו-Troubleshooting

### 10.1 בעיות 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 וכיווני פיתוח עתידיים

### 11.1 הישגי הפרויקט

הפרויקט הצליח להשיג את המטרות הבאות:

- אוטומציה מלאה של תהליכי deployment
- אבטחה מתקדמת עם Zero Trust Network
- ניהול secrets מקצועי עם AWS Secrets Manager
- GitOps workflow מלא עם ArgoCD
- Infrastructure as Code עם Terraform

### 11.2 כיווני פיתוח עתידיים

#### 11.2.1 שיפורים טכנולוגיים

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

#### 11.2.2 שיפורי תהליכים

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

## נספחים

### 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: סקריפטים

#### 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 שמופעלת ע"י Scheduler ומיועדת להפעיל או לעצור משאבים בסביבת AWS. היא מזהה את מופעי ה-EC2 עם תג מסוים ומפעילה או עוצרת אותם בהתאם למצב (mode), וגם מעדכנת את הגדרות ה-Scaling של קבוצות ה-Node ב-EKS כדי לצמצם או להגדיל את המשאבים 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 ה-IAM ב-AWS, לבדוק את גיל מפתחות הגישה שלהם, ולזהות מפתחות ישנים מעל מגבלת 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];
    snprintf(cmd, sizeof(cmd), "echo $(( ($(date +%Xs) - $(date -d '%s' +%Xs)) / 86400 ))", create_date_iso);

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

    return atoi(output);
}

Strlist get_old_keys(const char* username) {
    char cmd[512];
    snprintf(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 'AccessKey.[AccessKeyId,SecretAccessKey]' --output text", username);

    FILE* fp = popen(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 מציגה אילו פעולות היו מתבצעות — יצירת מפתח חדש והשבתת המפתחות הישנים — 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 הציג מימוש מקיף של פתרון cloud-native מודרני הכולל:

**הישגים טכנולוגיים:**

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

**יתרונות עסקיים:**

- זמן deployment מהיר ואמין
- אבטחה ברמה enterprise
- חיסכון בעלויות תפעול
- מדרגיות גבוהה
- אמינות ושמירת SLA

### 12.2 רפלקציה על התהליך

הפרויקט חשף אתגרים מעניינים בתחומים הבאים:

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

### 12.3 המלצות להמשך

**לטווח קצר:**

- יישום automated testing ב-pipeline
- הוספת alerting ו-monitoring מתקדמים
- שיפור documentation ו-runbooks

**לטווח ארוך:**

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

## מקורות ביבליוגרפיה

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

---

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

## מצגת הפרויקט

### פרקטיקום טכנאי תוכנה – DevOps

מנחם שפרבר



#### תיאור החברה ולקוחותיה

בינת תקשורת מחשבים הינה חברה מובילה באספקת פתרונות IT, תקשורת, ענן ואבטחת מידע למגזרים הציבורי, הביטחוני, והעסקי בישראל. חטיבת ה DevOps בבינת אחראית על תכנון והטמעה של תהליכי אוטומציה, ניהול תשתיות ענן ו-IT מודרני, ושיפור רציפות הפיתוח והפריסה עבור מגוון רחב של לקוחות. לבינת צוות מקצועי ודינמי, הפועל בסביבה טכנולוגית מתקדמת ומשלב עבודה עם כלים ומתודולוגיות עדכניות מעולמות ה DevOps, Cloud, Agile.



## רקע לפרויקט ודרישות הלקוח

הלקוח פנה לחברת בינת במטרה לבצע שדרוג מהותי לאבטחת הגישה לתשתיות הארגוניות, וזאת לאור סיכוני אבטחה שהתגלו בשימוש בפתרון ה-VPN הקיים. הדרישה המרכזית הייתה להטמיע פתרון Zero Trust Network (ZTN), אשר יאפשר בקרת גישה קפדנית ומבוססת זהות, לצד צמצום מרבי של שטח החשיפה.

בנוסף, הלקוח הציב יעד להקמת סביבת Kubernetes בענן AWS, שתספק יכולות ניהול מתקדמות, גמישות תפעולית, ואוטומציה מלאה של תהליכי פריסה ותחזוקה של אפליקציות ותשתיות. הפרויקט כלל אינטגרציה של מנגנוני ניהול סודות מאובטחים, לצד יישום מתודולוגיית GitOps, על מנת להבטיח פריסה וניהול גרסאות בצורה עקבית, מאובטחת ויעילה.

## שיקולי תכנון

מודולריות – בניית המערכת כך שכל רכיב פועל בנפרד, מה שמקל על תחזוקה ושדרוגים.

אבטחה מתקדמת – שילוב מנגנוני הגנה חכמים למניעת גישה לא מורשית ואיומי סייבר.

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

יכולת התרחבות עתידית – תכנון המערכת כך שתוכל לגדול ולהתאים לצרכים חדשים בקלות.

## כלים וטכנולוגיות



```
# 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
}
```

### AWS Environment Setup

הגדרת ארכיטקטורת רשת: VPC, תתי-רשת, Routing,  
כתיבת מודולי Terraform להקמת אוטומטית,  
הקמת EKS Cluster הגדרת Security Groups

```
# 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
}
```



Kubernetes (EKS) Setup  
Bootstrap של EKS באמצעות Terraform,  
הגדרת גישה לקלאסטר עם kubectl,  
הקמת Node Groups ואוטוסקיילינג



### Zero Trust Networking (Twingate)

הטמעת Twingate כפתרון ZTN,  
הפעלת Connector בתוך ה-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
  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
}
```

## כלים וטכנולוגיות

GitOps with Argo CD  
התקנת Argo CD על הקלאסטר  
הגדרת Argo Applications  
חיבור לריפוי כ-Source of Truth

```
# gitops/cluster/namespace.yaml
name: Build and Deploy to EKS
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: build-and-deploy
  namespace: argo-cd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd
    targetRevision: HEAD
  destination:
    namespace: argo-cd
    server: https://kubernetes.default.svc
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace
```



### Secrets Management

בחירת מנהל סודות: AWS Secrets Manager  
הזרקת סודות לאפליקציה בצורה אוטומטית  
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:
          accessKeyIDSecretRef:
            name: access-secret
            key: access-key
            namespace: external-secrets
          secretAccessKeySecretRef:
            name: access-secret
            key: secret-access-key
            namespace: external-secrets
```



### Ingress and Load Balancer

התקנת NGINX או AWS ALB Ingress  
חשיפת שירותים עם Ingress, עדכון  
Route 53 ב-DNS

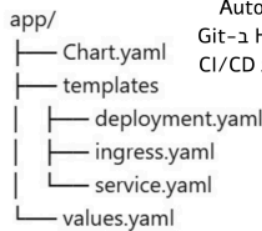
```
# values/nginx-controller.yaml
controller:
  progressDeadlineSeconds: 600
  ingressClassSource:
    name: nginx
    enabled: true
    default: true
  service:
    type: LoadBalancer
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
      service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
      service.beta.kubernetes.io/aws-load-balancer-ssl-target-type: "Instance"
      service.beta.kubernetes.io/aws-load-balancer-ssl: "aws-ssl-cert-arn:arn:aws:acm:eu-central-1:123456789012:certificate/12345678-1234-5678-9012-123456789012"
  # This is how to add ingress routes
  # helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
  # helm repo update
  # helm upgrade --install ingress-nginx ingress-nginx --namespace ingress-nginx --create-namespace
```



## Application Deployment

שימוש ב-Helm ליצירת Templates, יצירת קבצי מניפסט:  
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 Terraform ו-Helm ב-Git  
אוטומציה של הפריסה באמצעות CI/CD



```
# 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 }}
          envFrom:
            - secretRef:
                name: {{ .Values.deployment.container.envFromSecretName }}
          volumeMounts:
            - name: secrets-store-inline
              mountPath: /mnt/secrets-store
              readOnly: true
          volumes:
            - name: secrets-store-inline
              csi:
                driver: secrets-store.csi.k8s.io
                readOnly: true
                volumeAttributes:
                  secretProviderClass: mnt-secret
```

```
# values.yaml
namespace: app

deployment:
  name: hello-counter
  replicas: 1
  container:
    name: counter
    image: mwendish/count-users-db
    port: 80
    envFromSecretName: app-secrets

service:
  name: service-hello-counter
  port: 80

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

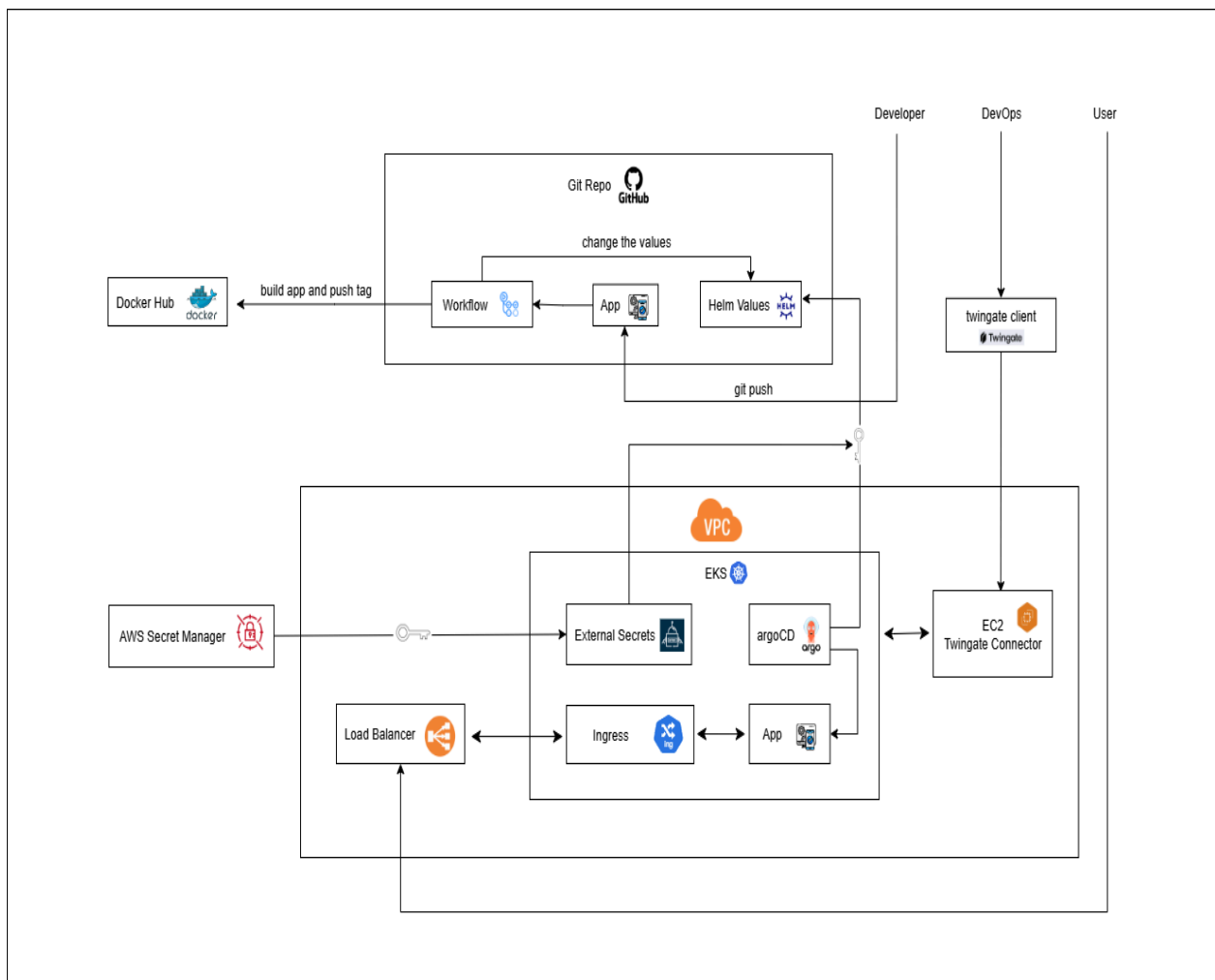
volume:
  type: persistentVolumeClaim
  claimName: mysql-pvc
  hostPath: /mnt/data

liveness:
  enabled: true
  secret:
    CREATE DATABASE IF NOT EXISTS visitor_counter;
    USE visitor_counter;
    CREATE TABLE IF NOT EXISTS users (
      id INT NOT NULL AUTOINCREMENT PRIMARY KEY,
      username VARCHAR(50) UNIQUE NOT NULL,
      password VARCHAR(120) NOT NULL,
      created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
```

## יחסי גומלין עיקריים בין מרכיבי המערכת

Twingate מאפשר גישה מבוקרת רק למורשים ומעביר תעבורה מוצפנת ל-EKS דרך קונקטור בתוך ה-VPC.  
Terraform אחראי על כל הקמה תשתיתית – EKS, Subnets, VPC, Security Groups, ועוד.  
Argo CD מושך את הגדרות הפריסה מתוך Git ומעדכן את הקלאסטר אוטומטית.  
Helm משמש לפריסת אפליקציות בגרפים מודולריים – כולל שירותים, Ingress, Secrets, וכו'.  
AWS Secrets Manager מספק סודות בצורה מאובטחת לקלאסטר דרך שירות IRSA.

# ארכיטקטורה ותהליך זרימה



## סיכום מרכיבי הפרויקט

במסגרת הפרויקט נבנתה תשתית ענן מתקדמת ב-AWS באמצעות Terraform, שכללה יצירת VPC, תתי-רשתות, קבוצות אבטחה ו-EKS לניהול קוברנטיס. הוטמע פתרון Zero Trust Network באמצעות Twingate, על מנת לספק גישה מאובטחת ומבוססת זהות לכלל הרכיבים. בקלאסטר עצמו נפרסו רכיבי ניהול מרכזיים כמו Ingress Controller ו-Load Balancer, לצד פתרונות לניהול סודות בצורה מאובטחת באמצעות AWS Secrets Manager או External Secrets Operator. תהליכי הפריסה והעדכון התבצעו בגישת GitOps עם Argo CD, בשילוב Helm לניהול תבניות האפליקציות, וכל זאת תחת מערך אוטומציה המאפשר שליטה, ניטור והתרחבות עתידית בצורה פשוטה ויעילה.



Twingate



GitHub



## כיווני פיתוח עתידיים

### לטווח קצר

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

### לטווח ארוך

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