*Project Title*

**Automated Web Hosting Solution**

**AutoScaleHost**

*‘One Click, Any Environment, Fully Deployed’*

By

Sanya Sinha

**Table of Contents**

[**Abstract** 3](#_Toc204114246)

[**1. Introduction** 4](#_Toc204114247)

[**1.1 Problem Statement** 4](#_Toc204114248)

[**1.2 What is AutoScaleHost ?** 4](#_Toc204114249)

[**2. Key Terms** 5](#_Toc204114250)

[**3. Tools and Technologies Used** 5](#_Toc204114251)

[**4. Prerequisites** 6](#_Toc204114252)

[**5. What does AutoScaleHost Deploy ?** 6](#_Toc204114253)

[**6. How this Works ?** 7](#_Toc204114254)

[**7. File Structure** 8](#_Toc204114255)

[**8. How to Run Locally** 9](#_Toc204114256)

[**9. How to Deploy via UI ?** 9](#_Toc204114257)

[**10. Notes for Running on your Machine** 9](#_Toc204114258)

[**11. UI Preview** 10](#_Toc204114259)

# **Abstract**

AutoScaleHost is a cloud-based automated web hosting platform that helps users create a uniform and efficacious method for deploying web apps across development, UAT (test) and production environments. By applying Infrastructure as Code (IAC), AutoScaleHost allows for the deployment, configuration and scaling of web servers and load balancers based on a virtual machine instance within any environment with a single click or tap of a button, customized for each designed environment's requirements to eliminate the risks of manual configuration, provide consistency of operations, and thereby reduce the time spent in deployment.

In the rapid pace of today's environment, organizations require proven, repeatable, and simply scalable infrastructure to enable processes for switching workloads that are being updated/delivered at a more frequent schedule. AutoScaleHost provides solutions to organizations to help maintain that reliability by accelerating the deployment lifecycle while at the same time reducing deployment effort or reducing the chances of making mistakes, which has become a must-have among teams in pursuit of speed/reliable operations. Teams mustn’t have any up-front cost objectives or needs stopped/ hindered based on their compound need for proper levels of accountability and appreciation from their organization's stakeholders across the globe who are consuming their products in the cloud.

# **1. Introduction**

This document describes how to build an **automated web hosting solution** on Microsoft Azure using Infrastructure as Code (IaC). The solution provisions two virtual machine (VM) web servers behind an Azure Load Balancer. The entire setup can be deployed to Development, UAT (test), and Production environments with a single click or command, using environment‑specific parameters.

## **1.1 Problem Statement**

**Automated Web Hosting Solution with single click replication to Uact and Prod Environment:**

An IAC Tool leveraged to deploy Virtual machine-based web servers, Load balancer. A website hosted on two virtual machine instances added to the Backend of an Azure Load balancer. IAC tool capable of dynamically deploying the setup to Development, Uact (test), and Production Environment by using Parameters specific to an environment.

## **1.2 What is AutoScaleHost ?**

The **AutoScaleHost** project allows you to easily create and deploy a website in virtual machines (computers in the cloud) with a single click. It is a full-stack Infrastructure as Code (IaC) project that automates the provisioning and deployment of scalable web infrastructure on **Microsoft Azure** using **Terraform**, with a **Node.js-based user interface**.

You can have the project work in various stages of software development:

* **Development:** A version to test new code.
* **UAT (User Acceptance Testing) or UACT** : A version to assure everything works as expected before going live.
* **Production:** The final version used by real end users.

The system automatically sets up:

* Two Virtual Machines (VMs)
* A Load Balancer (to manage traffic between the VMs)
* Apache Web Server
* Public IPs for web access

# **2. Key Terms**

|  |  |
| --- | --- |
| **Term** | **What It Means (Simply)** |
| **Virtual Machine (VM)** | A computer in the cloud. |
| **Website Hosting** | Putting your website files on a server so others can view it online. |
| **Load Balancer** | A tool that shares website traffic between servers to keep things fast and reliable. |
| **Environment** | A version or stage where code is deployed: Dev, Test, or Live (Prod). |
| **Infrastructure as Code (IAC)** | Writing code (like a script) that creates cloud resources for you automatically. |
| **Azure** | Microsoft’s cloud platform – it lets you run VMs, databases, and more. |
| **Terraform** | A popular IAC tool that lets you write infrastructure setup in code and deploy it easily. |
| **UI Integration** | A front-end interface to trigger deployments. |

# **3. Tools and Technologies Used**

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| **Microsoft Azure (Student Account)** | Cloud platform for hosting VMs and load balancer |
| **Terraform** | IAC tool for automating infrastructure creation |
| **Visual Studio Code (VS Code)** | Code editor |
| **Git & GitHub** | Version control and project repository |
| **Azure CLI** | Command-line interface to interact with Azure |
| **Node.js + Express** | Backend for the user interface |
| **HTML/CSS + JS** | Simple frontend for deployment interface |
| **Shell Scripting** | Automation scripts for provisioning |

# **4. Prerequisites**

Before using AutoScaleHost, ensure:

1. **Azure Setup**

* A valid Azure account
* Service Principal created with:
  + Client ID
  + Client Secret
  + Subscription ID
  + Tenant ID
* Permissions to create resources in Azure

1. **Local Setup**

* Install:
  + Terraform
  + Azure CLI
  + Node.js & npm
  + Git

# **5. What does AutoScaleHost Deploy ?**

1. **Two Azure Virtual Machines (VMs):**

* Ubuntu-based
* Apache installed
* Host a sample webpage

1. **One Azure Load Balancer:**

* Backend pool includes both VMs
* Rule to forward traffic on port 80 (HTTP)

1. **Dynamic Environment Configuration:**

* Dev (dev.tfvars)
* UAT (uat.tfvars)
* Prod (prod.tfvars)

# **6. How this Works ?**

**1. User Interface**

* UI built in Node.js with form options for Dev/UAT/Prod.
* User selects environment and clicks "Deploy".

**2. Backend API**

* The Node.js server receives the request.
* It executes terraform apply with the corresponding variable file (e.g., dev.tfvars).

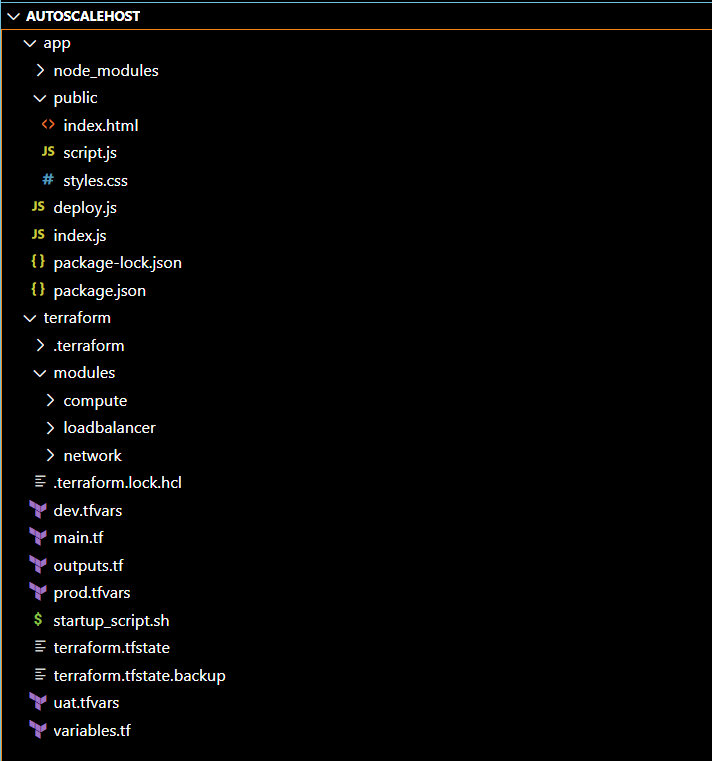
**3. Terraform Execution**

* Reads main.tf, variables.tf, and \*.tfvars
* Deploys:
  + Resource Group
  + Network Interfaces, VNet, Subnet
  + Public IPs
  + Load Balancer and Backend Pool
  + VMs with Apache setup

**4. Apache Page Access**

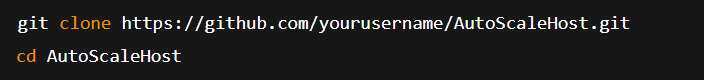
* After deployment, users get public IPs of the VMs.
* Load Balancer IP also routes traffic evenly.

# **7. File Structure**

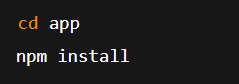


# **8. How to Run Locally**

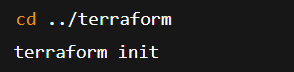
1. Clone the Repository



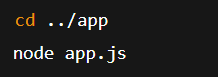
1. Install Node Dependencies



1. Initialize Terraform



1. Run the Node.js App



Access the app at: [**http://localhost:5000**](http://localhost:5000)

# **9. How to Deploy via UI ?**

1. Open the app in browser (localhost:5000)
2. Choose environment (Dev / UAT / Prod)
3. Enter Azure credentials:

* client\_id, client\_secret, tenant\_id, subscription\_id

1. Click **Deploy**
2. View status and public IPs returned

# **10. Notes for Running on your Machine**

1. Keep dev.tfvars, uat.tfvars, prod.tfvars **inside the terraform/ folder**, not under env/.
2. Ensure Terraform CLI is installed and added to PATH.
3. You may need to modify:

* Resource Group name
* Region (location)
* VM size (e.g., Standard\_B1s)
* Apache script (custom webpage)

# **11. UI Preview**

