Cross-Cloud Deployment Tool

Rodrigo Ferreira

BSc. (Hons.) Computer Science

March 24th, 2023

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date: March 24th, 2023

Signed: Rodrigo Ferreira

Abstract

This report details the evolution and development of a virtual machine (VM) selector web tool, created to put an end to the lack of easy and user-friendly options for multiple cloud provider selector tools that benefit both users and companies. Even with the market of cloud and virtualization skyrocketing these past years, there is a scarcity of tools that help users get the best option for their needs

With this web tool's help, users can now focus on choosing the best option for their needs instead of searching each cloud provider and spending hours selecting the most suitable VM.

Table of Contents

Figures

Chapter

1. Introduction

Virtualization is the possibility of software creating a layer on top of the hardware that then permits the elements of a computer, like memory, processor, storage and more to be split into multiple virtual computers [1]. With each one of these divided machines, it is possible to run multiple operations systems and programs, giving the user more resource efficiency, easier control and faster provisioning [11].

A computer system that has been virtualized or emulated is referred to as a "virtual machine" (VM) in computing. Computer architecture serves as the basis for virtual machines, which provide all the features of a physical computer [1]. These VMs can be designated into two classes:

System virtual machines replicate a complete computer system and are used as stand-ins or testing grounds [1].

Process virtual machines, also known as Managed Runtime Environments (MREs), which can also perform a single process in an OS independent habitat [1].

Thus, virtual machines are computer simulations that offer the same practicality as physical computers do, but just as a copy.

The concept of VM was initially launched in the 1960s. The primary implementations incorporated the use of corresponding time-sharing systems, which enabled multiple users to have access to an identical computer [1]. These premature virtual machines made it look like each user was using the computer separately but in reality, it was executing one program at a specific time, permitting access to their own digital materials as well. Ever since then, virtualization technology has evolved immensely and is now an essential part of modern systems, primarily in cloud computing, where it offers a shapeable and scalable base for hosting numerous applications and services [1].

The importance of virtual machines in the contemporary computing scope can be experienced through the growth of the virtualisation market. Small and medium-sized enterprises are increasingly adopting VMs in their management, taking advantage of the benefits they offer [2]. This trend got a massive boost during the Covid-19 pandemic, as companies motivated their staff to work remotely [3].

The virtualization market was valued at US$13.449 billion in 2020, with forecasts theorising it will reach US$45.578 billion by 2027 [2]. This expansion is caused by the strengths of VMs, such as their flexibility, scalability, and cost effectiveness.

During my inquiry on VMs and Cloud Providers, I noticed a difference in VM selector tools across distinct cloud providers. Despite the fact that each provider has its own tools for comparing, there is a lack of options for the same goal across multiple providers and picking the most suitable choice for particular necessities.

The fix to this problem is to build a tool that permits users and companies to review virtual machines over various cloud providers and choose the optimal selection for their circumstances. By delivering a comprehensive evaluation of various VMs, the tool will guide users in making educated decisions and utilizing the conveniences of virtual machines in their operations to the maximum.

1. Aims

The goal of this project is to create an effortless-to-operate tool that allows users to evaluate virtual machines over multiple cloud providers, form the necessary code to construct a VM and present comprehensive instructions for employing the tool.

To accomplish this, the resolution will focus on achieving the following key points:

- Develop a user-friendly tool that can be used by users with small or no expertise related to Terraform. The tool will contain a basic and intuitive interface.
- Compare virtual machines over diverse cloud providers, allowing users to select the requirements needed for their desired virtual machine.
- Generate the necessary Terraform code to construct a virtual machine on the user's preferred cloud provider. This will help users quickly and easily put together a VM without the necessity to learn coding or any technical competence.
- Develop a detailed guide with all the necessary instructions for users to use the tool without any trouble.

Chapter

2. Background

1. Related Concepts

To guarantee its success, it is vital to shape a preliminary understanding of the base concepts and the world in which the undertaking will be executed prior to commencing.

Cloud

Cloud technology is considered a server or a group of servers that enables the users/companies to have access to files, programs, and databases, without these users having to have physical access. Virtualisation plays a big part in cloud computing, opening the way for users to go across multiple servers [4]. This technology allows users to utilise all the functionalities, services and applications on virtual computers that operate like physical ones, that are also more efficient and scalable [4]. Everyone has access to cloud services by way of software-as-a-service (SaaS), platform-as-a-service (PaaS), infrastructure-as-a-service (IaaS), and function-as-a-service (FaaS) models [4].

SaaS delivers users with entry to software applications over the world wide web, without needing local installations [4]. PaaS allows members to access and develop applications on a cloud platform, removing the requirement for hardware and software infrastructure [4]. IaaS offers users access to virtualized computing resources, including servers, storage and networking, enabling the more adaptable and scalable deployment of services [4]. FaaS lets users deploy applications in the cloud which can be quickly operated and handled by the cloud provider.

Cloud operations have private, public, hybrid, and multi-cloud options. Private clouds are run by a single enterprise for its own use, and public clouds are offered by cloud suppliers to the public [4]. Hybrid clouds merge private and public cloud services to provide maximum flexibility and scalability [4]. Multi-cloud implementations require the use of multiple cloud suppliers to maximize services and minimize risks [4].

Cloud Providers

Cloud service suppliers (CSPs) offer cloud computing facilities, such as infrastructure, software as a service, and computer resources [5]. These facilities are charged through varied pay-as-you-go stipulation models, permitting organizations to pay only for the resources they consume [5].

CSPs present multiple pluses, such as cost and flexibility, scalability, movability, and disaster recovery [5]. Nevertheless, utilizing cloud services also proposes correspondences, such as concealed costs, cloud transfer, cloud security, performance and outages, hard to interpret contract terms, and trader lock-in.

Organisations should consider their goals and risk profile, both in the present and in the long term, previous to selecting a CSP [5]. CSPs have their strengths and weaknesses that are valuable looking into [5]. Organizations should also debate contracts and service-level agreements actively, have a cloud exit program, and aspect in additional staffing requirements for monitoring and managing cloud use to be cost-efficient [5].

The top seven cloud market players are Amazon's AWS, Microsoft Azure, Google Cloud Platform, Alibaba Cloud, Oracle, IBM, and Tencent Cloud [6]. Each of these CSPs has its exclusive advantages and disadvantages, a price system, and services available.

Right now, AWS commands the highest share of the market with a wide selection of cloud infrastructure services; like computing, storage and databases [6]. Microsoft Azure, with its intense focus on the corporate sector, has an array of services, among them, hybrid cloud solutions and analytics powered by AI [6].

Google Cloud is well-known for its machine learning and large collection of data, while Alibaba Cloud is way ahead in the Chinese market and provides economical solutions for small and medium enterprises [6].

Terraform

HashiCorp Terraform is an infrastructure as a code tool [7]. It gives the user the ability to encode cloud and on-premises resources in comprehensible formats that can later be versioned, reused, and shared [7]. Terraform also encourages different cloud providers like AWS, Azure, Google Cloud, and Kubernetes, to make use of APIs to create and manage resources on various stages and administrations [7].

The core of the Terraform process is made up of three stages: write, plan, and execute [7]. In the write phase, users characterise resources over numerous cloud suppliers and administrations. In the plan stage, Terraform produces an execution plan describing the infrastructure it will create, upgrade, or delete [7]. In the apply stage, Terraform performs out the proposed operations in the proper request [7].

Terraform takes an immutable approach to infrastructure, diminishing the perplexity of overhauling or changing services and infrastructure [7]. It tracks infrastructure changes, creates a plan, and prompts for endorsement before modifying infrastructure [7]. Terraform configuration files are declarative, meaning they describe the end state of infrastructure, and Terraform handles the underlying logic [7]. Terraform also supports reusable configuration components called modules that determine configurable accumulations of infrastructure, sparing time and urging best practices.

2. Analysis of Similar Applications

To better compare and analyse the viability of the project, it is important to review existing applications with similar functionality. The following applications will be discussed, highlighting their pros and cons to support the choices made in the project.

Azure

App: https://azure.microsoft.com/en-us/pricing/vm-selector/

Microsoft Azure, a cloud provider, has built a useful tool for helping users in the selection of the right virtual machine (VM) on their platform. This tool has access to the latest available data about their services and many configuration options for users looking for more flexibility. However, it may give an impression of complexity to some users, especially beginners. In addition, it requires going through at least five different steps for even the simplest virtual machine; making this tool time-consuming and ineffective when creating a VM. Furthermore, this tool aids the search for a VM but lacks the extra guide in creating it directly in the user's account. And lastly, because of its design, it doesn't present the user with many alternatives when showcasing the results of the search.

AWS

App: https://aws.amazon.com/ec2/instance-types/

During my research before starting the project, there was a tool developed by AWS that helps users to choose the best option, but it came across a website that splits the types of virtual machines and informs the users which machines to select in specific cases. This website might be helpful for more advanced users and those who have those conditions, but for less experienced users and those who need more flexible machines, it might be confusing and challenging to choose the best option. Also, another disadvantage is that the page does not allow the creation of virtual machines on the client's account.

Azure Price

App: https://azureprice.net/

Another available resource is Azureprice, which is the most similar tool that was found during the search for this project. This tool provides a list of virtual machines with their prices and highlights the cheapest region to use them, which also allows users to search for specific requirements. But it does not allow users to create virtual machines in their accounts, which may be a significant drawback for some users.

The review of these applications highlights the lack of user-friendly options that search virtual machines in different cloud providers with the user requirements. It also accentuates the absence of resources aiding in the creation of the VM within the tool. The conclusion to these observations is the need for a beginner-friendly tool with a wide range of options and creation guides.

Chapter 3. Design

This part of the report outlines the major design choices made during the creation of this project. These selections are important in understanding the development procedure, objectives, and flow of the project.

To start with, it will be unveiling the tools employed in this endeavour, going over the programming languages, libraries and databases that supported it. Additionally, it will describe the folder and code design that was integrated during the implementation.

In continuation, it will explain the user interface of the tool and the necessary steps the user is required to carry out in order to yield a virtual machine in their account. Highlights incorporate the input areas required, and the validity assessments, which are done prior to the making of the virtual machine.

1. Languages and Libraries

The 1st phase of the project revolves around the visualization of a web app as the ideal platform, resulting in Node.js, a JavaScript environment to run code on the server side, being established as the language of choice. This decision was due to the extraordinary dominance of JavaScript in the server-side programming language market, possessing a share of over 90% [8]. Additionally, Node.js is the most common framework on the web [9]. With this ubiquity, it has the greatest online user base, which furnishes plentiful samples and learning materials for developers to utilize.

Subsequently, npm, a significant package manager for Node.js, was used to find the most fitting packages to boost up the project's workflow and simplify the process.

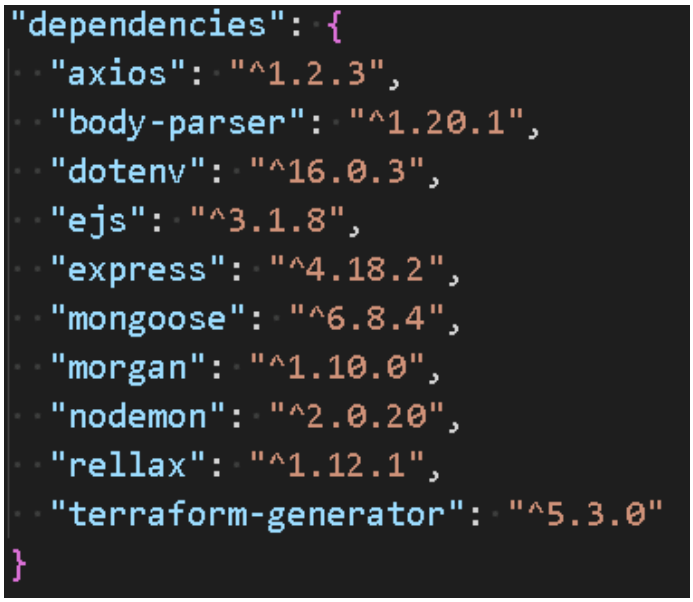

Figure 1 - Project Dependencies

Axios: HTTP client for the browser and node.js. Used to get all the information from the API done on the project.

Body-parser: Node.js body parsing middleware. Used to pass the information from the front end to the back end of the applications, for example, the users' requirements.

Dotenv: Loads environment variables from a .env file. The .env file is used to store private information that is not supposed to be shared on the repository. This is done for security reasons.

Ejs: It stands for embedded JavaScript. It adds some logic to the front side of the tool, having the possibility to use JavaScript variables as well.

Express: Core of the web applications. Express is a web framework for node.js. It is fast, minimalist and unopinionated.

Mongoose: This library is a MongoDB object modelling tool. It works in an asynchronous environment. It is used to connect the database and find the right documents.

Morgan: HTTP requests logger middleware.

Nodemon: Nodemon helps save time developing the project. When notice that there are changes in the code it restarts the server automatically. Otherwise, the server would need to turn down and turn on every time there was a change in the code.

Terraform-generator: Uses node.js to generate Terraform code. The result of this module is a Terraform plain text. Fundamental to this project.

2. Database

In the context of databases, the project had to follow between SQL or NoSQL options. Initially, the SQL type of databases, particularly MySQL, was taken into consideration, but it turned out that this was not the ideal resolution, having problems such as the incapability to bring in data from JSON documents, which were the output kind from the cloud providers' APIs and incorporated all the needed information about the virtual machines. So, the conversion was necessary and done before evolving any other part of the project.

Based on the search, MongoDB is the most reliable database program available. As a document database, it offers scalability, flexibility, and robust indexing and querying capabilities [10]. MongoDB also stores data in adaptable documents, similar to JSON, which allows for fields to vary between documents and data structures to evolve over time [10]. This type of storage streamlines the connection between the database and the application, and node.js has a well-documented library that seamlessly integrates with MongoDB.

In addition to its flexible storage options, MongoDB offers tools that enhance developer productivity. For instance, the company provides a free tool called Compass, which streamlines database management and enhances the user experience.

3. Folder Structure

Throughout this project, it was prioritized the organisation of the folder structure from the start. This approach not only facilitated the project's development but also sets the foundation for future work and potential collaboration. An organized environment is crucial for any project's success, and our folder structure will help maintain a clear and efficient workflow.

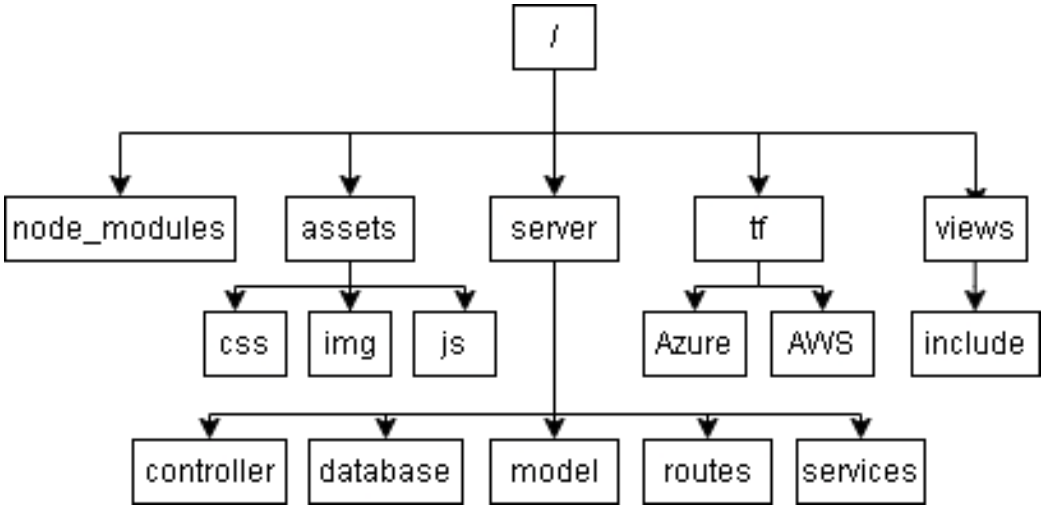

Figure 2 - Folder Structure

- /: Root folder. All the projects are within this folder.
- /node_modules: This is the folder used to store all the previously referred modules, keeping them in one place.
- /assets: Contains all the front-end extra files.
- /server: Contains all the back-end files.
- /tf: This file contains the files and programmes that are necessary to run the terraform.
- /views: The views files contain all the ejs files.
- /assets/css: These css files are used in the front end of the web app.
- /assets/img: Contains all the images used on the project.
- /assets/js: Contains all the JavaScript files used on the front end.
- /server/controller: Includes the controller and created files. Controller files are used to search on the respective collections and the create files are used to generate the terraform files

- /server/database: Connection with the database.
- /server/model: Model files connected with the collections on the database.
- /server/routes: Route files with all the routes that the web app supports.
- /server/services: Include the renders file where the axios module works with the API.
- /tf/Azure: Contains the generated terraform file.
- /tf/AWS: Contains the generated terraform file.
- /views/include: Contains all the files that can be reused on the front end.

4. Code Structure

Code structure is essential to the development of this project. Following a defined structure is essential to stay organised and facilitating debugging. Furthermore, maintaining a consistent structure helps increase productivity.

5. Usability

This segment is going to cover any technical part that the project needs to be functional. it is going to Inform the programs and computational functionalities to obtain the expected results. For the normal user, a GitHub Readme was created to address this problem. It is going also to guide how to create a virtual machine on the tool.

To begin with, the project needs to be on the user's machine, this can be accessed by downloading from the Git Hub repository or shared by other users. For the tool to be able to create a VM there are some prerequisites that need to be met beforehand.

Firstly, to be able to run the project the user needs to have on their machine npm (with the version above or equal to 8.19) and node (with the version above or equal to 18.12.1). Then, the user will be able to run the commands 'npm install' and 'npm start' respectively, to successfully run the project. But before running the tool, it is necessary to create the database. For this, the folder data presents the two collections needed. The user will then create the two collections on mongo DB and change the information, if necessary, on the .env file.

Afterwards, the tool would require the user to introduce the following to his machine: Terraform (with a version equal to 1.3.6), Azure command-line interface, and AWS command-line interface (with versions 2.93 and 3.9.11 respectively) with both programmes, logged-in with the user's cloud provider's account. Concluding all these previous steps the tool would be expected to create a VM on the user cloud account correctly.

Creating a VM is a simpler task than ever. The user would need just a few steps to have it created on his account. After the user has chosen the cloud provider (or not), it is time for the user to go to one of the search pages presented to him on the menu and choose the best VM for their requirements.

In addition, the user should press the 'create' button on the VM that desires to create, when the button is pressed, it is redirected to another page to confirm if the if that is really the VM that they want to create. A terraform file will open at the same time that it redirects to the page; this presents the terraform code file that the user would like to run. The user will have the opportunity to edit this file and add more resources if they wish.

Finally, on the last page when the user is redirected from the last one. It will exhibit two buttons, the 'plan terraform' and 'apply terraform'. These buttons are self-explanatory, the first button will open a text file with the terraform plan (the execute plan from terraform) and the second button will implement the terraform code into the user account.

6. User Interface

The user interface (UI) is an integral part of this project. For this matter, it was considered a simple and efficient UI for users with less experience being able to create a VM without any problem. Other aspects related to low visibility and low motor movement were considered.

The figure below shows the main page of the tool. On the main page only appears the title, the slogan, and the cloud providers that the tool can operate. Additionally, all the pages on the tool, the main page has access to the menu that can be found when pressing the burger menu on the navbar.

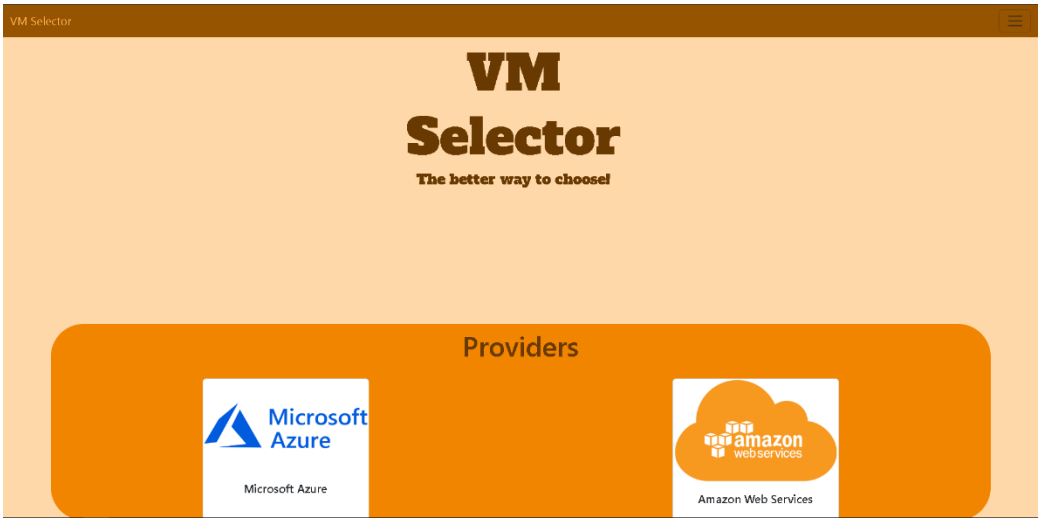

Figure 3 - Tool Main Page

When pressing the burger menu as it is possible to see below, it is possible to notice the 3 cloud providers links that redirect to the search pages. The first one redirects to the page where you can search and select virtual machines from all providers, the second one redirects to the Azure search page, and finally, the last one redirects to the AWS search page.

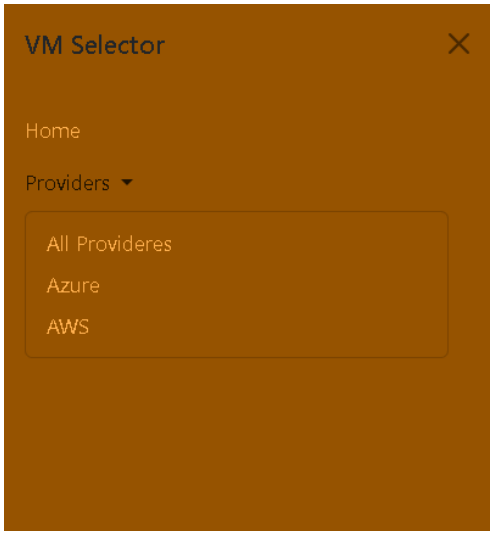

Figure 4 - Tool Menu

Below, we have an example of a search area. On this location, the user is able to select the conditions that the virtual machine would need to hold to be able to support the requirements. The needs only to select one requirement to be able to execute the search.

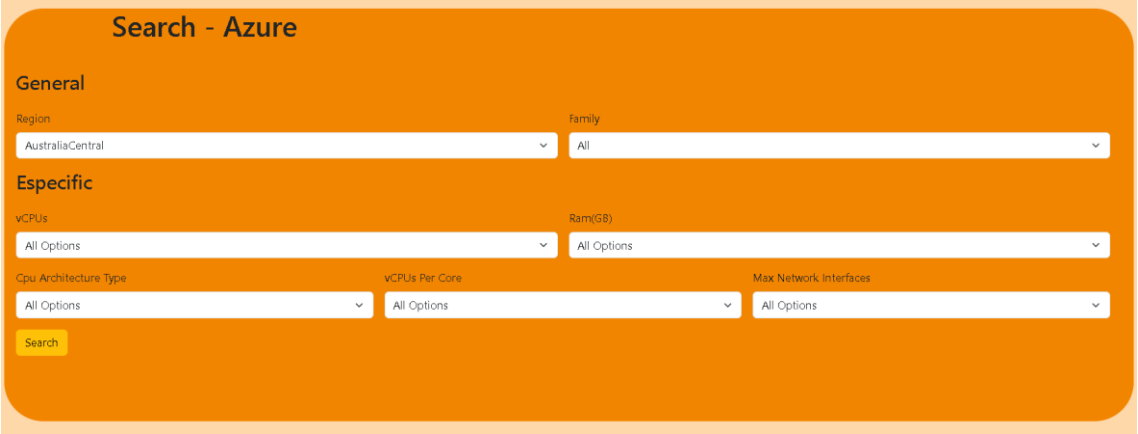

Figure 5 - Search Area

Beneath, it is possible to see a possible output from a search. The user would be able to check what the capabilities of the machine are by pressing the ‘capabilities’ button, which opens a modal with all information. Therefore, after the user decides each machine is the best, the ‘create’ button can be pressed, redirecting to the create page.

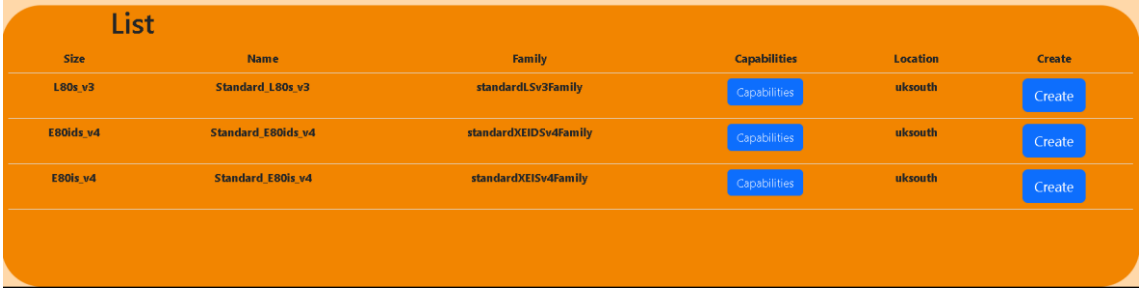

Figure 6 - Output List

On this next very simple page, the user could verify if he selected the correct machine before generating the terraform code. It is presented to the user the name of the cloud provider, the name of the virtual machine and finally a button to then be able to generate the necessary Terraform code. When the button is pressed it is as well redirected to the next page.

Figure 7 - Create Page

This would be the last page presented to the user before creating the VM on the user's account. This page is like the others simple and efficient. On the left button, the user can ask the plan for the terraform plan before ultimately applying the terraform code.

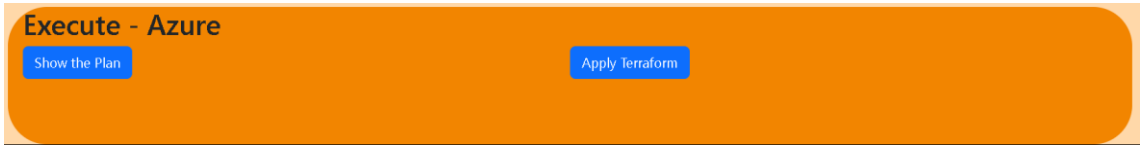

Figure 8 - UI to execute Terraform

On the UI it was followed a colour palette that enables a better distinction of areas of the page, it was also taken into account the font size on all pages. Additionally, the buttons were coloured into a very different colour from the palette, to facilitate the user identifying the buttons on the page.

Chapter

4. Architecture

1. Code

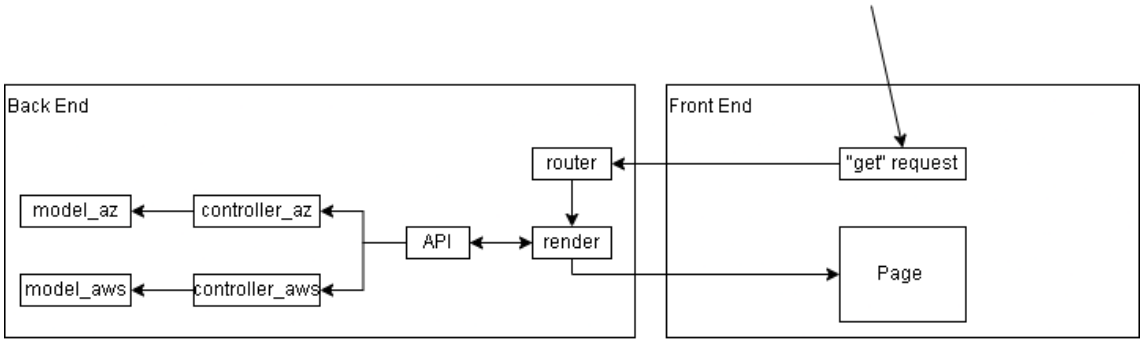

Figure 9 - Code Architecture

To complement the figure above, here is a brief explanation of the process. In the front end, the user sends a "get request" to the back end, which can be a simple main page render or a search for specific virtual machine requirements. The router interprets this request and calls the appropriate method on the “render”.

If needed, the render method will call the API developed for this project to obtain all the necessary information to render the page for the user. The API works with the information passed in the “get request” and sends the requested data to the “render” method. Finally, the back end sends a response to the user.

2. Database

The database has a straightforward structure, consisting of two collections named "az_instances" and "aws_instances". These collections hold all the information about the virtual machines provided by the cloud providers. Because of the extensive number of documents present in these collections, it was essential to build indexes to guarantee quick searches. It was taken advantage of the Compass Tool given by MongoDB to build these indexes, which will quicken search speed and deliver a more desirable user experience.

Chapter

5. Implementation

1. Obtaining Data

Gathering the essential data for the project is an indispensable preliminary step that must be completed before beginning the project. As outlined in my proposal, this project centers around the two biggest cloud suppliers in the market, Microsoft Azure and Amazon Web Services (AWS).

Due to the relevance of the data for this project, in-depth research was performed before to verify if and how we can secure the information on the virtual machines accessible from both cloud suppliers. After considerable effort, a solution was spotted to access the required data. However, in both cases, it was needed to rely on their command prompt APIs rather than extracting the data directly from the cloud providers' platforms.

In addition to installing the APIs on the project machine, we needed to execute specific commands to acquire the desired output.

For AWS, we had to specify the region from which we wanted to obtain instances. Therefore, we could only obtain instances from one region at a time.

‘aws ec2 describe-instance-types --region > output.json’

Conversely, for Azure, we didn't have any specific requirements, and we were able to retrieve all instances from all regions simultaneously. We used the "--show-all false" option to exclude instances that are in testing or those that have been discontinued.

‘az vm list-skus --show-all false --output json > output.json’

2. Searching

The search operation was a core feature for this project. Having spent time to search the best format and method to keep the data secure beforehand was helpful in this part of the development of the project.

This part of the tool was made within the controller folder. Within this folder there are the files responsible for the search and creation of the VMs. In order to implement this section, it was necessary to use a node module called ‘mongoose’, which is responsible for the connection with the database and performing any type of action on it.

Firstly, it was made sure that the mongoose module could be accessed within that file. Additionally, for the code to be able to be called from another page/function the export method was applied. The find function is defined as an asynchronous function – this is fundamental to functions that perform multiple database queries.

Furthermore, it is divided into four sections. The first one is receiving all the data from the front end and placing it on JavaScript variables. The second one is verifying what was required by the user and adding it to a variable. With the method used, the search is faster and simpler, shortening the waiting time. The third section refers to the pursuit within the database for the requirements that the virtual machines have, for the user to then be able to search on the front end. Finally, the send method is used to dispatch all the information.

3. Display

Displaying the results is as important as the search. When the information is sent to the front end as a JSON type who performs the display is EJS. With the help of some logic and the right format, it was possible to display all the information related to the search.

4. Generating, Plan and Apply

The execution part is involved in generating the terraform code into a terraform file, the possibility of the user checking the plan before applying, and finally the proper terraform application. All this process is done again on the controller file, but in a more organised manner in a different file than in the previous sub-section.

This type of generation is possible by terraform-generator module. This module does the difficult task of generating Terraform code from JavaScript. Executing this part of the code will open the terraform file for the user to be able to edit within his best knowledge to add or alter the code generated. After the user edits the terraform file, it is time to plan or apply the terraform code.

Chapter

6. Testing

During the testing phase, searching and choosing the virtual machine was skipped. After choosing the virtual machine, the correct output would be redirecting the user to the create page, which was proven to work already.

On the create page, the user would confirm the selection of the virtual machine displaying the right name. The tool passed all the tests for this section. After the user confirms the correct name of the VM the next step would be to press the ‘create’ button. The correct output would be redirecting the user to the execute page and opening the terraform file for the user. Again, every time tested this event the tool managed to complete it successfully.

.

```
resource "azurerm_resource_group" "rg_A4_v2" {
  name     = "rg_A4_v2_name"
  location = "uksouth"
}

resource "azurerm_virtual_network" "vir_A4_v2" {
  name                = "vir_A4_v2_name"
  resource_group_name = "rg_A4_v2_name"
  location            = "uksouth"
  address_space = [
    "10.0.0.0/16"
  ]
}

resource "azurerm_subnet" "sb_A4_v2" {
  name                 = "sb_A4_v2_name"
  resource_group_name  = "rg_A4_v2_name"
  virtual_network_name = "vir_A4_v2_name"
  address_prefixes = [
    "10.0.3.0/24"
  ]
}

resource "azurerm_network_interface" "ni_A4_v2" {
  name                = "ni_A4_v2_name"
  location            = "uksouth"
  resource_group_name = "rg_A4_v2_name"
  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.sb_A4_v2.id
    private_ip_address_allocation = "Dynamic"
  }
}

resource "azurerm_linux_virtual_machine" "A4_v2" {
  name                = "virtiualMachine"
  location            = "uksouth"
  size                = "Standard_A4_v2"
  resource_group_name = "rg_A4_v2_name"
  network_interface_ids = [
    azurerm_network_interface.ni_A4_v2.id
  ]
```

Figure 10 - Terraform Code Generated for an Azure Virtual Machine

Next, on the execute page, there are two actions that the user can execute, Plan and Apply. If the user would like to see the plan before applying the terraform code, it is possible, and every time this was tested with the tool it created the correct output.

```
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # azurerm_linux_virtual_machine.A4_v2 will be created
  + resource "azurerm_linux_virtual_machine" "A4_v2" {
      + admin_password                  = (sensitive value)
      + admin_username                  = "secret_admin"
      + allow_extension_operations      = true
      + computer_name                   = (known after apply)
      + disable_password_authentication = false
      + extensions_time_budget          = "PT1H30M"
      + id                              = (known after apply)
      + location                        = "uksouth"
      + max_bid_price                   = -1
      + name                            = "virtiualMachine"
      + network_interface_ids           = (known after apply)
      + patch_assessment_mode           = "ImageDefault"
      + patch_mode                      = "ImageDefault"
      + platform_fault_domain           = -1
      + priority                        = "Regular"
      + private_ip_address              = (known after apply)
      + private_ip_addresses            = (known after apply)
      + provision_vm_agent              = true
      + public_ip_address               = (known after apply)
      + public_ip_addresses             = (known after apply)
      + resource_group_name             = "rg_A4_v2_name"
      + size                            = "Standard_A4_v2"
      + virtual_machine_id              = (known after apply)

      + os_disk {
          + caching                   = "ReadWrite"
          + disk_size_gb              = (known after apply)
          + name                      = (known after apply)
          + storage_account_type      = "Standard_LRS"
          + write_accelerator_enabled = false
        }

      + source_image_reference {
          + offer     = "UbuntuServer"
```

Figure 11 - Plan Generated for the Creation of the Virtual Machine

When applied the in both cloud providers, azure most of the times would present the below error. It is not correlated with the terraform applied from the tool. It seems to be an internal error from the cloud provider that does not allow multiple resources to be created. Sometimes, testing the azure cloud provider, would create some of the resources but not able to create all of them.

```
Error: creating Subnet: (Name "sb_A4_v2_name" / Virtual Network Name "vir_A4_v2_name" / Resource Group "rg_A4_v2_name"): network.SubnetsClient#Cre
ateOrUpdate: Failure sending request: StatusCode=404 -- Original Error: Code="ResourceGroupNotFound" Message="Resource group 'rg_A4_v2_name' could n
ot be found."

  with azurerm_subnet.sb_A4_v2,
  on terraform.tf line 15, in resource "azurerm_subnet" "sb_A4_v2":
  15: resource "azurerm_subnet" "sb_A4_v2" {
```

Figure 12 - Error output from Azure VMs

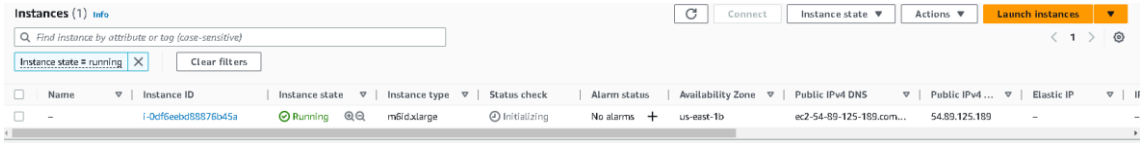

Figure 13 - Output AWS cloud provider

Otherwise, when tried to create on the amazon web services cloud provide, all the tests went succefully.

Chapter

7. Conclusion

1. Review of Aims

1. Develop a user-friendly tool that can be used by users with small or no expertise related to Terraform. The tool will contain a basic and intuitive interface.

Chapter 3 talks about this in more detail in the UI section. With help from Bootstrap and other tools like HTML and CSS, it was possible to achieve this aim, making the UI simple and efficient. This key point is important to assure that the user, as a result, has the best option for his requirements in his cloud provider. Otherwise, the UI could confuse the user and lead them to make mistakes

2. Compare virtual machines over diverse cloud providers, allowing users to select the requirements needed for their desired virtual machine.

Chapter 5 has information related to this goal. It’s one of the core aims of the project due to the fact that, without this part, the user would not be able to select the best option. In addition, the search method improves the flow and the usage within the tool. This allows for an optimized output time. The aim was achieved by developing a search tool that, no matter what requirements the user places on it, will present only the right options.

3. Generate the necessary Terraform code to construct a virtual machine on the user's preferred cloud provider. This will help users quickly and easily put together a VM without the necessity to learn coding or any technical competence.

Generating Terraform code is covered in Chapter 5 and was also achieved within this project. The Terraform code is generated and then displayed to the user. With this, the user can confirm what is being applied to their cloud provider account and, furthermore, if the user is experienced in Terraform, they can add any other desired resources.

4. Develop a detailed guide with all the necessary instructions for users to use the tool without any trouble.

A guide was created as a readme file so that every user knows what requirements their machine needs to perform the best result on this tool. This was covered in Chapter 3 on the usability part.

2. Limitations

Unfortunately, this tool has a couple of limitations. The first one is a region restriction, due to AWS only presenting the state of Virginia, USA. In relation to this, the “All providers page” is only able to search for this specific region.

The second limitation roots from Azure cloud provider, which, on occasion, does not have the capacity to produce all the resources to create a VM. Meaning that there is a restriction when creating Azure virtual machines.

3. Future Work

This project has room to grow, expand, and develop into a useful resource for both beginners and experts. The tool itself was created with the ability to be taken up and improved by anyone with enough coding knowledge. This project could become the foundation for other cloud providers, with the chance to solve its regional limitations or even create additional resources in the cloud

Bibliography

[1] IMB, "IMB," 2023. [Online]. Available: https://www.ibm.com/topics/virtualization. [Accessed 20 March 2023].

[2] Wikipedia, "Virtual machine," 12 March 2023. [Online]. Available: https://en.wikipedia.org/wiki/Virtual_machine. [Accessed 20 March 2023].

[3] Knowledge Sourcing Intelligence, "knowledge-sourcing," Nov 2022. [Online]. Available: https://www.knowledge-sourcing.com/report/global-desktop-virtualization-market. [Accessed 20 March 2023].

[4] Mordor Intelligence, "mordorintelligence," 2022. [Online]. Available: https://www.mordorintelligence.com/industry-reports/virtualization-software-market. [Accessed 20 March 2023].

[5] cloudflare, "cloudflare," 2023. [Online]. Available: https://www.cloudflare.com/learning/cloud/what-is-the-cloud/. [Accessed 20 March 2023].

[6] W. Chai, "techtarget," Oct 2022. [Online]. Available: https://www.techtarget.com/searchitchannel/definition/cloud-service-provider-cloud-provider. [Accessed 2023 March 20].

[7] A. Stashko, "avenga," 5 May 2021. [Online]. Available: https://www.avenga.com/magazine/top-cloud-service-providers/. [Accessed 20 March 2023].

[8] HashiCorp, "HashiCorp," 2023. [Online]. Available: https://developer.hashicorp.com/terraform/intro. [Accessed 20 March 2023].

[9] w3techs, "w3techs," [Online]. Available: https://w3techs.com/technologies/history_overview/client_side_language/all. [Accessed 20 March 2023].

[10] statista, "statista," July 2022. [Online]. Available: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. [Accessed 20 March 2023].

[11] mongodb, "mongodb," 2023. [Online]. Available: https://www.mongodb.com/what-is-mongodb. [Accessed 20 March 2023].

Appendix

1. Appendix 1 – Project Proposal