****

Automated Generation of VMIs with Hasicorp Packer

**Graduation Project**

**Automated Generation of Virtual Machine Images with Hasicorp Packer**

**Submitted By**

**Necati Hasan Turan 1101020051**

**Yaren Mısırlı 1600001547**

**Işıl Alıcı 1600002106**

Project Advisor

Prof.Dr. Murat Taylı

Department of Computer Engineering

İstanbul Kültür University

2020-2021 SPRING

****

May   
2021

**Graduation Project**

**Automated Generation of Virtual Machine Images with Hasicorp Packer**

**Submitted By**

**Necati Hasan Turan 1101020051**

**Yaren Mısırlı 1600001547**

**Işıl Alıcı 1600002106**

Advisor

Prof.Dr. Murat Taylı

THE EXAMINATION COMMITTEE

Jury Member Signature

1. Prof.Dr.Murat TAYLI ………………………..

2. Prof.Dr.Akhan AKBULUT ………………………..

3. Prof.Dr.Özgür Koray ŞAHİNGÖZ ………………………..

# ABSTRACT

Modern software development depends on the benefits of mechanization on development and operational stages, an engineering concept called as DevOps (development&operations).

In the age of cloud computing, many organizations and are now quitting monolith systems and turning their infrastructure to immutable infrastructure to simplify the stages develop-configure-deploy and its reliability through to concept of Infrastructure as Code. The general develop, configure and deploy stages are now is more complex than ever and needs to be maintained to be more time and cost effective. To create a configurated server and platform for our services and applications we use a virtual machine image on the platforms we develop and serve. A machine image is a virtual copy of an operating system that have a pre-baked operating system. This image must have installed software, patches, network configuration which is used to create a copy of running operating systems as machines for our desired purposes. Using pre-baked images provides effectivity and consistency and it reduces development times radically than any other techniques. However machine image types and their configurations change for each platform. That means we have to adapt every change and update for each platform we will used and serve our applications. With an immutable infrastructure, we create a new server each time, rather than making updates to a running server. The most efficient way to apply and use an immutable infrastructure is about making every parts of the system as a single code unit. Therefore we use the term Infrastructure as Code, containing all the necessary parts like virtual server images, specific alterations of the server. Creating all this platforms and their configurations is a very complex for developers and time and cost consuming stage. Therefore Hashicorp Packer propose a quick and decent way to solve this by creating pre-baked machine images with only json formatting file from any platform. Packer automates the creation of machine images of any kind and making it run remotely. It incorporates modern configuration management by encouraging you to use a framework like Chef or Puppet to install and configure software within your images created with Packer.

# ÖZET

Modern yazılım geliştirme DevOps olarak bilinen geliştirme ve operasyonel süreçlerin otomasyon halinde planlamasıyla oluşan mühendislik konseptine dayanmaktadır.

Bulut bilişim çağında, birçok kuruluş monolitik sistemleri bırakıyor ve geliştirme-yapılandırma-dağıtma aşamalarını Kod olarak Altyapı kavramına kadar güvenilirliğini basitleştirmek için altyapılarını ***değiştirilemez*** altyapıya dönüştürüyorlar. Genel geliştirme, yapılandırma ve dağıtma aşamaları artık her zamankinden daha karmaşık, daha fazla zaman ve maliyet gerektiriyor. Hizmetlerimiz ve uygulamalarımız için yapılandırılmış bir sunucu ve platform oluşturmak için geliştirdiğimiz ve hizmet verdiğimiz platformlarda ***sanal bir makine görüntüsü*** kullanıyoruz. Bir makine görüntüsü, önceden hazırlanmış ve ayarları tamamlanmış bir işletim sisteminin sanal bir kopyasıdır. Bu görüntü içerisinde, istediğimiz amaçlarla hızlı bir şekilde yeni çalışan makineler oluşturmak için kullanılan yazılımlar, yamalar, ağ yapılandırması yüklenmiş olması gerekir. Önceden pişirilmiş görüntülerin kullanılması, etkinlik ve tutarlılık sağlar ve geliştirme sürelerini diğer tekniklerden önemli ölçüde azaltır. Ancak makine görüntü türleri ve yapılandırmaları her platform için değişir. Bu, her değişikliği, yapılandırmayı ve güncellemeyi uygulamalarımızda hizmet vereceğimiz her platform için uyarlamamız gerektiği anlamına gelir. Değişmez bir altyapı ile çalışan bir sunucuda güncelleme yapmak yerine her seferinde yeni bir sunucu oluşturuyoruz. Değişmez bir altyapıyı uygulamanın ve kullanmanın en verimli yolu, sistemin her parçasını tek bir kod birimi yapmaktır. Bu nedenle, sanal sunucu görüntüleri, sunucunun belirli değişiklikleri gibi gerekli tüm parçaları içeren ***Kod olarak Altyapı*** terimini kullanıyoruz. Tüm bu platformları ve yapılandırmalarını oluşturmak, geliştiriciler için çok karmaşık ve zaman ve maliyet alıcı bir aşamadır. Bu sorunu çözebilmek için Değişmez Altyapı oluştururken Hashicorp Packer’ın tüm yeteneklerini kullanmayı amaçladık. Herhangi bir platformdan yalnızca JSON formatlama dosyasıyla önceden pişirilmiş makine görüntüleri oluşturarak bunu çözmenin hızlı ve iyi bir yolunu bu araştırmada uyguladık ve kanıtladık. Packer, her türden makine görüntüsünün oluşturulmasını otomatik hale getirir ve uzaktan çalışmasını sağlar. Packer ile oluşturulan resimlerinizde yazılımı kurmak ve yapılandırmak için Chef veya Puppet gibi bir çerçeve kullanmaya teşvik ederek modern konfigürasyon yönetimini içerir.

# ACKNOWLEDGEMENTS

Firstly, we would like to thank Istanbul Kultur University Computer Engineering department for their contributions and support through the years we educated and for the preparation of this project and report.

We would like thank all of the participants of the project for their valuable efforts.

Finally we would also like to express our great gratitudes and best wishes Prof. Dr. Murat TAYLI for his mentorship, experience, guidance and care through the project and making this project happened.

# TABLE OF CONTENTS

[ABSTRACT I](#_Toc73814607)

[ÖZET II](#_Toc73814608)

[ACKNOWLEDGEMENTS III](#_Toc73814609)

[TABLE OF CONTENTS IV](#_Toc73814610)

[LIST OF TABLES V](#_Toc73814611)

[LIST OF FIGURES VI](#_Toc73814612)

[SYMBOLS & ABBREVIATIONS VII](#_Toc73814613)

[1. INTRODUCTION 1](#_Toc73814614)

[1.1. Problem Statement 1](#_Toc73814615)

[1.2. Project Purpose 3](#_Toc73814616)

[1.3. Project Scope 3](#_Toc73814617)

[1.4. Objectives and Success Criteria of the Project 5](#_Toc73814618)

[1.5. Report Outline 5](#_Toc73814619)

[2. RELATED WORK 6](#_Toc73814620)

[2.1. Existing Systems 8](#_Toc73814621)

[2.2. Overall Problems of Existing Systems 15](#_Toc73814622)

[2.3. Comparison Between Existing and Proposed Method 16](#_Toc73814623)

[3. METHODOLOGY 17](#_Toc73814624)

[3.1. DesIgn OvervIew 17](#_Toc73814625)

[3.2. MODULE A 19](#_Toc73814626)

[3.3. System ArchItecture 19](#_Toc73814627)

[3.4. Module B 27](#_Toc73814628)

[1.1.1. Continuous Delivery 30](#_Toc73814629)

[1.1.2. Dev/Prod Parity 30](#_Toc73814630)

[1.1.3. Appliance/Demo Creation 31](#_Toc73814631)

[4. Experimental studies 50](#_Toc73814632)

[5. DISCUSSION 53](#_Toc73814633)

[6. Conclusions 54](#_Toc73814634)

[REFERENCES 55](#_Toc73814635)

# LIST OF TABLES

[Table 2.1: Comparison of methods 16](#_Toc73263910)

# LIST OF FIGURES

Figure 1: 2

Figure 2: ……………………………………………………………………………………………...2

Figure 3: 3

Figure 4: 6

Figure 5: 6

Figure 6: 7

Figure 7: 8

Figure 8: 10

Figure 9: 13

Figure 10: 18

Figure 11: 20

Figure 12: 21

Figure 13: 21

Figure 14: 22

Figure 15: 22

Figure 16: 23

Figure 17: 23

Figure 18: 24

Figure 19: 24

Figure 21: 25

Figure 22: 26

Figure 23: 27

Figure 24: 37

Figure 25: 38

Figure 26: 38

Figure 27: 39

Figure 28: 39

Figure 29: 39

Figure 30: 40

Figure 31: 41

Figure 32: 41

Figure 33: 42

Figure 34: 43

Figure 35: 44

Figure 36: 45

Figure 37: 45

Figure 38: 46

Figure 39: 47

Figure 40: 48

Figure 41: 49

Figure 42: 50

Figure 43: 50

Figure 44: 51

### SYMBOLS & ABBREVIATIONS

VMI: Virtual Machine Image

JSON : JavaScript Object Notation

VPC : Virtual private cloud

IaC : Infrastructure as a Code.

IaaS : Infrastructure as a Service

PaaS : Platform as a Service

AWS : Amazon Web Services

AMI : Amazon Machine Image

SUDO : Super User Do

Ec2 : Elastic Cloud Compute

IEEE: Institute of Electrical and Electronics Engineer

HCL : Hashicorp Configuration Language

GAN:Generative adversarial networks

NAS:Network attached storage

API: Application Programming Interface

GPIO: General Purpose Input/Output RAM:Random access memory

SSH: Secure Shell OS:Operating system

K8S : Kubernetes  
CaaS : Container as a Service.

ASNaC :Application-Specific Network-as-code

CD : Continuous Delivery

CI : Continuous Integration

AMAZON EBS : Amazon Elastic Block Store

VPCs : Virtual Private Clouds

# INTRODUCTION

Contemporary notions of the IT industry evolved by the changes on cloud computing. Companies, and us as developers are observing fundamental changes by the emergence of cloud computing and its own changes on software development cycles.

To make our points clear, first we must understand how many basic stages are there of cloud computing.

There are broadly known three layers of clouds;

1. Infrastructure as a Service -IaaS-,
2. Software as a Service -SaaS-,
3. Platform as a Service -PaaS-

To be clear about understanding of the Among these three layers, IaaS systems (e.g., AWS,Azure,GCloud,DigitalOcean,Linode) offer fundamental infrastructure configurations and tools like VMs and virtual storages, and these have got the most recognition from the IT market. [4].

Either we focus SaaS or IaaS, we must organize and automate our works and applications on the servers, especially on the cloud based organizations.

Therefore updating, managing, creating basically configurating the whole infrastructure means a lot of hardwork and that means more money and time to consume.

To achieve effectivity on time, budget and human sources we focused on Immutable Infrastructure services to achieve managing VMIs and systems by using Hashicorp Packer to automate and clarify all these steps of IaC and we will examine all these layers and their problems in this paper.

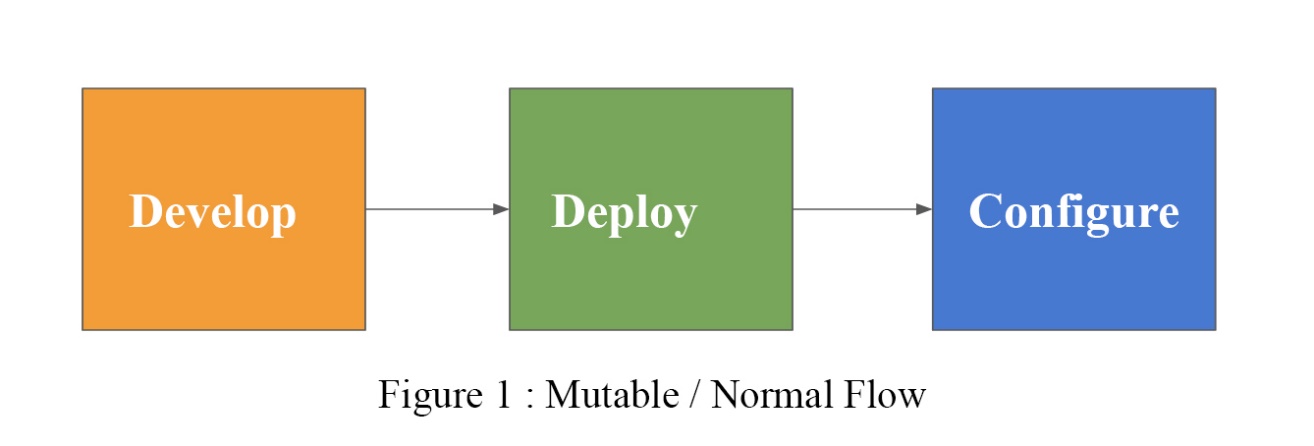
## Problem Statement

When we try to deploy a series of apps, configurations and services for a certain aim on an instance of a virtual image, it will take longer time to make it ready for work and do exact setting of our desired purpose.

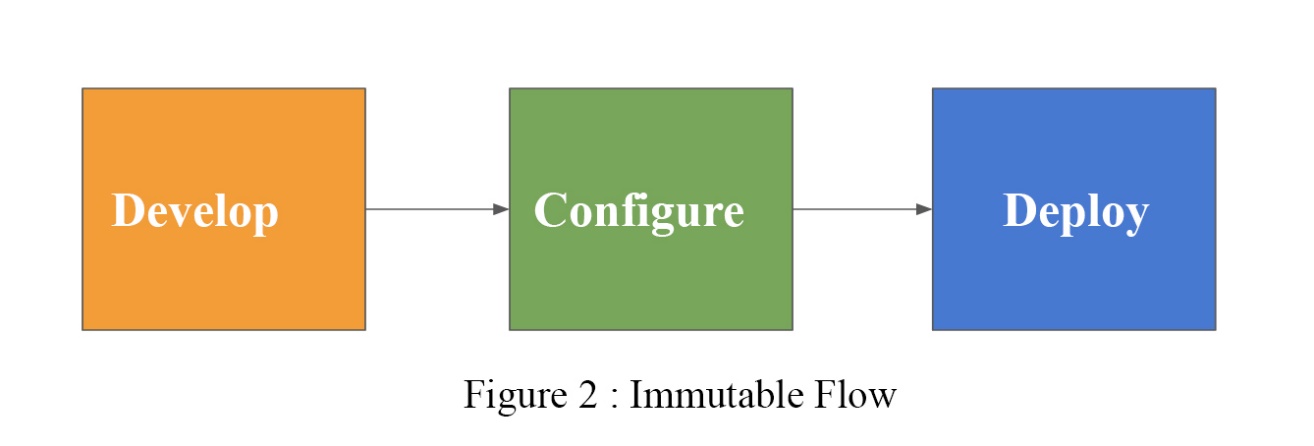
We, developers always want to automate the system so DevOps teams can easily maintain and alter each time we need to be interact with our servers.[5]

On DevOps[1]; software codes emerging as an automation is called as Infrastructure-as-Code (IaC). Infrastructure-as-Code means; promoting managing the information and late works inside re-usable scripts of IaC rather than old-school reserving it for the vast amount of work for developers, which makes the process quite slow, time and money wasting, and generally open to make errors [2].

Traditional mutable server infrastructure needs to be continually updated and altered for each configurations. That means developers must work with this the infrastructure and make secure shell into their servers -either upgrade or downgrade- packages one by one manually, change configuration files on a server by working for each (an infrustructure can have hundereds of servers) and deploy new set of code onto existing servers. Whole these configurations means that this servers servers are changeable (mutable) so they can be altered after they’re created. [7]

8

However we need to eliminate this process and make autonomous. Therefore we resort to the Immutable Infrustructures which means kill the server if you want to make a change. To reduce the time for creating and configuring machine images we need to develop complex infrastructure and modify each time whenever we needs an update.



We used Immutable Infrustructure There it’s a problem not to have a tool like Packer.

## Project Purpose

This project aims to show every capability of Packer by creating and launching machine images on AWS EC2 platform while showing how long it can take without packer.

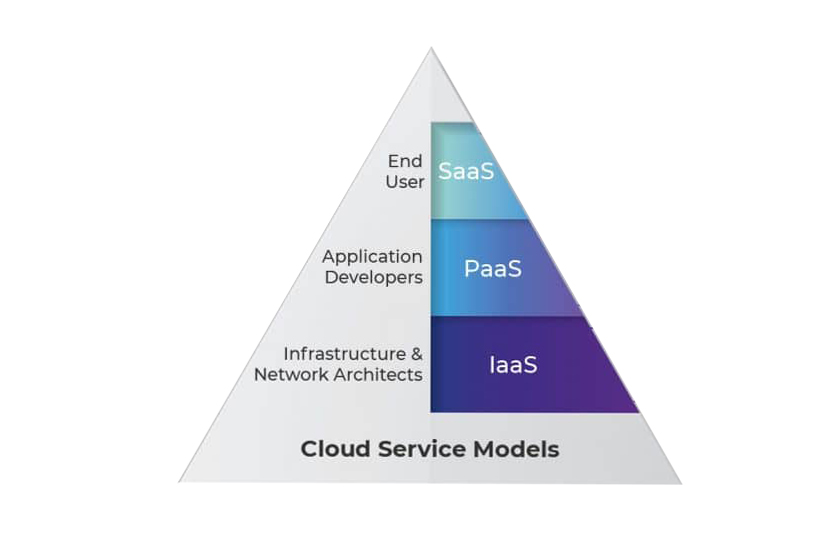
While we show other techniques and their capacities, we will also show how easy it could with only using Packer as a helping tool to organize whole infrastructure for our services and applications.

We used Immutable Infrustructure[7] to prove our point we compare with other concepts like Mutable Infrustructure or monolith systems.

## Project Scope

DevOps is an collective&extensive concept which reduces the blockage between development teams and operational staff to make a continuous and fast delivery and allow precise&fast reactions to evolving necessities in the infrastructure development cycles.[3]

Developing applications, running them and their applications needs to be established as IaaS,PaaS or as Monolithic traditional service.

f figure 3 :Cloud Service Models

Cloud providers offers a different kind of infrastructure for each purposes, the meaning of an infrastructure component (e.g., a virtual machine) implies creating a small amount of code that only focus on the aimed provider as cloud service prodivder.[10]

In this project, an automated machine images generation will be implemented by using Hashicorp Packer. We focus on the cloud concept as IaC. IaC approach supports code-oriented tools that use only scripts to specify configuring, creating, updating and finally executing the cloud infrastructure resources (e.g. VMIs) and their appliances (e.g. 3rd party applications, network configurations). [4][2]

We will discuss why IaC concept is necessary, what else we have, what can we do about solving popular industrial problems and finally we will show how we can achieve all of this with only Packer.

Project will show each parts&components of Packer and their applications on different operating systems and cloud based services[5].

## Objectives and Success Criteria of the Project

The primary goal of the success of the project is to create a problemless generation of automated instances of virtual images and needed configurations in it. After the desired configurations and alterations is achieved on the our instance, the goal is to deal with in the project will be to have creation process only by a JSON file. The manual creation of the VMIs will be saved and showed in the project first.[6]

This recording will be showed on the AWS EC2 platforms. Then, with the Packer’s abilities and components we will look at how we can do all of these manual configuration with an only a JSON file remotely. We don’t even have be on the EC2 console or any platform we want to create our instances. Thus, the process of the generation of the automated virtual images will be done by our simple code blocks with the open source tool, Hashicorp Packer.

## Report Outline

First, we will talk about virtual images, immutable and mutable infrustructures and then we will talk about types of virtual images. Then we will give a basic report about Amazon AWS ecosystem. Then we will talk about Hashicorp ecosystem. Finally we will take look at Packer, it’s capabilities and components. In the methodology section, all parts of the our whole work and the Packer itself will be completely explained and showed. On the part of 5, the project with similar works has been examined and the evaluations are presented in a table.

In the experimental results section, the studies and observations made to achieve the targeted results are included. The data obtained as a result of these experiments are presented below with figures and tables.[16] In the discussion part, the procedures and findings obtained from the experiments were evaluated and the ideas were given. In the conclusion part, a short description of the project and future ideas are given.

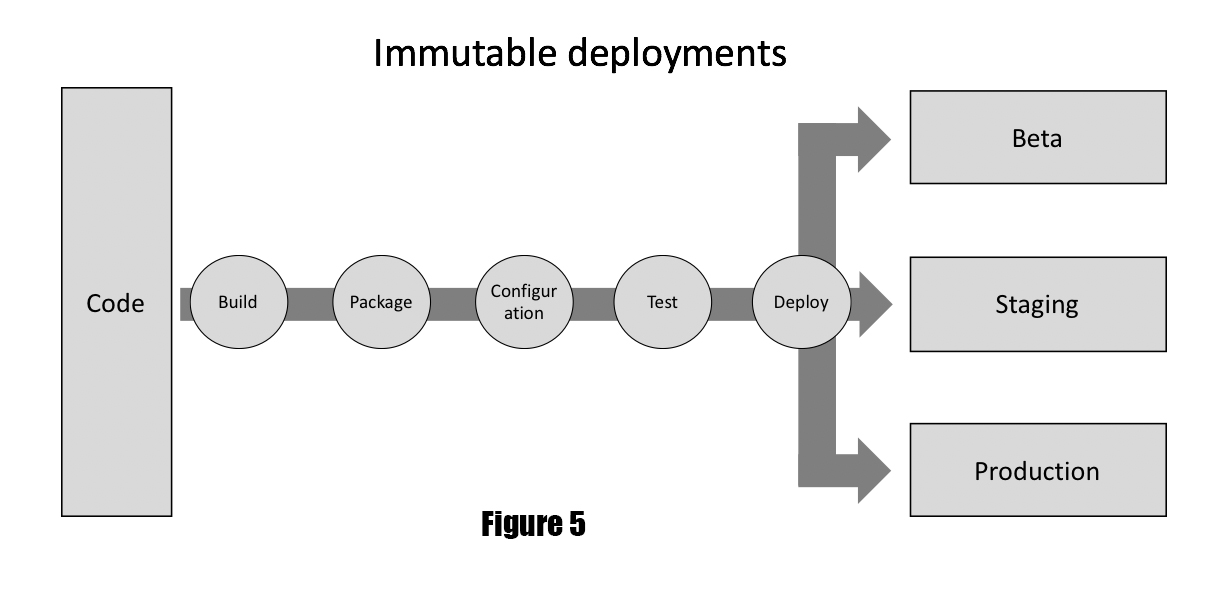
# RELATED WORK

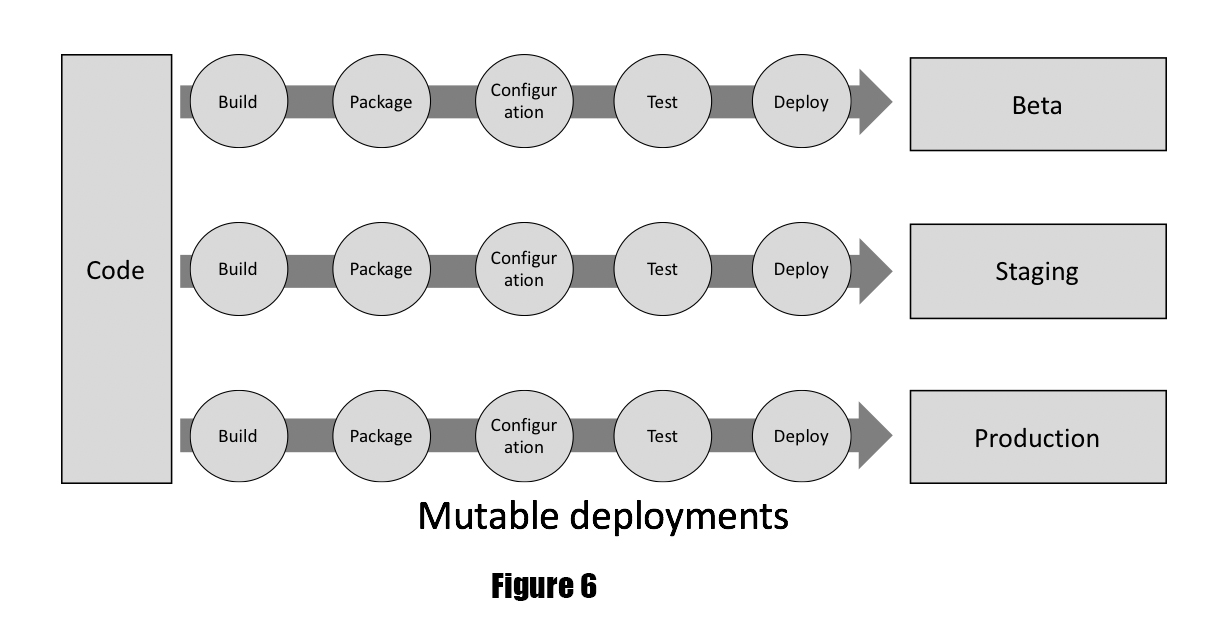
In this part we will examine alternatives and competitors of Hashicorp Packer and we will show a basic benchmark among others.

Matej Artac & Damian Andrew Tamburri [11], compiles different fields on Infrustructure As Code principle with numerous techniques. Infrastructure design and development is about the application development lifecycle stages that examines and set the basic principle of infrastructure requirements for that software such as Amazon cloud, Azure and the other numerous kind of VMIs needed in each fields.[23]

The infrastructure pattern typically require numerous installation and settings of the scripts needed to be applied can be described in 3 parts. (i) create and set the required machines (either physical in a local system or virtual in a cloud based system) for the application itself to be applied; (ii) deploy and make the settings of the the needed software and its own configurations for those VMs; (iii) finally apply and run the required secondary services or the third party applications for the software or services to be operated on a system.

We will discuss and deeply examine Vagrant, Ansible, Terraform, Docker, Kubernetes,Cloudify , Terraform and their basic comparisons.

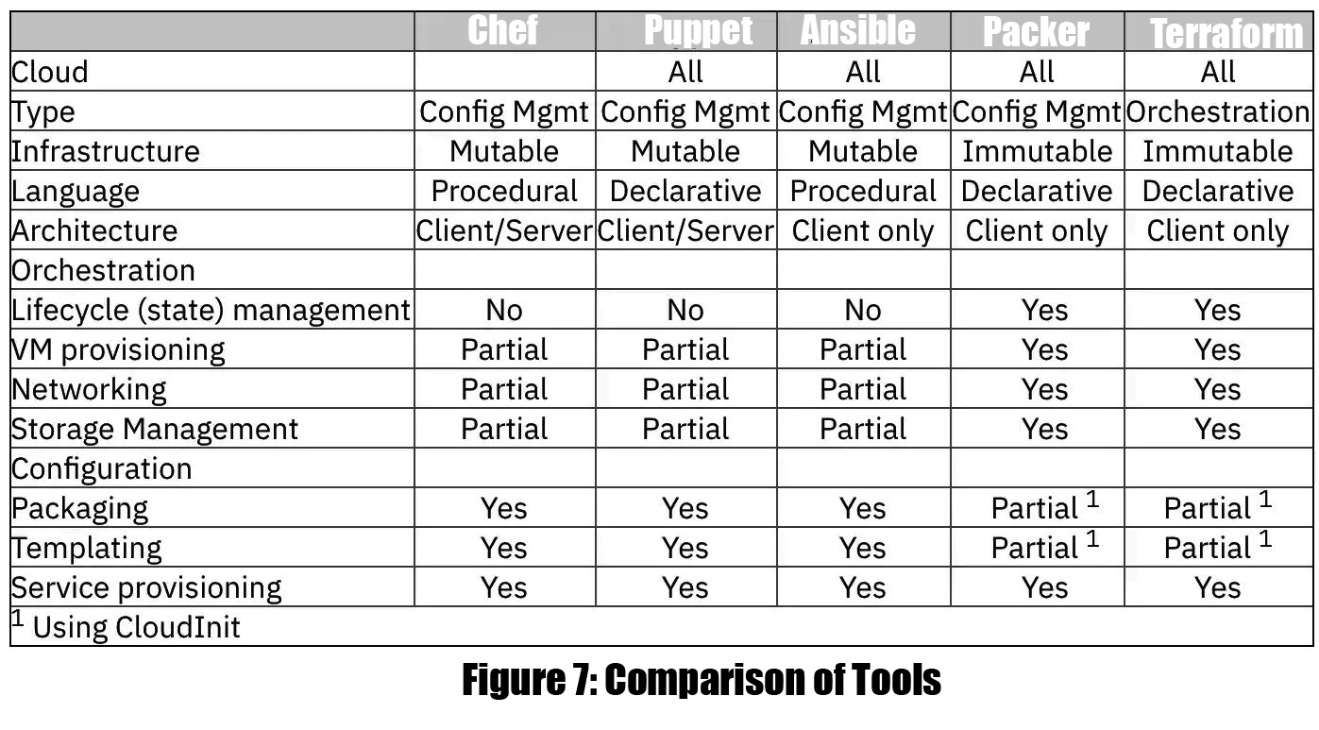
As we mentioned earlier, there are 2 basic types of infrastructures; ***mutable*** and ***immutable***.



Mutable infrastructure uses tools such as ***Chef, Puppet, Ansible*** that automatically updates application on the server by creating a long history of changes and those are very hard to reproduce while quite difficult to examine for any errors occurring during the deployment and configuration stages.

Generally, in mutable infrastructure DevOps and staff encounter the situation where someone updated the server leading to the low quality production server and that making to downtime or unknown outcomes in the infrustructures.[22]

Here is a figure make readers clear what all of this tools has as disadvantages and advantages.



## Existing Systems

As we mentioned earlier, there are 2 basic types of infrastructures; mutable and immutable.

Mutable infrastructure uses tools such as ***Chef, Puppet, Ansible*** that automatically updates application on the server by creating a long history of changes and those are very hard to reproduce while quite difficult to examine for any errors occurring during the deployment and configuration stages.

Generally, in mutable infrastructure DevOps staff encounter the situation where someone updated the server leading to the low quality production server and that making to downtime or unknown outcomes in the infrastructures.

Therefore immutable infrastructure is better for consistency.

#### 2.1.1.Mutable

#### Ansible

Ansible is an open source tool sponsored by Red Hat, and formerly it was the simplest way to automate a server.

It’s incredibly easy to use and its learning curve is quite small. However it’s not the best for the immutable infrastructures and we need other tools to use in *immutable* systems.

Besides its disadvantages, Ansible is quite small,fast and easy to implement.

It generates a VM by only a playbook file.(YAML)

Here is a basic installation of the Ansible on Linux :

|  |
| --- |
| $ sudo apt-get upgrade  $ sudo apt-get update $ sudo apt-get install software-properties-common  $ sudo apt-get check  $ sudo apt-add-repository --yes --update ppa:ansible/ansible  $ sudo apt-get install ansible |

Each playbooks has plays and only have inventories and variables.

Basic components of ansible :

* Modules
* Module utilities
* Plugins
* Inventory
* Playbooks
* The Ansible search path
* Modules

In the example below, the first ***play*** aims the web servers; the second play targets the database servers:

|  |
| --- |
| **-** name**:** Update the servers  hosts**:** webservers  remote\_user**:** root  tasks**:**  **-** name**:** Making it sure apache is at the latest ver.  ansible.builtin.yum**:**  name**:** httpd  state**:** latest  **-** name**:** Write an apache configuration file  ansible.builtin.template**:**  src**:** /srv/httpd.j2  dest**:** /etc/httpd.conf  **-** name**:** Update database server  hosts**:** databases  remote\_user**:** root  tasks**:**  **-** name**:** Ensure postgresql is at the current ver.  ansible.builtin.yum**:**  name**:** postgresql  state**:** latest  **-** name**:** Ensure that postgresql is started  ansible.builtin.service**:**  name**:** postgresql  state**:** started |

Its syntax is easy and packages are pretty small. Relatively faster than Chef and Puppet.

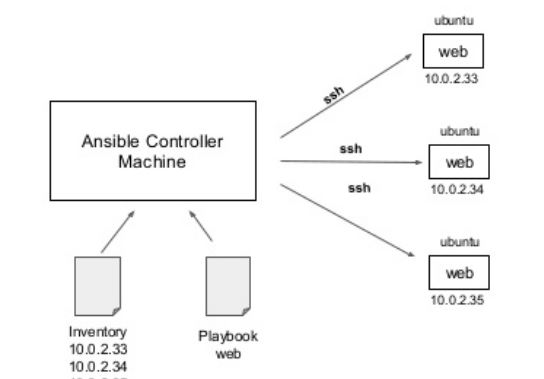


Figure 8 : Ansible’s Basic Structure

#### Chef

Chef works as a *settings management* tool on mutable servers. It serves a fast delivery and according to benchmark tests, it’s faster than Ansible on AWS [6].

Chef is a config management helper tool and the packer is a provisioning tool.

Chef is concerned with installation and management of software on existing servers, while packer provisions the servers themselves. Therefore when using Terraform or Packer, packer is a lot better option than a configuration management tool itself.

Moreover Chef reneges to a mutable infrastructure design and making it hard to examine configuration problems, but packer checks every update as a deployment of a new server configuration.

Chef uses an imperative language.

While packer don't have always to be a master role, Chef instantiate a procedural language and needs to be as master server for running state storing also for the agent application on every server.

Chef has three crucial components; Masters, The Chef Server and the Worker Nodes.

Chef works on vagrantfiles and can be accessible with Vagrantbox’s.

Here is a basic implementation of Chef as vagrantfile :

|  |
| --- |
| cat > Vagrantfile <<'EOH'  CFG\_BOX = 'bento/ubuntu-18.04'  CFG\_IP = '192.168.33.199'  CFG\_HOSTNAME = 'chef-automate.test'  $deployscript = <<-SCRIPT  export DEBIAN\_FRONTEND='noninteractive'  apt-get update  apt-get install -y --no-install-recommends curl unzip  apt-get clean  sysctl -w vm.max\_map\_count=262144  sysctl -w vm.dirty\_expire\_centisecs=20000  echo "${CFG\_IP} ${CFG\_HOSTNAME}" | tee -a /etc/hosts  curl -fsSL https://packages.chef.io/files/current/automate/latest/chef\_linux\_amd64.zip -o /tmp/chef\_linux\_amd64.zip  unzip -qod /usr/local/bin /tmp/chef\_linux\_amd64.zip  chmod +x /usr/local/bin/chef-automate  chef-automate deploy --accept-terms-**and**-mlsa  echo "Server is up and running. Please log in at https://${CFG\_HOSTNAME}/"  echo 'You may log in using credentials provided below:'  cat /home/vagrant/automate-credentials.toml  SCRIPT  Vagrant.configure(2) **do** |config|  **class** **AcceptLicense**  **def** to\_s  **return** 'true' **if** ENV['ACCEPT\_CHEF\_TERMS\_AND\_MLSA'] == 'true'  puts <<TERMS  To continue, you'll need to accept our terms of service:  Terms of Service  https://www.chef.io/terms-of-service  Master License and Services Agreement  https://www.chef.io/online-master-agreement  I agree to the Terms of Service(y/n)  TERMS  if STDIN.gets.chomp == 'y'  'true'  else  puts 'Chef Software Terms of Service **and** Master License **and** Services Agreement were **not** accepted'  exit 1  end  end  end  config.vm.box = CFG\_BOX  config.vm.hostname = CFG\_HOSTNAME  config.vm.provider "virtualbox" do |v|  v.name = 'chef-automate'  v.memory = 4096  v.cpus = 2  v.customize ['modifyvm', :id, '--audio', 'none']  end  config.vm.synced\_folder '.', '/opt/a2-testing', create: true  config.vm.network 'private\_network', ip: CFG\_IP  config.vm.provision 'shell', env: {'CFG\_IP' => CFG\_IP,  'CFG\_HOSTNAME' => CFG\_HOSTNAME,  'ACCEPT\_CHEF\_TERMS\_AND\_MLSA' => AcceptLicense.new}, inline: $deployscript  **end**  EOH |

#### Puppet

Puppet is a tool accesible in both open source and as paid version and it has its own language as interpreter.

However to use in extended versions, we must pay so it has a disadvantage to Packer, since Packer is completely open source.

Unlike chef, Puppet adapts a declarative,model based language that is similar to JSON. You describe the resource’s state but you cannot interact in how this state is actually done

With Puppet, we create *manifests* and modules, while in Chef we create *recipes* and *cookbooks*.

Puppet is a configuration management tool too and its syntax is differentiated from the underlying OS and its applications' syntax. This leads you to define high-level concepts such as user, application, and service.

Puppet adapts every needs as data. Node's current state and the node’s targeted end-state, and the steps needed to step from one-by-one.

However since Puppet is not very useful on immutable systems, we must eliminate this as an option too.

#### 2.1.2.Immutable

#### Terraform

As Packer, Terraform also produced by Hashicorp as open source provisioning tool.

Terraform is an open-source IaSC tool that provides a consistent CLI automation workflow to manage multiple cloud services altogether. Since it convey IaC mentality, Terraform codifies cloud APIs into declarative configuration parts and files. It can work with Packer or alone.

In contrast with Packer, Terraform uses HashiCorp Configuration Language (HCL)

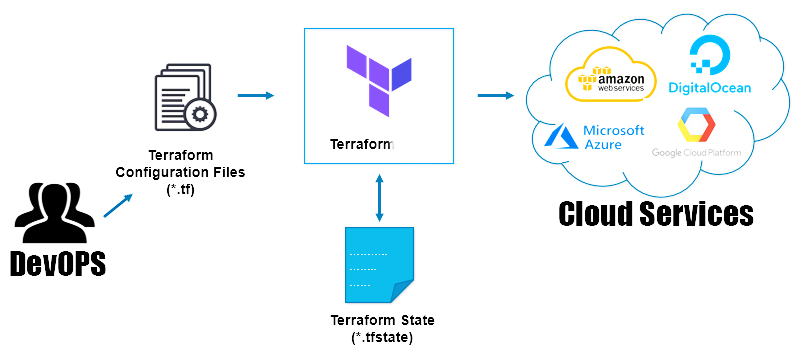


Figure 9 : Basic Structure of Terraform

Terraform is better than Ansible,Puppet and Chef when it comes to configuration because it also has a feature to manage orchestration.[5]

Terraform uses its own file format while Packer can easily worked with only JSON file.

Basic code structure of a Terraform unit ;

|  |
| --- |
| └── tf/  ├── modules/  │ ├── network/  │ │ ├── main.tf  │ │ ├── dns.tf  │ │ ├── outputs.tf  │ │ └── variables.tf  │ └── spaces/  │ ├── main.tf  │ ├── outputs.tf  │ └── variables.tf  └── applications/  ├── backend-app/  │ ├── env/  │ │ ├── dev.tfvars  │ │ ├── staging.tfvars  │ │ ├── qa.tfvars  │ │ └── production.tfvars  │ └── main.tf  └── frontend-app/  ├── env/  │ ├── dev.tfvars  │ ├── staging.tfvars  │ ├── qa.tfvars  │ └── production.tfvars  └── main.tf |

## Overall Problems of Existing Systems

While HashiCorp Packer automates the creation of *any* kind of VMs and their configuration, other tools not yet covering all systemic settings. Packer embraces modern configuration management, orchestration in the understanding of Immutable Infrastructure&Mutable Infrustructure by encouraging DevOps teams to use automated scripts as pre-baked templates and encouraging to deploy and configure the applications within your pre-baked (Packer-made) VMIs on any platform.

Packer brings VMIs into the contemporary age, release the potentials of cloud systems opening new opportunities like traffic enhancing.

Packer only use a simple JSON file to achieve all its promises while Terraform uses its own format.

While others don’t have these abilities, Packer does all with its growing communities as an open source tool.

Provisioning, configuring is the easiest since its learning curve is amongst the best.[9]

Packer is free, small, platform independent and no need to be master kernel needs to manipulate a deployment on any SaaC.

## Comparison Between Existing and Proposed Method

Table 2.1: Comparison of methods

| Packer | Terraform | Ansible | Puppet | Chef |
| --- | --- | --- | --- | --- |
| Can Work Alone | Can Work Alone | Cannot | Cannot | Cannot |
| Cross platform builds | Cross platform builds | Cross platform builds | Cross platform builds | Cross platform builds |
| Provisioning & Orchestration | Provisioning & Orchestration | Config Mngmnt | Config Mngmnt | Config Mngmnt |

Table 2.2: Comparison of methods

|  | Packer | Terraform | Ansible | Puppet | Chef |
| --- | --- | --- | --- | --- | --- |
| Source | Open | Open | Open | Open | Open |
| Cloud | All | All | All | All | All |
| Infrastructure | Immutable | Immutable | Mutable | Mutable | Mutable |
| Language | Declarative | Declarative | Procedural | Declarative | Procedural |
| Agent | No | No | Yes | No | No |
| Master | No | No | Yes | No | No |
| Community | Large | Large | Large | Large | Large |
| Maturity | High | High | High | High | High |

Note that, Table and Figure numbers start with the Section number and continue with the unique float identifier. All floats (Figures, Tables, Pseudocodes, Code Listings, etc.) should be referred in the text and if the float came from a different resource, you must provide a complete citation (e.g. Reprinted from [3]).

# [METHODOLOGY](#_Toc470871184)

In this section we will show manual manipulation of the servers, setting cloud based VMIs and their configurations while we also show how easy it will be with Hashicorp Packer.

Packer is an extremely small and fast tool and it has a large community it as built in CI/CD pipeline and configuration management.

Within Packer’s abilities to reach in cloud systems, it’s easy to implement system independent construction, we choose EC2 to prove our points.

Therefore we must understand the Amazon Web Services’ EC2 console and its applications first.[23]

## DesIgn OvervIew

Contemporary cloud services like AWS have built-in pre-baked VMs to serve customers desires.

Preloading required softwares,patches,settings and configurations on an Amazon EC2 VMI copy, then generating an image of that takes a couple of minute for each time.

After all, the resulting image might be used to create new copies (instances) with all software and configuration previously loaded. This process permits EC2 instance to be on-line and be ready for serve in a quick way. [21]

It’s not only make it simple the deployment of new copies but also mainly handy when an instance is member of an Auto Scaling group and is it’s reacting to a bug in load of the system.

However, if the instance creation&configuration takes longer time than it needed to be then it loses its advantage on the aim of dynamic scaling in the cloud & service.

To this purpose we used Packer to pre-bake AMIs without interfering EC2 console.

For this purpose; let’s take a quick look on AWS and EC2 console.

Steps of the implementations on AWS;

1. Set up your environment
2. Getting security Details
3. Choosing platforms and getting Pair keys as PEM file
4. Configuring IAM credentials
5. Running a Packer automation workflow

For this purpose, let’s give a quick view on EC2 with figures and tables.

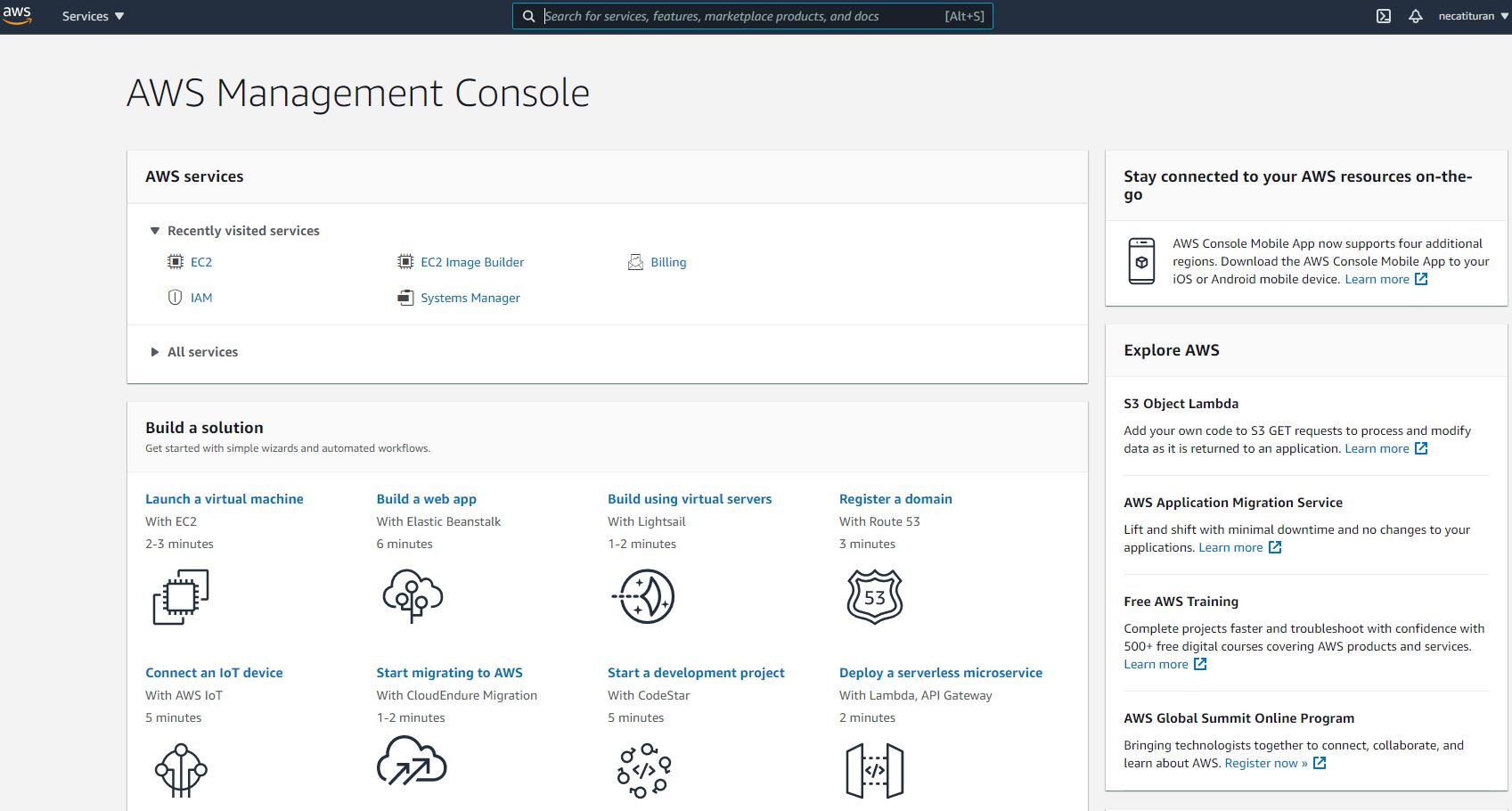


Figure 10 : AWS Management CONSOLE

#### What is Amazon EC2?

Amazon Elastic Compute Cloud (Amazon EC2) give customers a expansible integrated computing service in the AWS Cloud ecosystem.

Using Amazon EC2 serve your obligations to fund in physical hardware supplies so we can develop and deploy applications faster without needing a physical equipment.

We can use AWS to create and launch as many as virtual systems as we need and we can configure entails with security, networking, storage management.

## MODULE A

#### Details On EC2

To make our points clear, we want to make readers understand why we choose Amazon Web Services.

Therefore let us line up some important features of EC2 that we will use on this paper.

* VMIs, known as instances that implies virtual computing&configurating environments on cloud
* Pre-baked VMI and configuration templates for our copies (instances), known as AMIs, that whole collections, the requirements for our need,
* Numerous settings of CPU, data storages, memories and networking volume for our copies, called as instance types
* Secure login confirmations for our copies using key pairs so we can secure our services
* Storage capacities for temp data (deleted as need to stop/hibernate/terminate our copies)
* Persevering storage capacities for our data using Amazon EBS
* Numerous physical locations for our resources (Regions and Availability Zones)
* A firewall that make us to clarify the protocols, ports, IP etc.
* Static IPv4 addresses for the purpose of dynamic cloud computing, called as Elastic IPs.
* Virtual networks (VPCs) we can make for our instances that are logically isolated from the rest of the cloud services.

## System ArchItecture

As we mention earlier, we first need to show setting a new instance, a new virtual machine on a cloud service like AWS manually. For this purpose, let us show the steps and calculate how much time it will need to be to configure&set up a new instance to a hundreds of VMIs if we needed.

To create an EC2 instance;

* Sign in to the Amazon and then go to the AWS Console.
* Go through EC2 console by choosing EC2 under Compute.

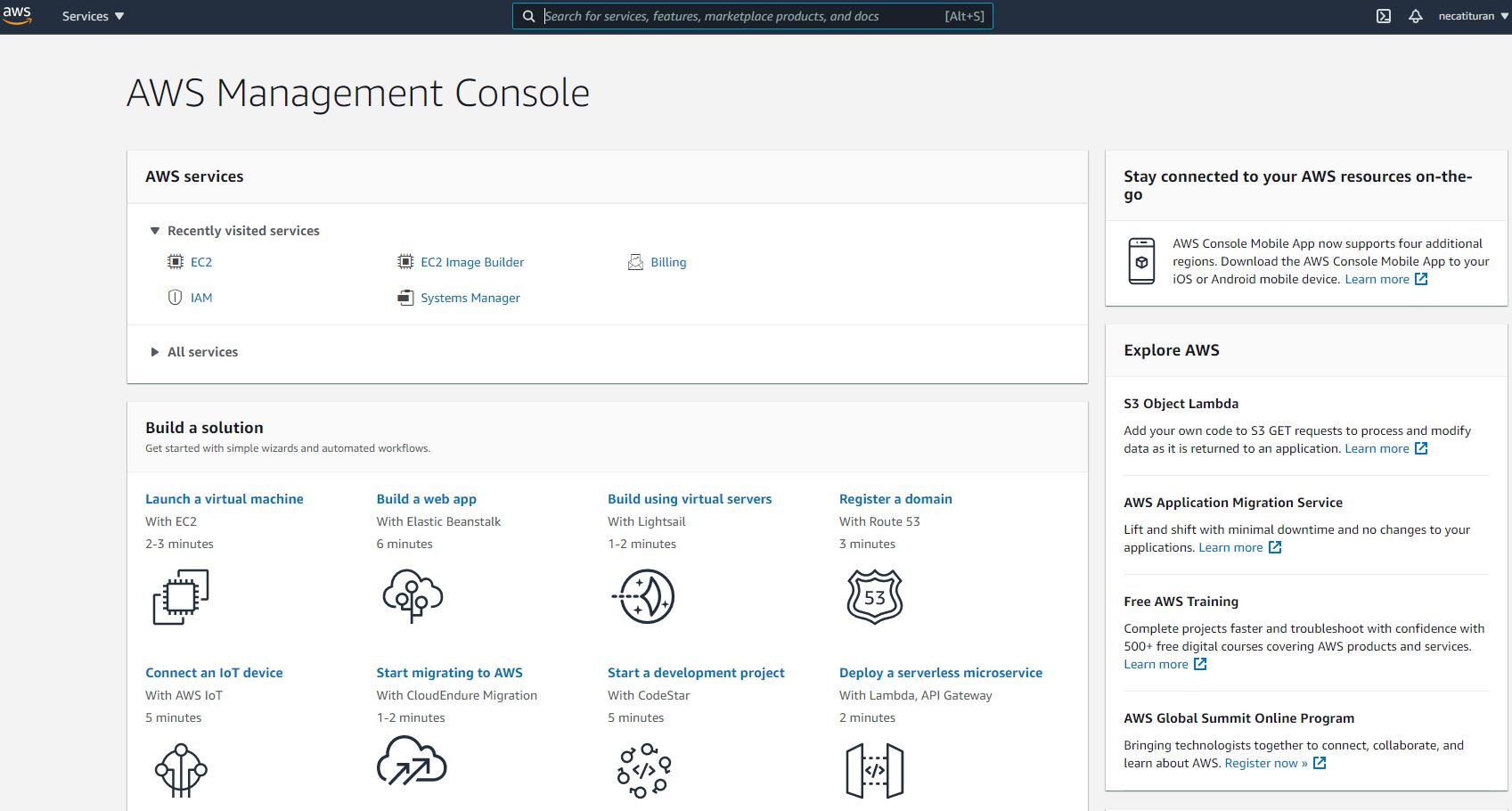


Figure 11 : General view of AWS Management Console

Choose all services;

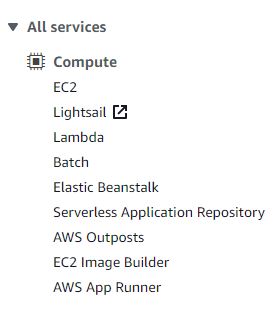


Figure 12 : Services

If you are using the Show All Services view, your screen looks like this:

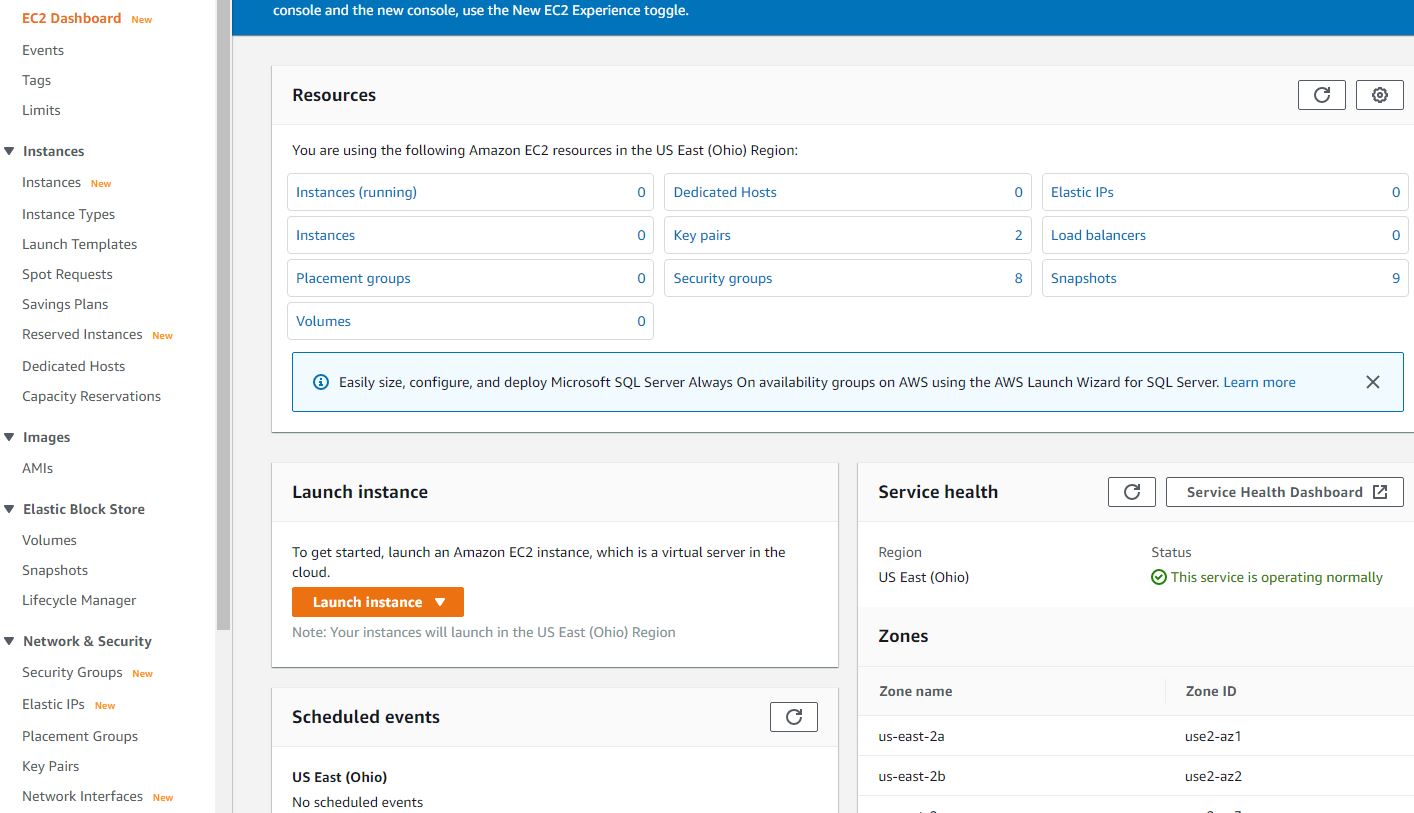


Figure 12 : EC2 Panel

There will be panel for;

* Instances
* Images
* Elastic Block Store
* Network&Security

Let’s first focus on AMIs. We previously created all these AMIs with Packer only.

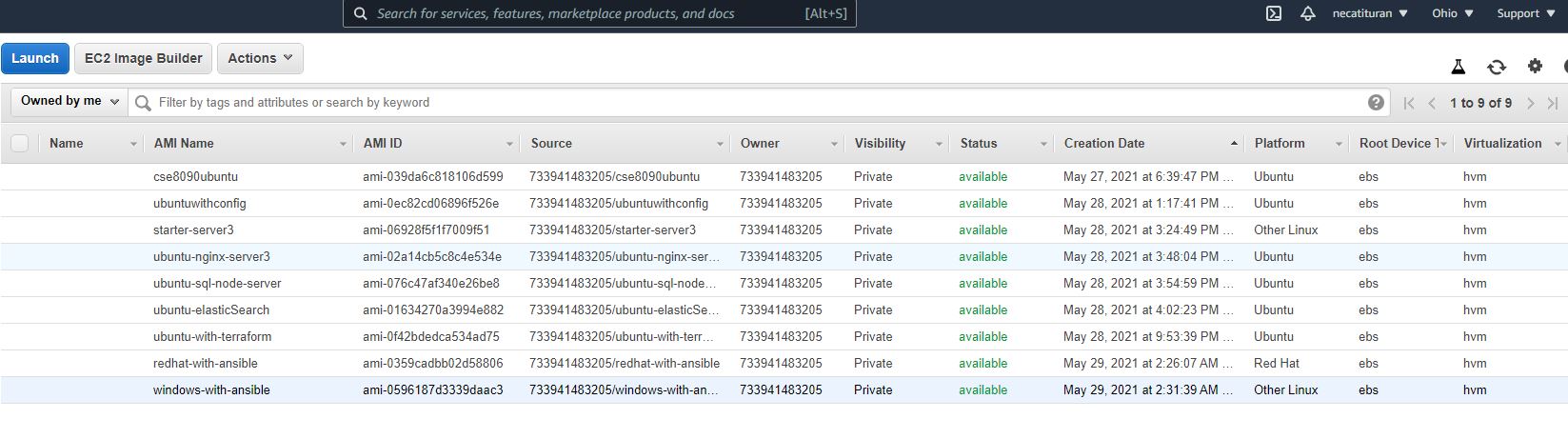


Figure 13 : AMIs Created By Packer

However this time it will be created manually by only using EC2 options. There will be 7 steps of AMI creation.

To make all configuration and set-ups done let’s follow these steps carefully.

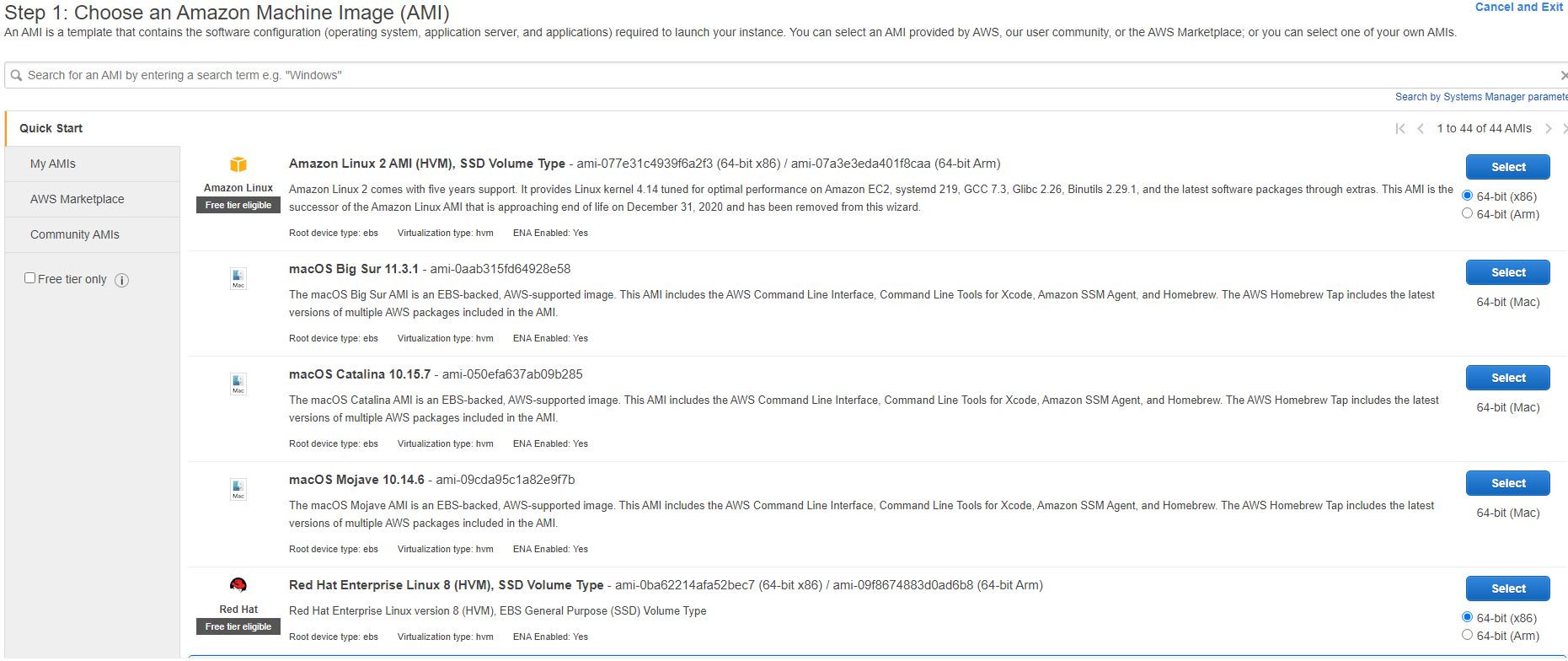


Figure 14 : First step of AMI Creation Manually

The AMI ID will be needed for creating instances of AMIs with Packer. However since it’s now manual creation let’s dive into Select option and choose an Amazon Linux AMI first.

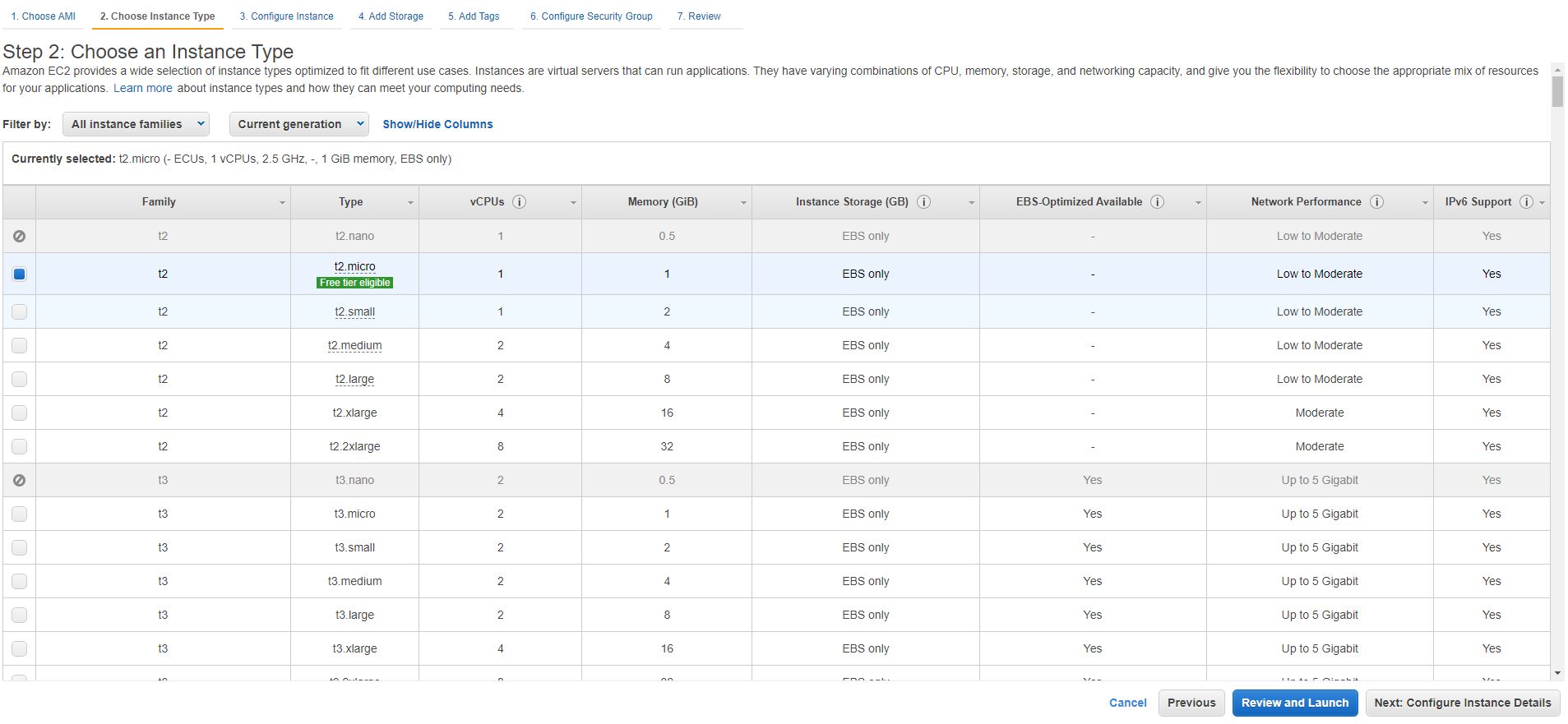


Figure 15 : AMI Type

Instance type is important to companies&services needs. This time we will continue choosing with t2.micro AMI instance.

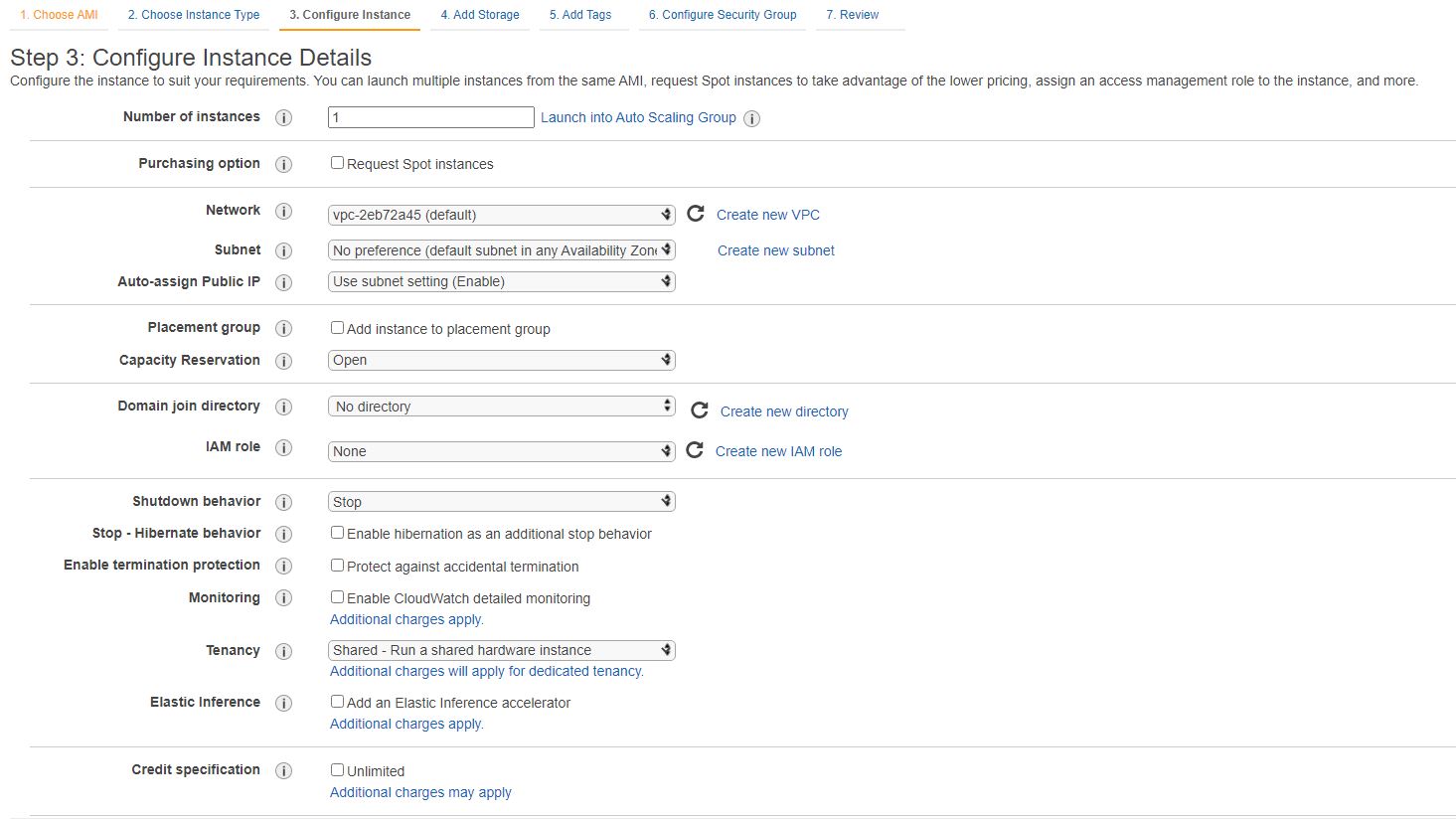


Figure 16 : Configuration Details on AMIs

For configuration we have many options. Some configurations as Availability Zone and Scaling Type, are determined when you create the instance and cannot be changeable later.[9]

Hibernations, Network types, Subents, Public IP specifications, IAM role and many of them are modifiable here but it can take too much time if we need to make it all manual.[17]

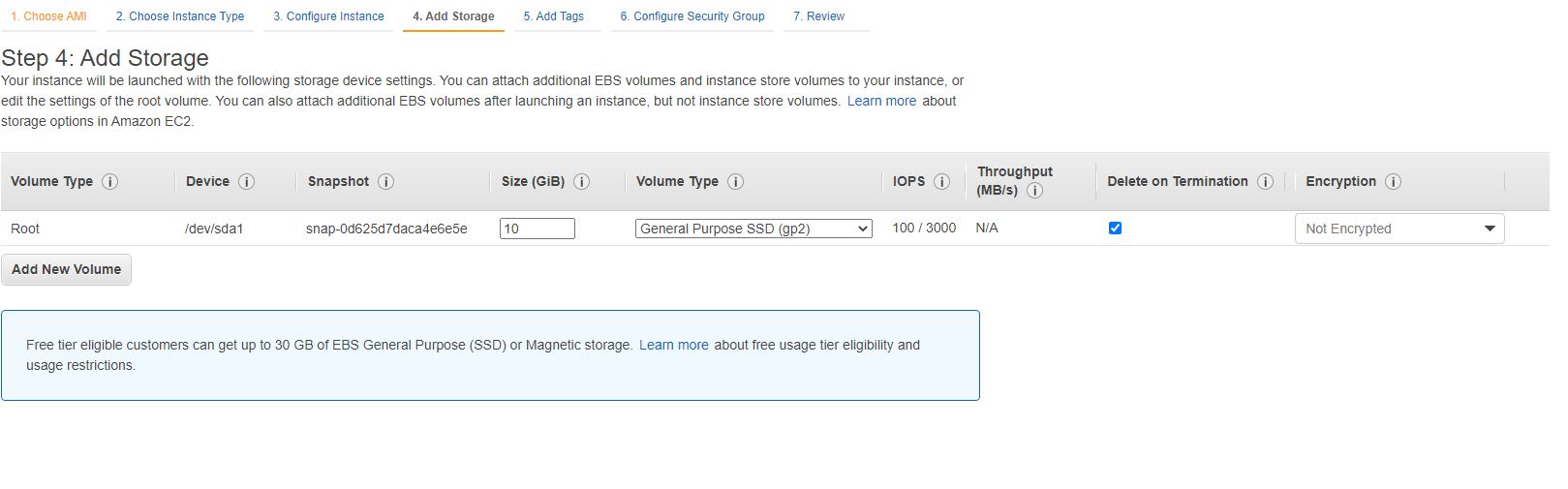


Figure 17 : Storage Details of AMI

We can add new volumes on our storage and make it special like, deletion after termination. We can also encrypt any part of it.

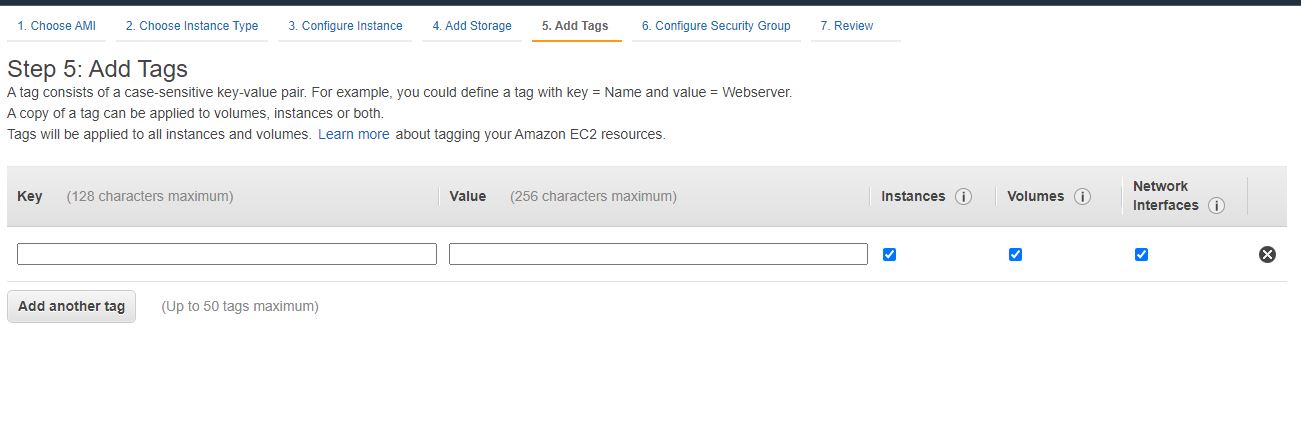
Tags are important if we need to make sure to specify one another.

Figure 18 : Tags of AMI

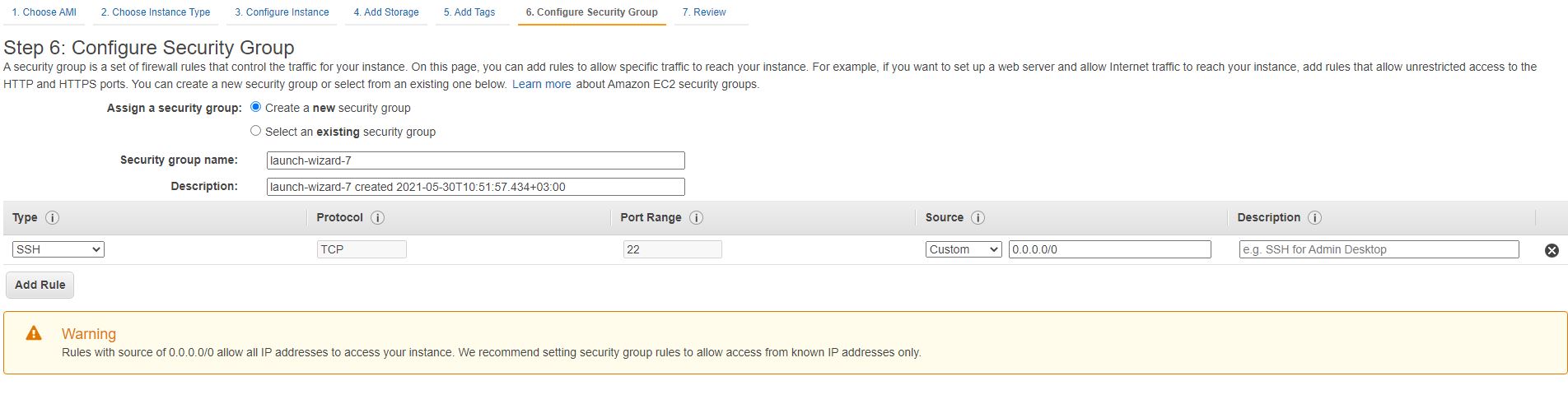


Figure 19 : Configuration fo Security Groups

Configuring security groups are crucial when it comes to delivery and availability. DevOPS teams must decide which types of rules (e.g.HTTP,HTTPS) to make AMI available from everywhere.[6][4]

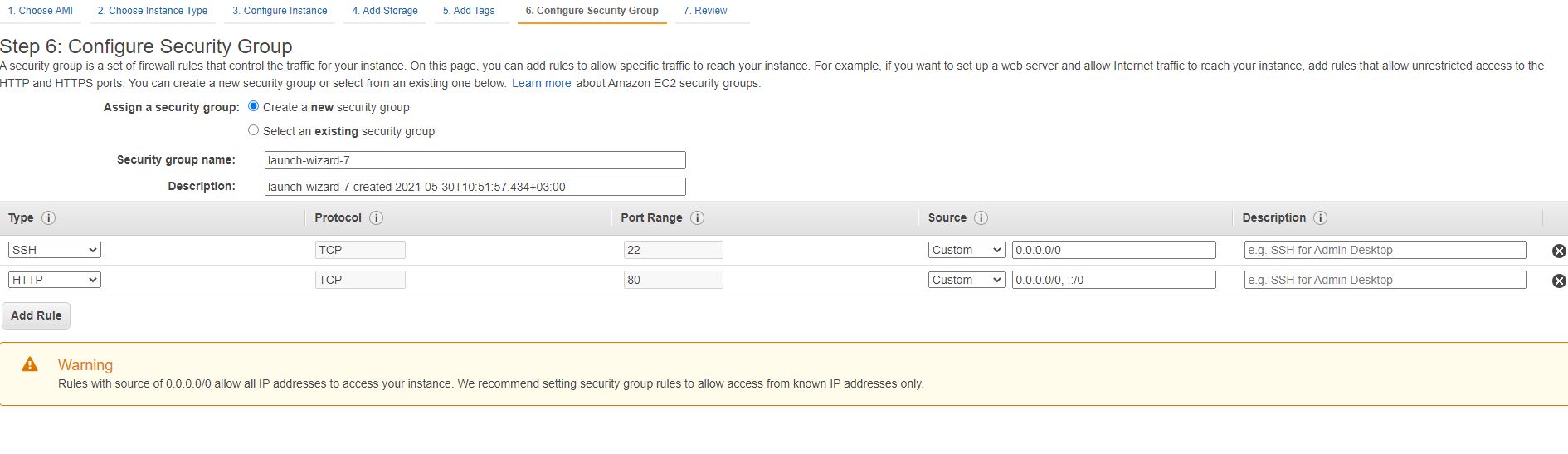


Figure 20 : Adding HTTP Rule

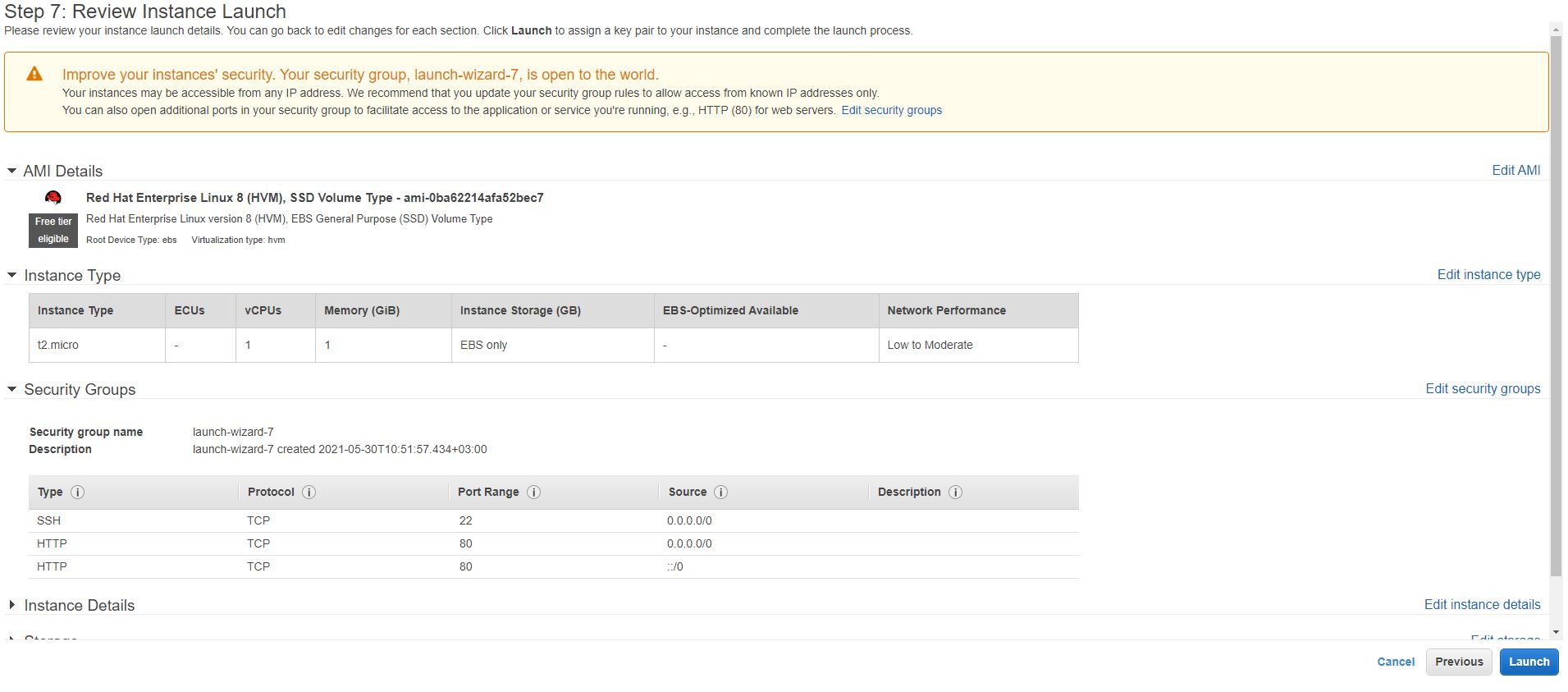


Figure 21 : Adding HTTPS Rule

To make our AMI reachable from other platforms and other virtual systems (e.g. vim,kernel,winRM) we must create a key pair. Key pair has a format of PEM (Privacy-Enhanced Mail) and it’s a known standar for encryption certificates.

Its structures basics has this format :

|  |
| --- |
| -----BEGIN CERTIFICATE-----  (Primary SSL certificate: your\_domain\_name.crt)  -----END CERTIFICATE-----  -----BEGIN CERTIFICATE-----  (Your Intermediate certificate: DigiCertCA.crt)  -----END CERTIFICATE----- |

So for this purpose we need to download and hide a .pem file to reach our AMIs when it needed.

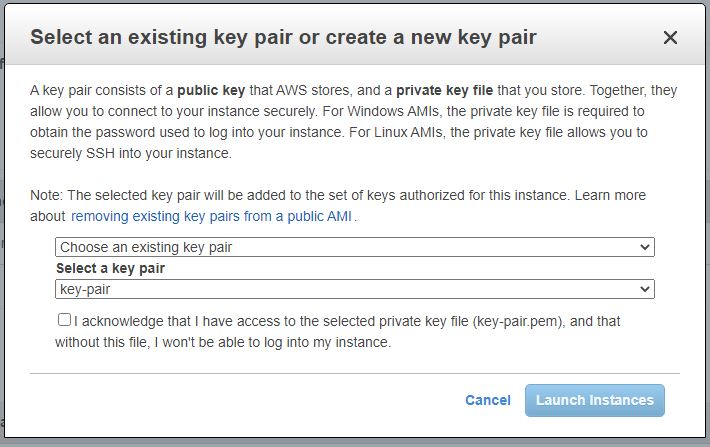


Figure 22 : Key pair (pem file)

After all this long and detailed steps we are close to end this creation of specific AMIs arranged for our purposes.

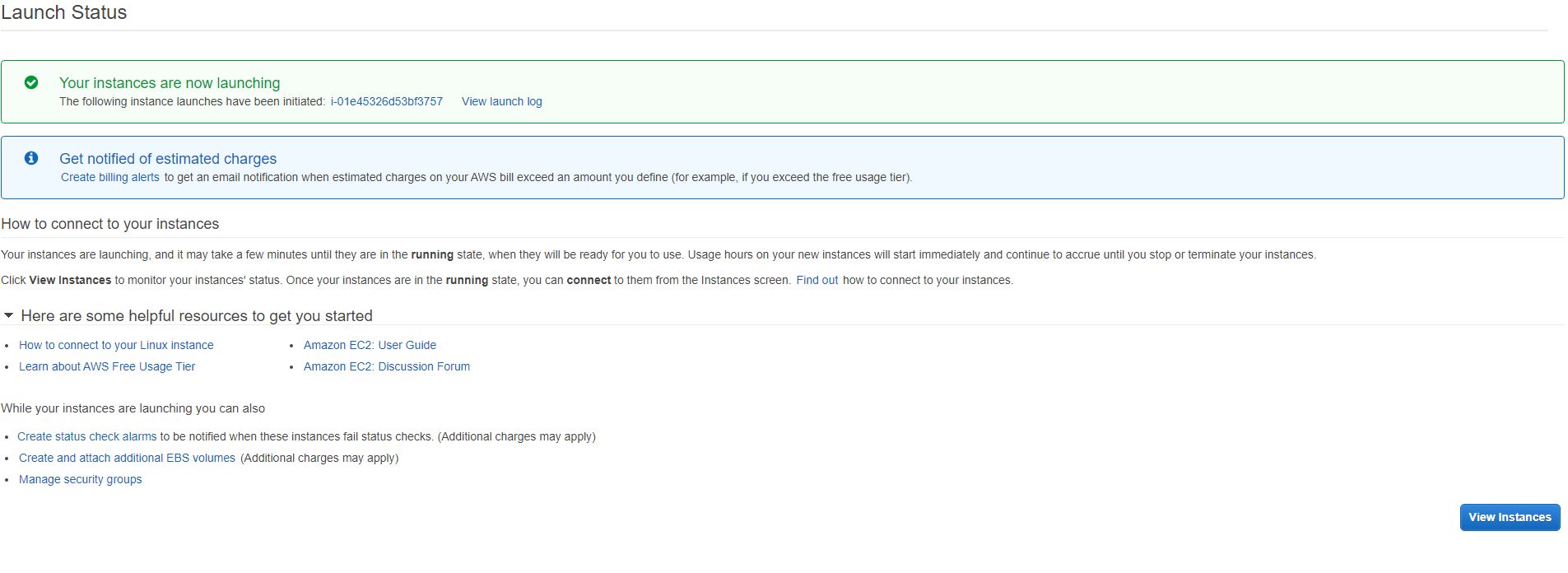
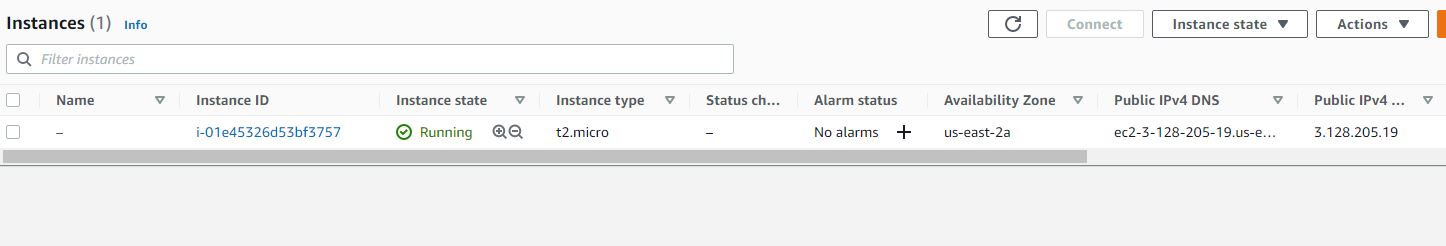


Figure 23 : End of the creation of AMI

Now in the instances panel, we can see our copy (instance of AMI) on the server.



As you can see these long steps can be quickly done by only Packer itself.

First, let’s understand comprehensively Packer’s hierarchy and components.

## Module B

#### USING PACKER

Packer can be used in the numerous ways:

* With a a pre-compiled bin. By using released binaries for all supported platforms and architectures.
* Installing from source code itself. (Can be useful if we only use shell commands rather than GUI)
* Using a system's package management tools (especially on linux deploys).

To use Packer and its components we have chosed Windows 10 as a base platform for configuring its specialities. Therefore we used Chocolatey as a helper tool for installing Packer itself.

To install chocolatey we must open PowerShell as administrative. After then we used shell commands as this ;

|  |
| --- |
| Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1')) |

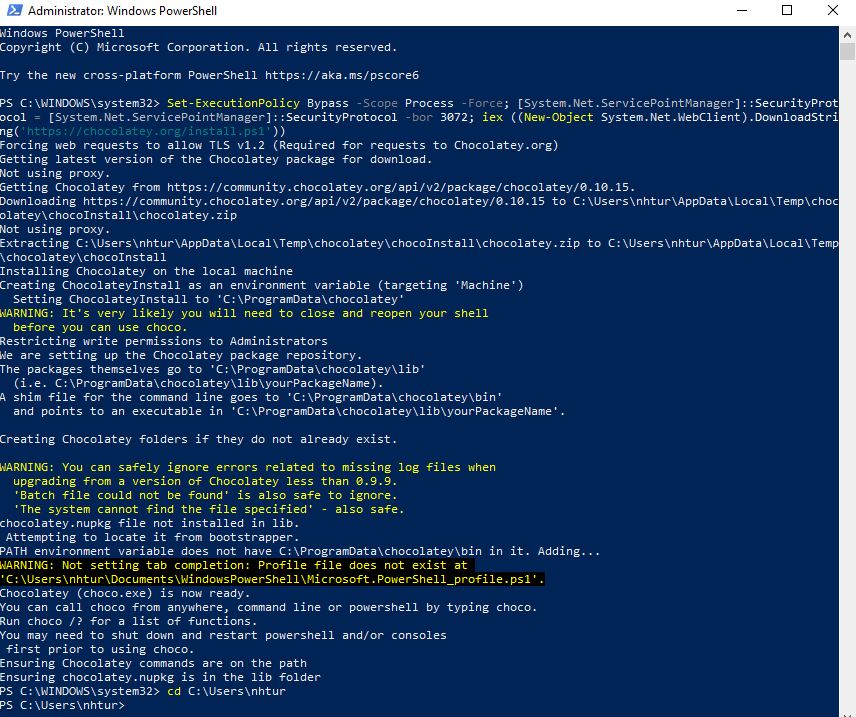


Figure 24 : Installing Chocolatey

Now the system is ready to install Packer with Chocolatey.

In powershell, again we used a Chocolatey command.

|  |
| --- |
| $ choco install packer |

After installation, a verification would be handy.

|  |
| --- |
| $ packer  usage: packer [--version] [--help] <command> [<args>]  Available commands are:  build build image(s) from template  fix fixes templates from old versions of packer  inspect see components of a template  validate check that a template is valid  version Prints the Packer version |

Now we are ready to use all advantages of Packer. Before all of those, let us summarize what Packer have and how we can use its abilities.

#### Packer’s Advantages

### Continuous Delivery

Packer is small, transferable, and command-line driven so it’s easy to use remotely from any platform.

This makes Packer the best tool to put in the CD pipeline.

Packer also might be used to create new VMIs for any kind of platforms on every change to compete Chef,Puppet,Ansible or for any other tools.

Belonging to the pipeline, the newest created VMIs might be instantiated,launched and tested after verifying the infrastructure mutations of the system’s work.

So we can first test and deploy and if requirements are done and tests passed, then we can be sure about the the image will work when it’s deployed on the server. This conducts a new kind of stability,testability and configurability to infrastructure mutations.

### Dev/Prod Parity

Packer helps keep all stages of the work; dev-staging-production all done in once.

Packer also might be used to generate hundreds of VMIs for any kind of platforms simultaneously.

So if we use Amazon Web Services for production but some other VMware for dev, we can generate both an AMI and a VMware image using Packer simultaneously from same Packer template with a JSON file.

### Appliance/Demo Creation

Since the tool creates consistent VMIs for more than once platform in parallel, it’s the best tool for creating applications as one-use product demos for our services.

As software changes, DevOPS teams can automatically generate applications with the software pre-baked (installed).

This means that users can get ready with new software by deploying it to the cloud service of their any desires.

#### Packer’s Structure

Packer has 6 fundamental component in its hierarchy;

* Builders
* Provisioners
* Variables/Templates
* Paralel Builds
* Post-Processors
* Communicators

#### Builders

Builders are used for generating VMIs and creating images from different platforms.

e.g. EC2, VMware, VirtualBox.

Packer has many builders by default but DevOPS teams might also be extended to add new builders for their own purposes.

With packer-init plugin;

|  |
| --- |
| packer {  required\_plugins {  amazon = {  version = ">= 0.0.1"  source = "github.com/hashicorp/amazonAWS"  }  }  } |

On the command-line for simple AWS builder;

|  |
| --- |
| "builders": {  "type": "amazon-ebs",  "access\_key": "A+53dKIAIOSdw2FODNN7EXAMPLE",  "secret\_key": "MDwJaEMMPLIK7MDENG/bMPxRfiCYEXAMPLEKEY",  "region": "us-east-2",  } |

#### Provisioners

Provisioners use built-in and 3rd party applications to install and configurate the VMI after the installation and booting it.

Basically provisioners make the system ready for any usage purpose, so common scenarios for provisioners are these:

* managing plug-ins
* changing the OS kernel
* attending and generating new users
* installing&downloading application codes

A simple shell provisioner example ;

|  |
| --- |
| {  "type": "shell",  "inline": ["echo foo"]  } |

#### Templates – Variables

User variables make sure DevOPS teams can template their system to be further set with custom-defined variables from the coding interpreters like PowerShell,BASH etc., environment variables, Vaults, or directories as files.

This allows DevOPS teams create a framework from custom templates in order to save save tokens, environments and platform specific objects also any other types of information out of custom specific templates.

This feature maximizes the movable of the template for our needs.

|  |
| --- |
| {  "variables": {  "aws\_access\_key": "",  "aws\_secret\_key": ""  },  "builders": [  {  "type": "amazon-ebs",  "access\_key": "{{user `aws\_access\_key`}}",  "secret\_key": "{{user `aws\_secret\_key`}}"  // ...  }  ]  } |

Vault tokens can be used variables too;

|  |
| --- |
| "VAULT\_ADDR"  "VAULT\_AGENT\_ADDR"  "VAULT\_CACERT"  "VAULT\_CAPATH"  "VAULT\_CLIENT\_CERT"  "VAULT\_CLIENT\_KEY"  "VAULT\_CLIENT\_TIMEOUT"  "VAULT\_SKIP\_VERIFY"  "VAULT\_NAMESPACE"  "VAULT\_TLS\_SERVER\_NAME"  "VAULT\_WRAP\_TTL"  "VAULT\_MAX\_RETRIES"  "VAULT\_TOKEN"  "VAULT\_MFA"  "VAULT\_RATE\_LIMIT" |

#### Paralel Builds

Parellel build is a very powerful and crucial feature of Packer and it makes Packer better than others.

Packer can build an AMI and a Azure Image virtual machine in parallel provisioned with the same templates (JSON scripts) and this make them near-identical images.

Imagine requiring using 2 different platform but we can remotely create them with the same scripts as a JSON file.

|  |
| --- |
| source "docker" "ubuntu-bionic" {  image = "ubuntu:bionic"  commit = true  }  build {  sources = [  - "source.docker.ubuntu"  + "source.docker.ubuntu",  + "source.docker.ubuntu-bionic",  ]  ## ...  } |

#### Post-Processors

It’s similar to the provisioners. Post-processors run after the VMI is built by the packer builder and by the provisioner.

Post-processors are not a necessity but it shows a powerful option. They might be used to upload artifacts, packages, or any other needs.

|  |
| --- |
| build {  #build image  post-processor "checksum" {  checksum\_types = [ "md5", "sha512" ]  keep\_input\_artifact = true  }  post-processor "amazon-import" { # upload image to amazon  }  } |

#### Using Packer To Create AWS AMIs

For the final part of the methodology section, we finally show how to use Packer to immediate creation of VMI instances.

**System Tools & Specifications**

* VS Code
* Chocolatey
* Powershell
* Amazon AWS Account
* Windows 10 OS

#### Setting Up VS Code

After opening a new windows we created a new JSON file with a name as AWS. We no ready to create our first AMI from AWS.

For each platforms and services, we created a Packer JSON File as it can be seen on the figure.

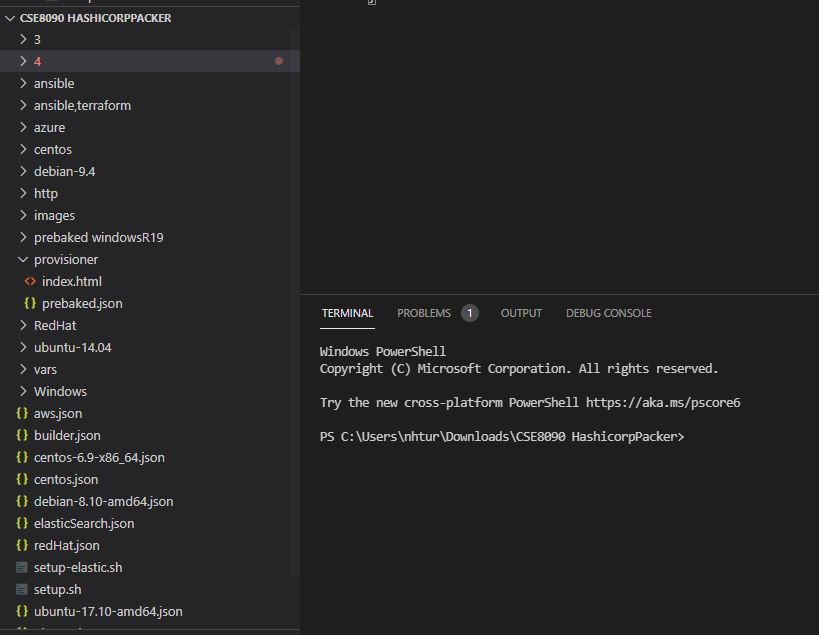


Figure 25 : Setting The VS Code for Packer

Every OS has different credentials, we must first check from AMAZON EC2 servers.

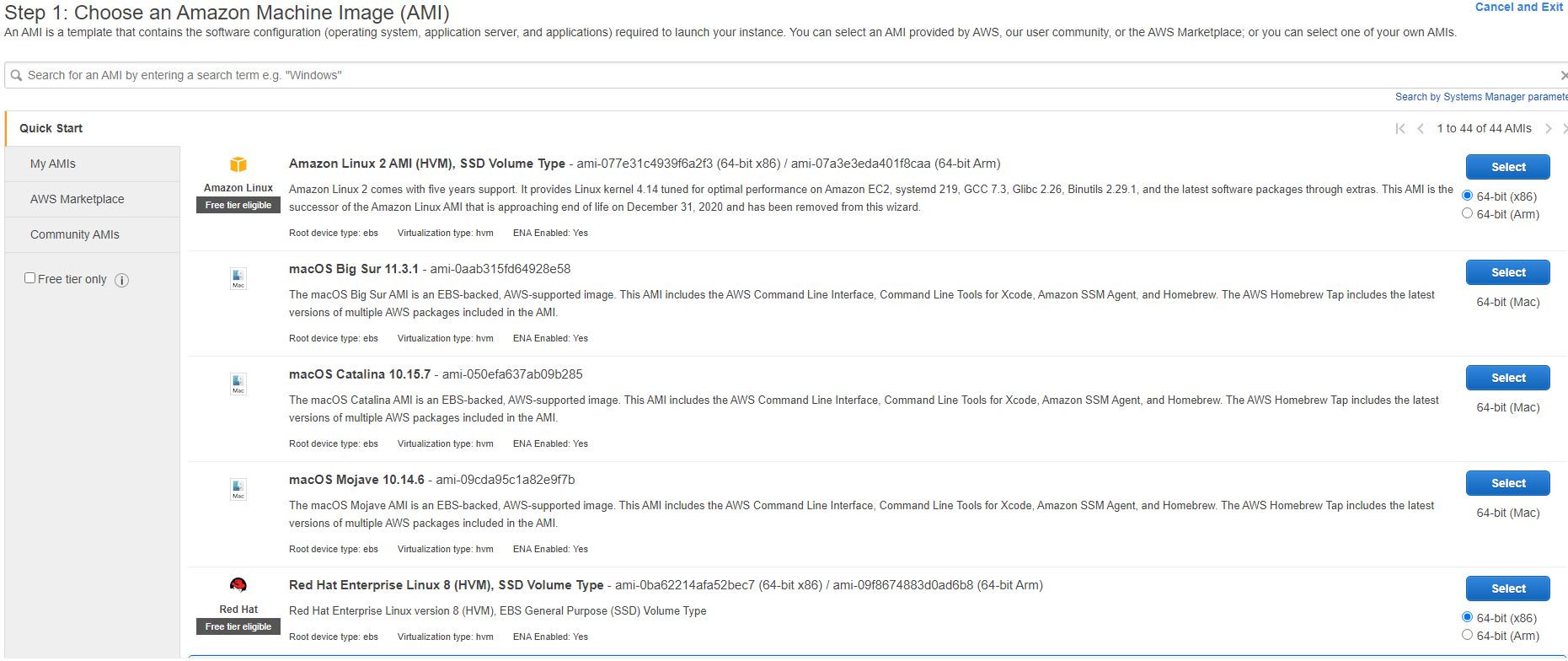
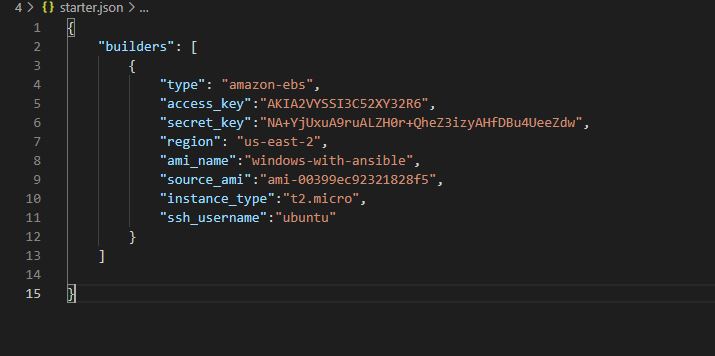


Figure 26 : Choosing AMI ID and Properties



Every AMI has different IDs and properties, we will need those credentials on the JSON file we will use.

First we start with a builder.



**We need some components to build our first AMI ;**

* Type: Which platform ?
* Access key: Specific key for each account
* Secret key: Secret key for each account
* AMI name : Can be chosen any name, no matter
* Source\_ami : Specific ID of the AMI provided by Amazon AWS
* Instance\_type: Which type of storage we need, previously showed in the 2nd section.
* Ssh\_username:This is provided by amazon. Generally it’s “root” or “ec2-user”

For access and secret keys we must take our secret keys from “My Security Credentials” as it seen from the figure.

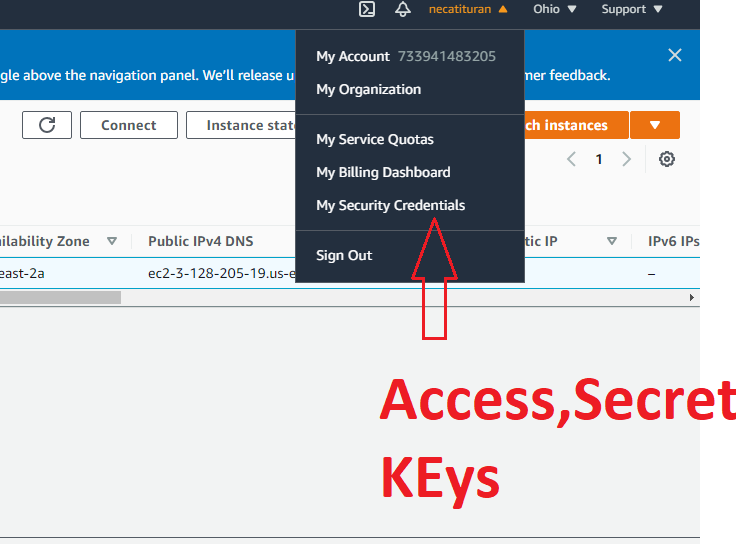


Figure 27: Security Credentials

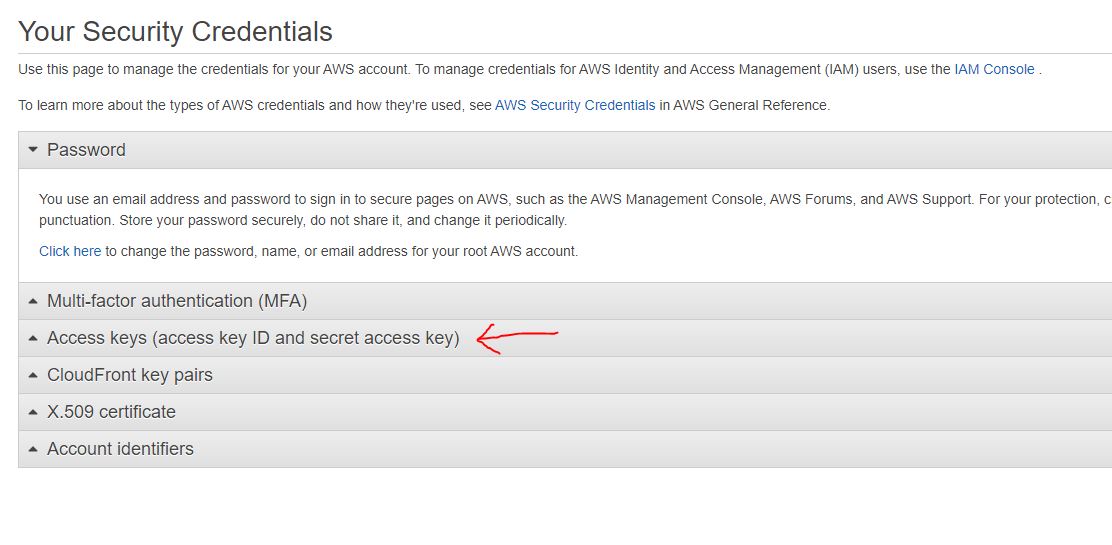


Figure 28 : Access Keys

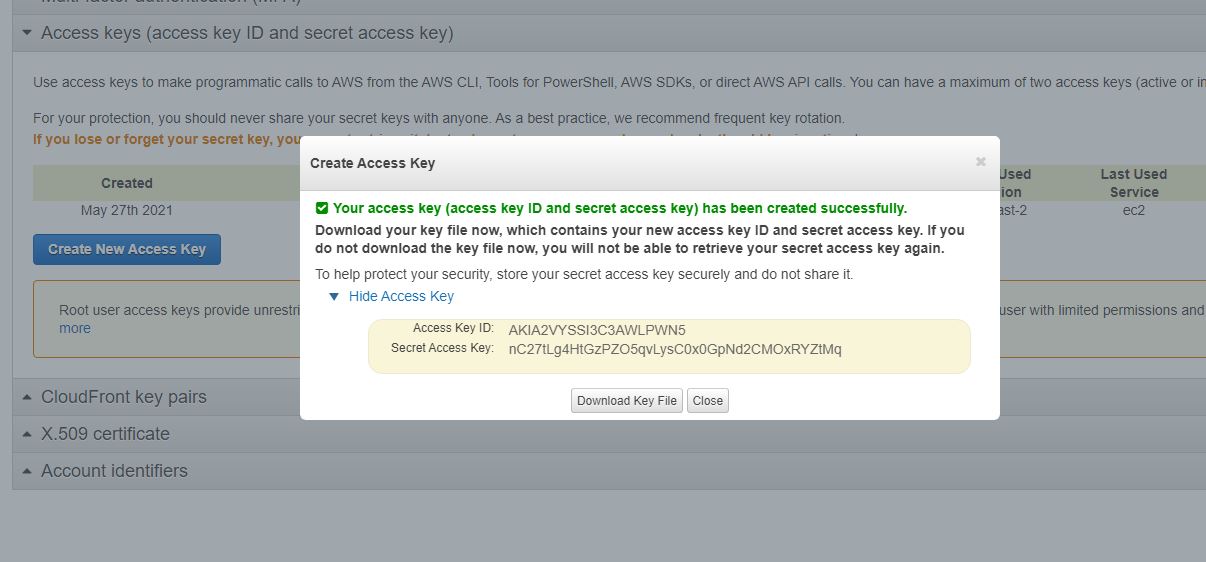


Figure 29 : Downloading Key

Now we can test our first AMI in the VS Code Terminal.

To build our AMI on the terminal, we used this basic command.

|  |
| --- |
| packer build test.json |

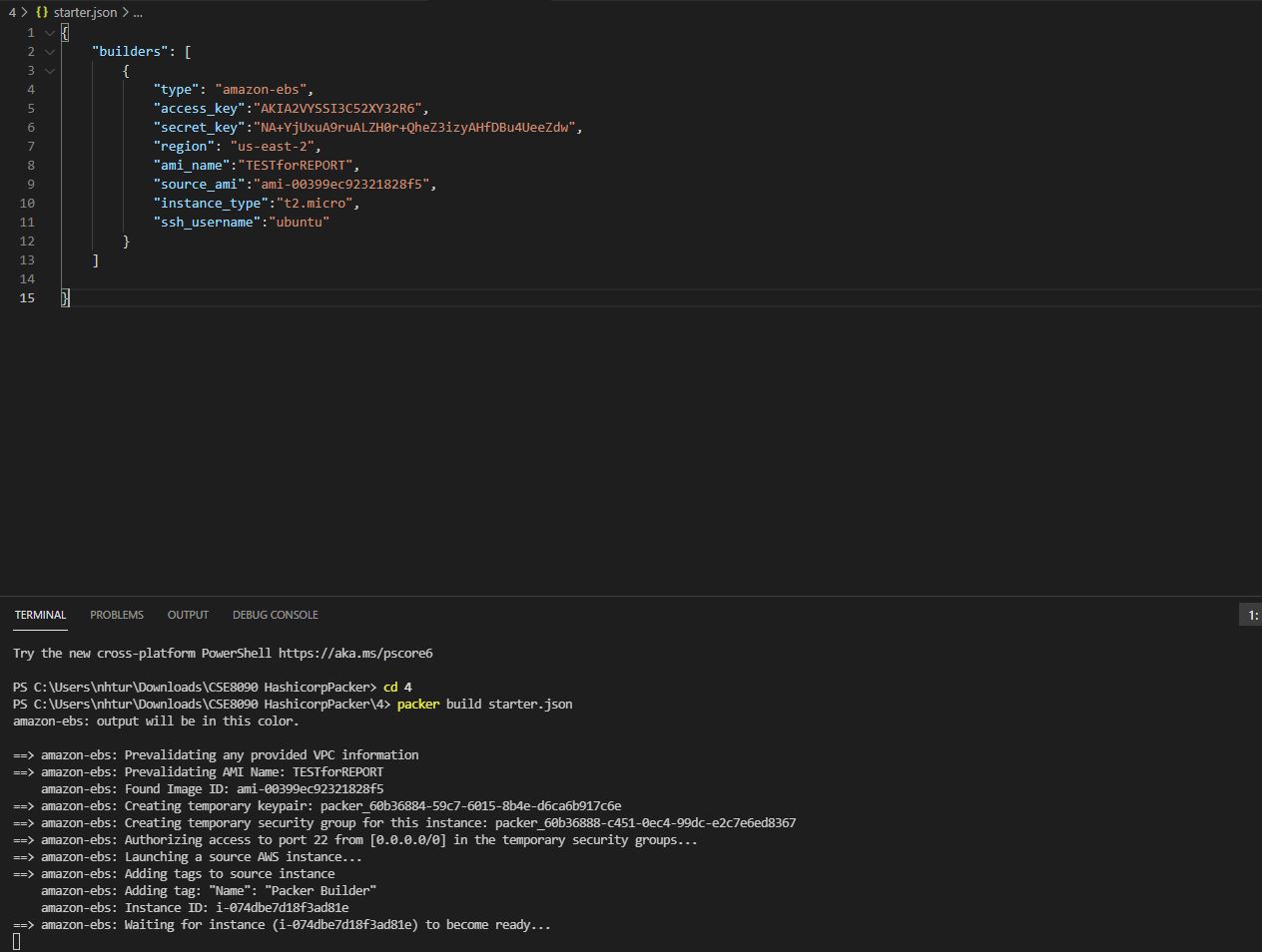


Figure 30 : Packer Basics : Build

Packer will create and AMI and for that purpose it will create an instance and then it will stopped that immediately after creating.

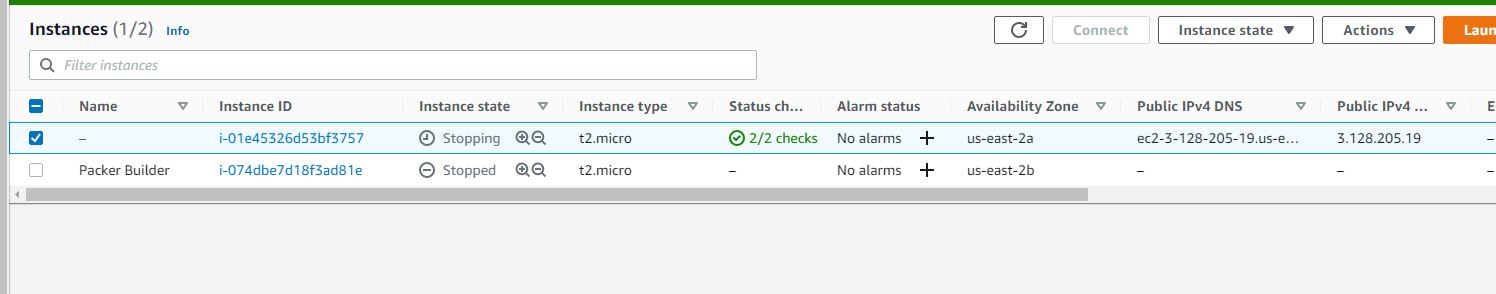
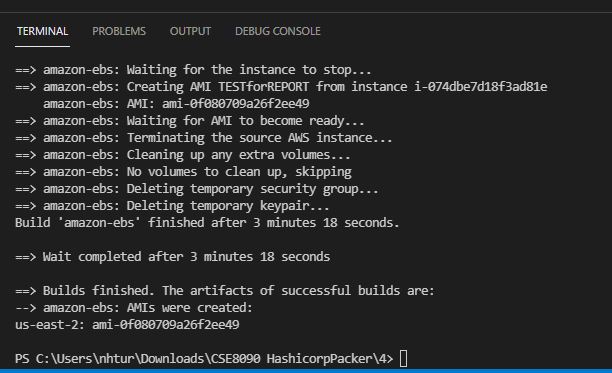


Figure 31 : Packer Builder Seen on AWS

AMI successfully created in 25 seconds from terminal by remote access using only Access keys and secret keys.

 Figure 32 : AMI Created

AMI now can be seen on the EC2 console panel’s AMI section.

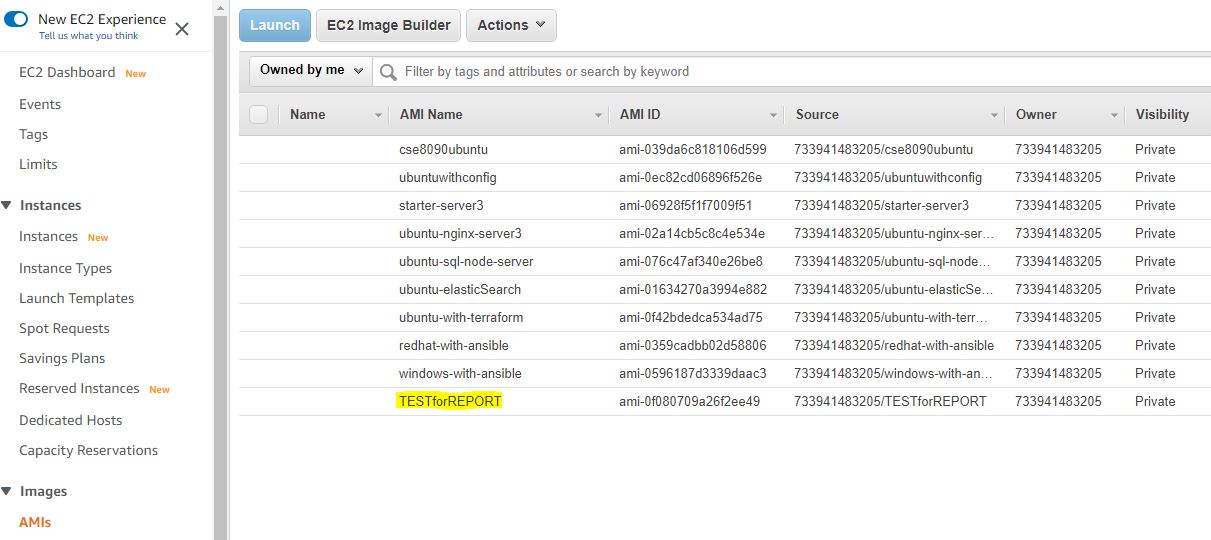


Figure 33 : AMI Seen on AWS

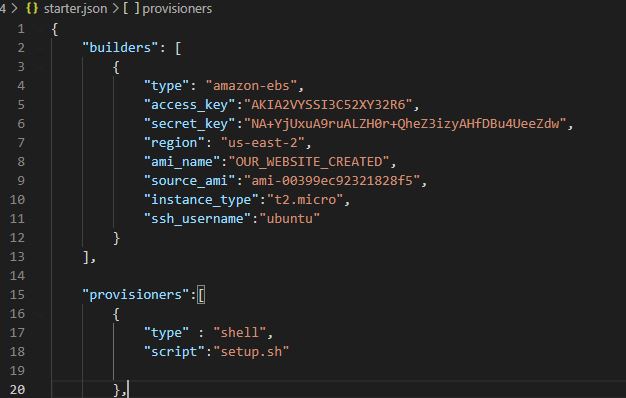
Now we can go further and by only using Shell commands, we can create a directory for our WebSite in the AWS cloud.

We can manipulate all the website and its packages from our JSON File.

Let’s go step by step ;

First again we will build our builder by specific details of the chosen AMI (which is ubuntu for this example)

To make configurations, we need a set of Provisioners.

 Figure 33 : First Website Only With Packer

We need shell and script provisioner to create a .SH file to manipulate the AMI.

Let’s look at first .SH File.

We need NGINX server to be configured first to create a website on AWS. We also need to update the system for this purpose and to make it publicly reachable we need HTTP and HTTPS enabling.

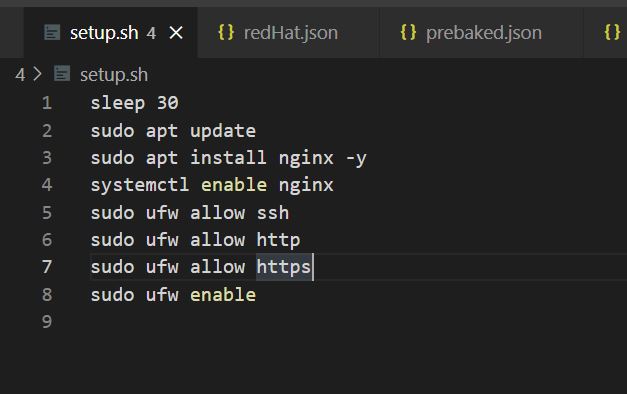
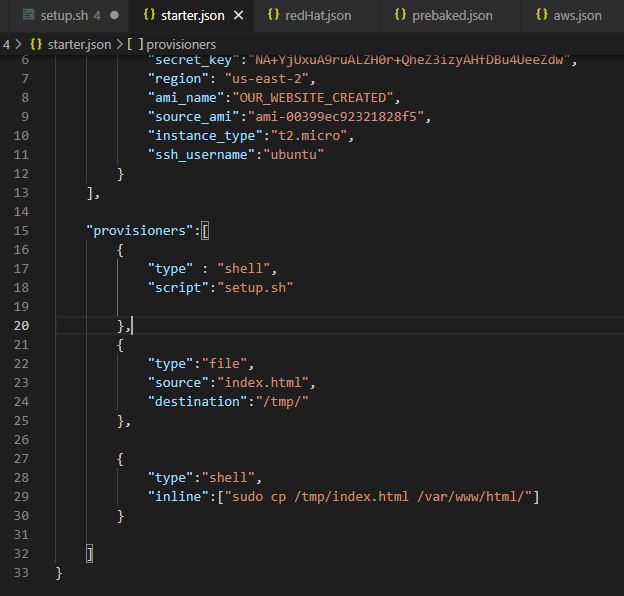
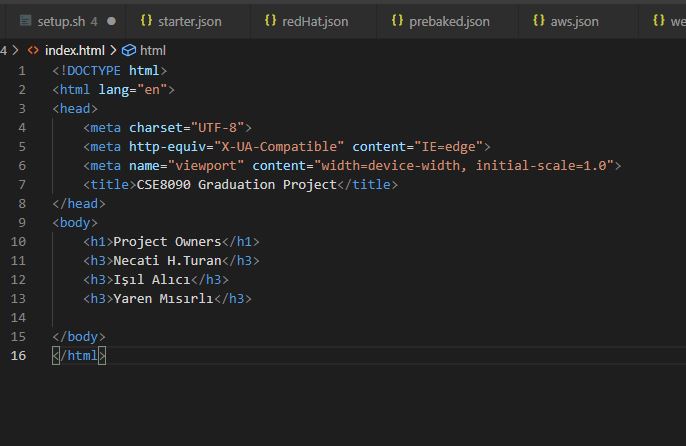


Figure 34 : Shell Commands for NGINX Server

After creating the setup.SH file, we need to attach and make it run in AWS by using packer’s provisioners.

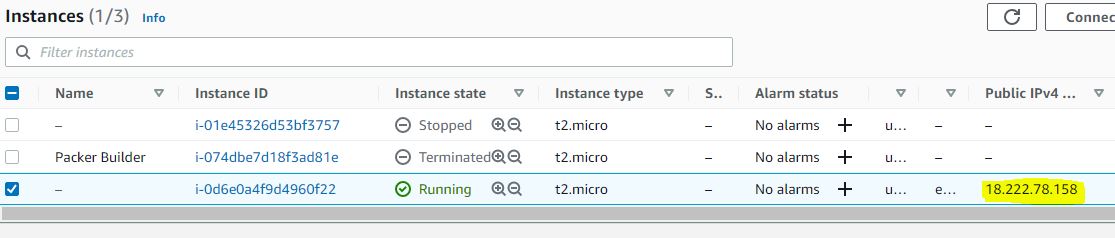
****  **Figure 35 : Sudo Copy Directory**

We copy our index.html file to the server by using Sudo privilege from var/www/ to tmp directory.

****

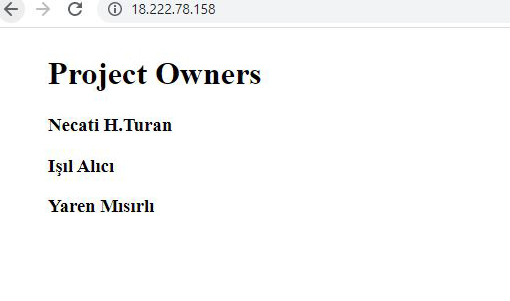
**Figure 36 : Simple HTML File with NGINX**

**Here is our AMIs public IP address, it’s running on the our AWS account.**

****

**Figure 37 : Public IP**

**We can now reach our website with the IP address from anywhere.**

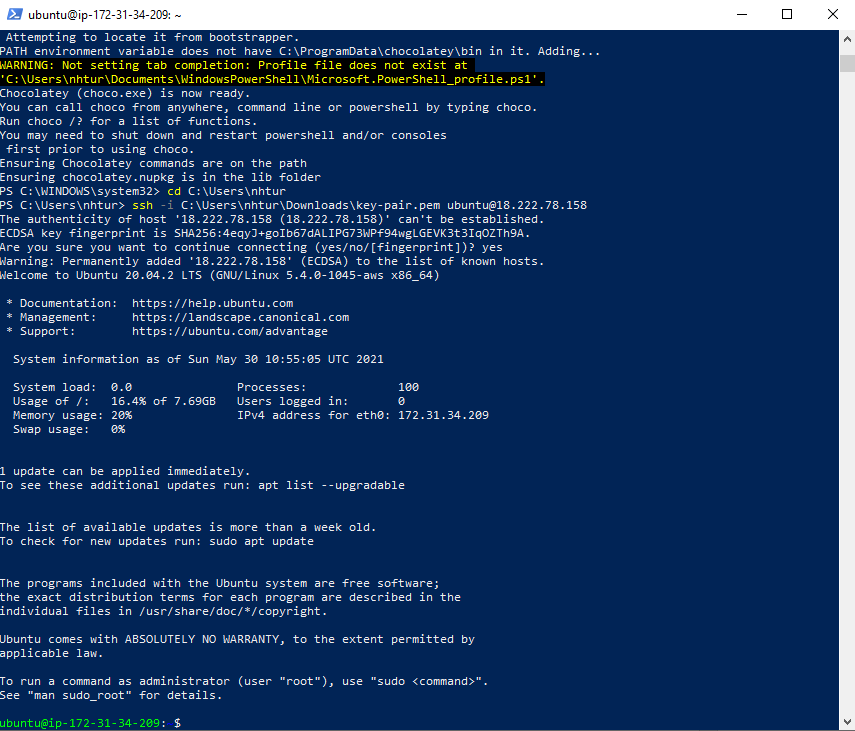
****

**Figure 38 : Up and Running The Website**

From now on, we can manipulate our services and kernell’s feature remotely by only using PowerShell or from any platform we need.

By using our SSH, writed by only key pair direction and the our IP address ;

|  |
| --- |
| ssh -i C:\Users\nhtur\Downloads\key-pair.pem ubuntu@18.222.78.158 |



**Figure 39 : SSH Reach**

We can change and configure privileges by using SUDO or any platform specific command-line helpers.

Here is a list of readable, writable and other permissions of files.

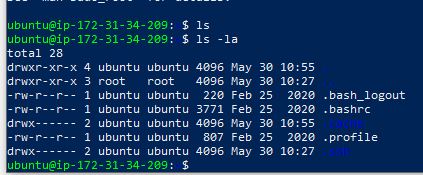


Figure 40 : DRWXR

# Experimental studies

After our work done with Packer, to prove our points and show comprehensive comparisons on Packer, we aim to show studies from other works and statistics about Packer’s opponents.

We showed Packer’s abilities and Immutable servers capabilities.

Here is a resulted diagram of the Immutable again.

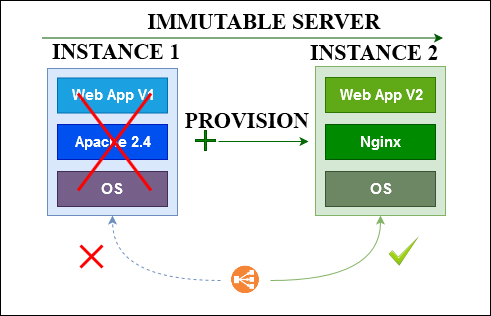


Figure 41:Diagram of Immutable Achieved On This Paper

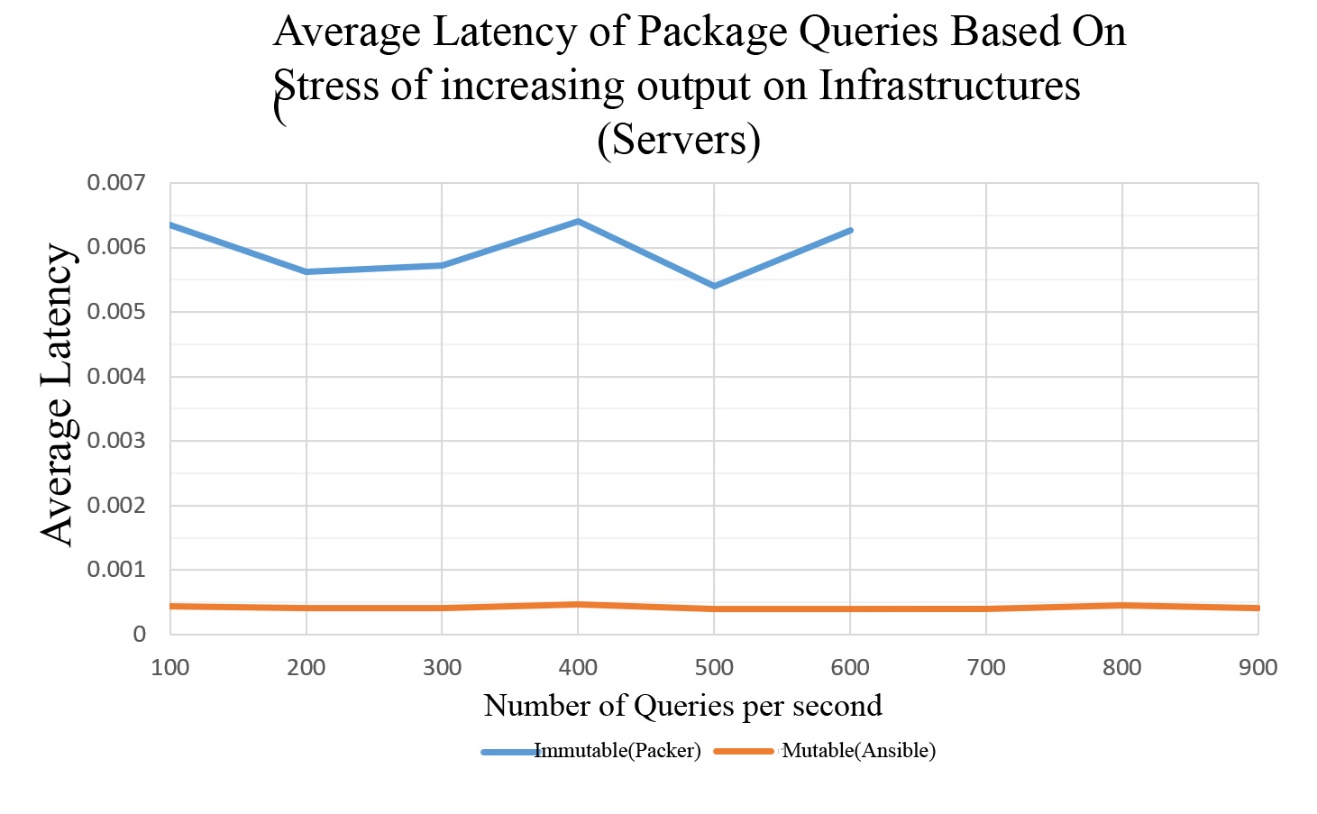


Figure 42 : Benchmark of Packer vs. Opponent

Packer is exceeds over its opponents by memory usages on cloud services like EC2,Azure.  
Here is a figure to show this passage[14].

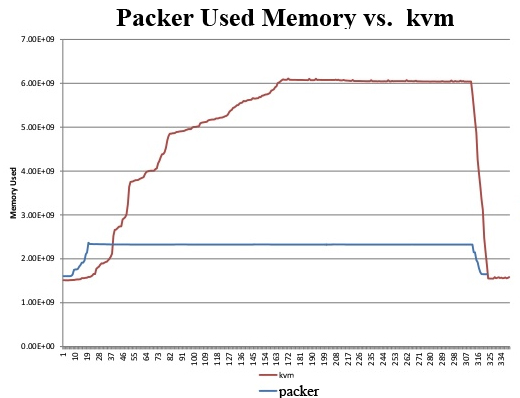


Figure 43:Packer vs. Opponent on Memory Usage

There is also a study on UBUNTU, to prove immutable works faster especially on large scale apps like Notion (notetaking app)[11][13].

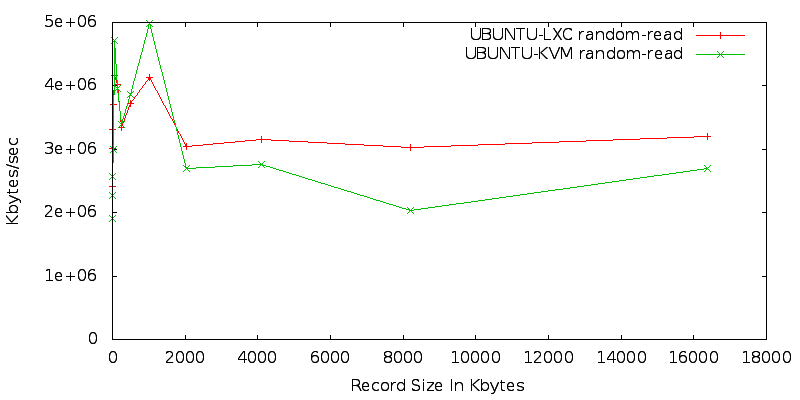


Figure 44: Ubuntu on Cloud Record Sizes

# DISCUSSION

In The Automated Generation of Virtual Machine Images with HasiCorp Packer project aimed the change the notion of using infrastructure of the software deployment and tried to reduce significant amount of time and cost of the general develop-configure-deploy phase of the systems that happene to be installed on local deviceso or cloud based systems like AWS,Azure or Digital Ocean. With these concept of the deployment we automated the system of creation of the VMs and its configurations with the each phases of the required cycle of the generations VMs and secondary/third party applications.

# Conclusions

For developing the most effective deployment system for our services and applications we defined the mutable and immutable systems.

Immutable systems are knowable due to the fact that they have no alterations in it. With our basic introduction of Immutable Infrastructures tried to make a corresponding shift between Immutable infrastructure, is enlight the fact it instantiate a better ,more efficient and most importantly a faster provided desired whole system and we used for this purpose the Hashicorp’s ecosystem’s Packer. We have also have discussed the packer’s abilities and benefits when it comes to comparison of the other DevOPS tools like Linkit, Ansible and Terraform. To make our points solid, this paper tried to cover all the issues and uses of a infrastructure by using Packer itself and by showing other techniques used to be popular. We also talked decently about DevOps maintenance and configurated systems of their own evolution. The main purpose and the main contribution of this paper is to enlighten reader about the approach of immutable infrastructure among other techniques by using Packer and comparing others with it.

# REFERENCES

Every citation made in the body of the project report must appear in the References. Similarly, every item listed in the References must be cited in the body of the report. Follow the APA standard method for citing and listing both the print references and online references. Examples;

1. L. J. Bass, I. M. Weber, and L. Zhu, DevOps - A Software Architect’s Perspective., ser. SEI series in software engineering. AddisonWesley, 2015.
2. K. Morris, Infrastructure as Code. O’Reilly Media, Inc., 2016. [Online]. Available: https://www.oreilly.com/library/ view/infrastructure-as-code/9781491924334/
3. P. Lipton, D. Palma, M. Rutkowski, and D. A. Tamburri, “TOSCA Solves Big Problems in the Cloud and Beyond!” IEEE Cloud Computing, pp. 1–1, 2018.
4. Within-project Defect Prediction of Infrastructure-as-Code Using Product and Process Metrics Stefano Dalla Palma∗ , Dario Di Nucci∗ , Fabio Palomba† , and Damian A. Tamburri‡ ∗Tilburg University, Jheronimous Academy of Data Science, Netherlands †Software Engineering (SeSa) Lab — University of Salerno, Italy ‡Eindhoven University of Technology, Jheronimous Academy of Data Science, Netherlands
5. Managing Virtual Appliance Lifecycle in IaaS and PaaS Clouds, Michal Kimle∗ , L’ubomír Košarišt’an, Boris Parák, Zdenek Šustr
6. On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric, Julio Sandobalin, Emilio Insfran
7. Peter Vaillancourt, Bennett Wineholt, Brandon Barker, Plato Deliyannis, Jackie Zheng, Akshay Suresh, Adam Brazier, Rich Knepper, and Rich Wolski. 2020. Reproducible and Portable Workflows for Scientific Computing and HPC in the Cloud. DOI:https://doi.org/10.1145/3311790.3396659
8. Jack Cook, Richard Weiss, and Jens Mache. 2020. Refactoring a full stack web application to remove barriers for student developers and to add customization for instructors. <i>J. Comput. Sci. Coll.</i> 36, 1 (October 2020), 35–44.
9. Elena A. Araujo, Álvaro M. Espíndola, Vinicius Cardoso Garcia, and Ricardo Terra. 2020. Applying a Multi-platform Architectural Conformance Solution in a Real-world Microservice-based System. In <i>Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse</i> (<i>SBCARS '20</i>). Association for Computing Machinery, New York, NY, USA, 41–50. DOI:https://doi.org/10.1145/3425269.3425270
10. A. Cepuc, R. Botez, O. Craciun, I. -A. Ivanciu and V. Dobrota, "Implementation of a Continuous Integration and Deployment Pipeline for Containerized Applications in Amazon Web Services Using Jenkins, Ansible and Kubernetes," 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet), 2020, pp. 1-6, doi: 10.1109/RoEduNet51892.2020.9324857.
11. Managing Virtual Appliance Lifecycle in IaaS and PaaS Clouds, , L’ubomír Košarišt’an, Boris Parák, Zdenek Šustr ˇ
12. NIFTY, [Online] Available: https://github.com/CESNET/nifty [Accessed: November 11, 2016].
13. ITCHY, [Online] Available: https://github.com/CESNET/itchy [Accessed: November 11, 2016].
14. ] HEPIX Virtualisation Working Group, August 26, 2012, [Online] Available: http://grid.desy.de/vm/hepix/vwg/doc/pdf/Book-a4.pdf [Accessed: November 11, 2016].
15. Docker, [Online] Available: https://www.docker.com/ [Accessed: November 11, 2016].
16. git-scp, [Online] Available: https://git-scm.com/ [Accessed: November 11, 2016].
17. cloud-init, [Online] Available: https://cloudinit.readthedocs.org/en/latest/ [Accessed: November 11, 2016].
18. COMFY, [Online] Available: https://github.com/CESNET/comfy [Accessed: November 11, 2016].
19. Y. Brikman, Terraform: Up and Running, 1st ed. Newton, MA, USA: O’Reilly Media, 2017.
20. Y. Martínez, C. Cachero, and S. Meliá, ‘‘MDD vs. traditional software development: A practitioner’s subjective perspective,’’ Inf. Softw. Technol., vol. 55, no. 2, pp. 189–200, Feb. 2013
21. Amazon Web Services. (2011). OpsWorks. Accessed: Aug. 29, 2019. [Online]. Available: <https://aws.amazon.com/opsworks/>
22. A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, ‘‘A systematic mapping study of infrastructure as code research,’’ Inf. Softw. Technol., vol. 108, pp. 65–77, Apr. 2019
23. Immutable Infrastructure Calls for Immutable Architecture,Conference Paper 2019, DOI: 10.24251/HICSS.2019.846