Matias Alander

Migration from data centre to AWS cloud using packer

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Bachelor's Thesis

7 November 2022

Abstract

Author:	Matias Alander
Title:	Migration from data centre to AWS cloud using packer
Number of Pages:	36 pages + 1 appendix
Date:	7 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	IoT and Cloud Computing
Supervisors:	Riku Kuikka, Infra Lead Janne Salonen, Head of School, ICT

The purpose of this thesis is to create an automated pipeline that builds immutable image to AWS. This was done to a SaaS company that wanted to migrate a system from datacentre to a cloud utilizing packer.

The project was part of the SaaS company's cloud migration. The theoretical part focuses on the benefits of cloud migration and how to prepare going towards cloud. The project migrated from data centre to AWS cloud utilizing Hashicorps packer and ansible. The project is built on a previously built iac infracstrure.

The outcome of this thesis gives the SaaS company a good stepping block for making their cloud transformation. The pipeline created allows a more automated way to move Windows servers to cloud and scale them if needed.

Keywords: AWS, Packer, ansible, Hashicorp, multi cloud, migration

Tiivistelmä

Tekijä:	Matias Alander
Otsikko:	Migraatio datacenteristä pilvipalveluun käyttäen packeria
Sivumäärä:	36 sivua + 1 liite
Aika:	7.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan
Ammatillinen pääaine:	verkot ja pilvipalvelut
Ohjaajat:	Infra Lead Riku Kuikka Osaamisaluepäällikkö Janne Salonen

Insinöörityön tarkoituksena oli luoda automatisoitu putki, joka rakentaa muuttumattomia palvelinkuvia AWS-pilvipalveluun. SaaS-yritys halusi siirtää datakeskuksesta systeemin pilveen käyttäen packeria.

Projekti tehtiin osana SaaS-yritys pilvimigraatiota. Teoria käy läpi pilven hyötyjä ja sitä, kuinka valmistaa migraatiota pilvipalveluun. Projektiksi tuli siirtää systeemi datakeskuksesta AWS-pilveen käyttäen Hashicorpin packeria ja ansibelia. Projektissa hyödynnettiin jo valmiina olevaa iac-ympäristöä.

Lopputulokseksi tuli SaaS-yritykselle askelma heidän pilvimigraatio-projektiin. Projekti antaa putken automatisoidumpaan tapaan siirtää heidän Windows-palvelimet pilvipalveluun ja skaalata niitä halutusti.

Avainsanat: AWS, Packer, ansible, Hashicorp, multi cloud, migraatio

Contents

List of Abbreviations

AMI	Automated machine image
AWS	Amazons web services
Cd	Continuous delivery
Ci	Continuous integration
Ec2	Elastic compute cloud
Gcp	Googles cloud environment
HCL	Hashicorp configuration language
Hdd	Hard disk drive
HTTPS	Hypertext transfer protocol secure
IaaS	infrastructure as a service
Iac	infracture as a code
PaaS	platform as a service
RAM	Random access memory
RD	Remote desktop
Rdp	Remote desktop protocol
s3	simple storage service

SaaS	software as a service
SLA	Service level agreement
Ssd	Solid-state drive
Ssh	Secure shell protocol
vCPU	Virtual processor
Vpn	Virtual private network
YAML	YAML Ain’t Markup Language

1 Introduction

As technology is developing faster than ever, also infrastructure needs to adapt to the pace to allow normal server needs to transform and scale faster. The solution that many have found is cloud computing, which can provide the scalability and automation that normal datacentres struggle to provide at the same price. There is a growing need to automize and make immutable servers that scale and provide services

This thesis provides general understanding of cloud migration through theory and describes a final year project carried out to SaaS company. The theory part explains the benefits of cloud compared to onsite datacentre.

The SaaS company recommended a project for moving their service from datacentre to cloud. This was done by utilizing AWS, packer and ansible for creating a pipeline to make automated immutable machine images.

2 Project planning

The project was planned when the company was on migration and modernisation of code and systems. I was given the task of moving and utilizing new modern cloud technologies for windows servers. Old servers were made for load balancing utilizing third-party load balancer and they were hosted on datacentre.

My project was to make the servers immutable and so that the servers could be scaled up and down by need. The requirements were to make it work on AWS through the terraform and ci/cd pipes to make everything automatic so when the need for updating comes for the servers you would only need to make some changes in code to update all environments or make it go through each environment separately.

The plan was utilizing the packer which is from the same creators as terraform and build an image for AWS environment. The next step was to utilize the ansible script to build the required software for server and make it scalable for AWS environment. At the end of project there was also needed to build a rdp server for connecting the servers.

3 Migration to cloud

The current trend in the industry is moving towards the cloud. Cloud is on the rise and many enterprises are evaluating the process of cloud migration. [1] The process starts with a goal in mind for modernization. With a solid goal the evaluation for how it will be implemented on your services or applications can be done.

3.1 Benefits of migration to cloud

The benefits that are achieved from migration are usually lower upfront operating costs and the no need to upgrade physical hardware for servers. Many companies are for the scaling because cloud providers can offer better than normal datacentre operators. Some of the benefits companies seek from cloud environment are as follows:

- Cost reduction
- Scalability
- Security
- Resiliency
- Fast implementation
- Availability
- Reliability

Cost reduction is a massive thing for cloud as the upfront cost for creating cloud environments is small. Also, straight transfer from on premises datacentres does not always bring cost down. The best way to bring costs down on cloud is optimizing the workload for cloud to see the savings that cloud providers talk about. [2]

Scalability brings companies greater opportunities for optimizing workload. Having workloads in the cloud, you can quickly respond to peak demands and lower capacity when it is necessary. All that is done automatically and does not

require a lot of time and effort. Compared for on premises you would have to purchase and calculate the required equipment for those peak times for your service to be operational. [2]

Security is massive thing for cloud providers and its enterprises. To keep things secure cloud providers keep updating their security to be up to latest industry standards. [2] Resilience comes from cloud providers implementation of backups and disaster strategies that they provide. These implementations can be replicating data on multiple data centres. [2]

Fast implementation comes from clouds adaptability for diverse kinds of workloads. from the sheer number of datacentres that cloud providers offer for connecting. On cloud there is no need to think server space for future as there is always computing available. [2]

Availability is accessed for cloud platforms. Compared to datacentre or similar the cloud has greater availability as you can access it anywhere and it is not necessary to configure vpn. [2] Reliability comes from cloud providers SLAs which provide almost always on infrastructure. Cloud providers provide their SLAs uptime percentage of how many minutes in a month they are down by instance. On following tables 1-3 are top three cloud providers computing instances service level agreements. [2] Percentage is calculated using a formula.

$$\text{Monthly uptime\%} = \frac{(\text{Maximum available minutes} - \text{Downtime})}{\text{Maximum available minutes}} x100$$

[3]

Table 1 illustrates where AWS ec2 instances SLAs is shown by region and instance level and how much they give service credits if it happens. When studying the AWS SLA, the table shows how it has been the industry leader as it can provide the best availability on region level and instance level.

Table 1. AWS ec2 instance SLA between region-level and instance-level. [4]

Region-Level SLA	Instance-Level SLA	Service credit Percentage
99.99%	99.5%	10%
99.0%	99.0%	30%
95.0%	95.0%	100%

Table 2. Gcp SLA for compute engine. [5]

Instances in Multiple Zones and Load balancing	A Single Instance	Service credit Percentage
< 99.99%	< 99.50%	10%
< 99.00%	< 95.00%	25%
< 95.00%	< 90.00%	100%

Table 3. Azure SLA for virtual machines. [3]

deployed across two or more Availability Zones	Availability Set, or same Dedicated Host Group	Premium and Ultra SSD	Standard SSD Managed Disk	Standard HDD Managed Disk	Service Credit
< 99.99%	< 99.95%	< 99.9%	< 99.5%	< 95%	10%
< 99%	< 99%	< 99%	< 95%	< 92%	25%
< 95%	< 95%	< 95%	< 90%	< 90%	100%

These seven benefits are the main reasons why organisations move toward cloud. Some organizations choose to start using cloud because of the easier access for connecting sites as there is less need to be physically on sites.

3.2 Types of cloud service models

Cloud has been categorized for three main types which describe their responsibility between the company and cloud provider.

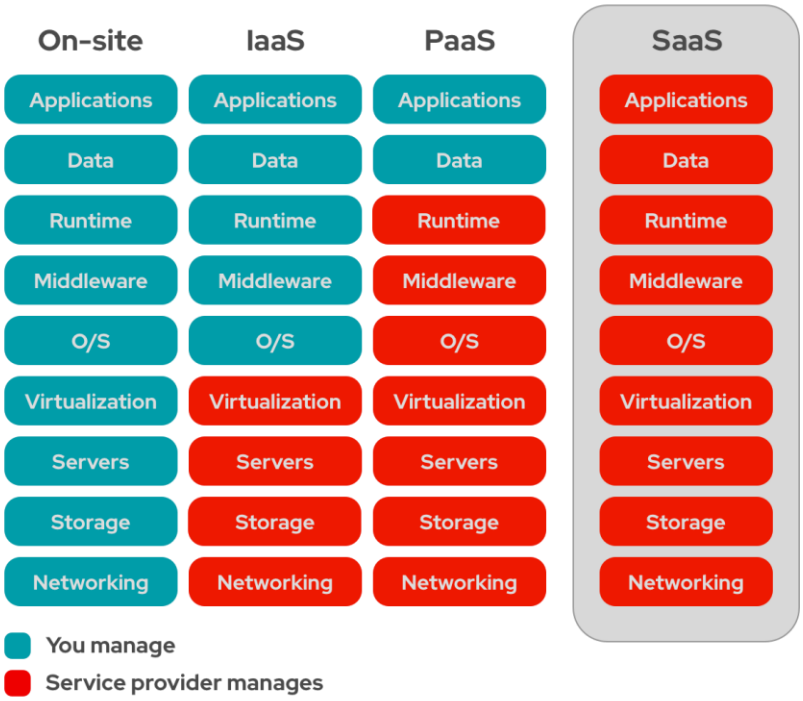
Infrastructure as a service (IaaS) is a type of cloud where a cloud provider gives access to their resources for computing and networking. This model gives most responsibility for the company as cloud providers responsibility is only virtualization, servers, storage, and networking. It is also the most flexible one as this gives the company clean sandbox to build their resources. [1]

Platform as a service (PaaS) provides a cloud-based environment with computing, storage, network, and other capabilities to support the complete life cycle of implementation, deployment, and delivery of applications, without the cost and complexity of buying and managing the underlying hardware software, provisioning, and hosting. [1]

Software as a service (SaaS) or cloud-based applications, run on distant computers “in the cloud” owned and operated by cloud providers. The user can access them from their computers through the internet and a web browser. [1]

On-site is the environment on your datacentre where management all lies on your shoulder. Its maintenance and operating cost are all on company’s shoulder. Security is maintained as often security personnel updates security guidelines. Compared other service models the underlying security and disaster recovery is all on organisation’s shoulders. [1]

For choosing right service model it should not be just one. Every application can be on different service models depending how much the company wants to control its management. Picture 1 presents the responsibilities of a company.



Picture 1. The responsibility for you and the provider of cloud between the cloud types. [6]

3.3 Cloud deployment models

Deployment model describes the popular implementations of could and their benefits. According to deployment model

Private cloud means that it has been bought from a cloud provider rack space or hardware that it is only for your company's control trough cloud providers interface. It has also stricter security policies for its governance. Single tenancy its selling point for security as your company is the only one that has access to it. Location of private cloud usually is on cloud providers data centre, but it can also be in on premises. Even if it is on on-site it can utilize cloud-native architecture. [1, 7]

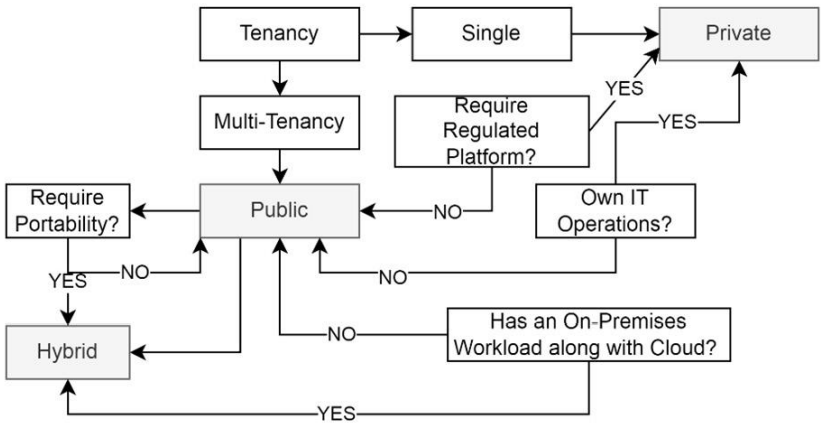
Public cloud is the most common type for cloud deployment chosen when beginning implementing a cloud for a company. Public cloud usually has more

generalized platforms compared to private as company can define what they want to have on their cloud deployment. With generalization comes savings and there is no overhead for resources if configured right. [1, 7]

Hybrid cloud means that you utilize cloud infrastructure and on-premises workloads together. This model brings portability as not every workload has to be on cloud infrastructure. This solution brings flexibility as some organizations cannot migrate all their workloads to cloud. Hybrid cloud gives several possibilities to an organisation how to utilize cloud. Portability is an important aspect on hybrid cloud as organisations tend to design their system not to be dependent on cloud providers specialized solutions. [1, 7]

These are the main three cloud deployment models but there is also a so called community cloud, which is like a public cloud, but its its resources are shared between companies that utilize a specific community cloud. [1, 7]

Picture 2 shows how to choose a cloud deployment model according to the company's needs.



Picture 2. A decision tree for choosing fitting cloud delivery model. [1]

3.4 Multi-cloud

Multi-cloud is a concept of utilizing more than one cloud provider. Every cloud provider offers different resources.

Taking advantage of more than one cloud gives more freedom regarding where you keep the data as every provider has their data centres around globe. Using multiple providers can also bring the flexibility for hiring talent. When using multiple clouds, it brings more complexity for implementation, but it gives better resilience for storing data when it not just on the mercy of one provider. [8]

AWS is the most used cloud provider as it is also the oldest. Its benefits come from the maturity on cloud ecosystem and their custom processors on cloud data centres. AWS designs their own processors and has massive engineering supply chain supporting their cloud infrastructure. [8]

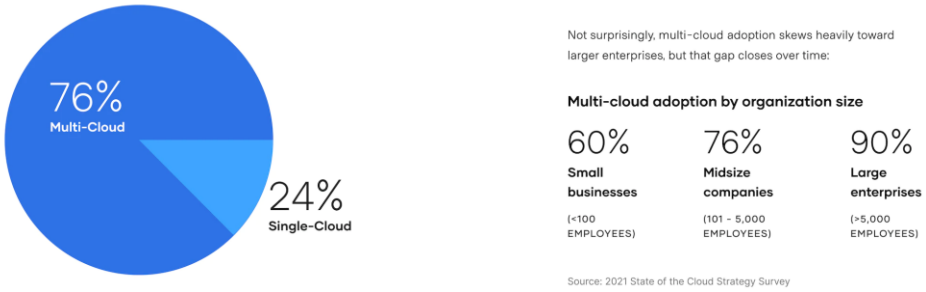
Azures benefits comes from its familiar Microsoft ecosystem and its Microsoft specific cloud applications. Azures active directory is a big selling point in its cloud environment. Azure has a large customer base from Microsoft environment and is broadly appealing to its customer base. [8]

Gcp is Googles cloud environment which has brought now widely used Kubernetes. AI implications are big in Google’s cloud platform. Their market understanding on AI is their big selling point. [8, 9]

The important thing of the multi-cloud is not being lockdown on vendor locking as many cloud providers have different implementations for their services. This is same benefit that hybrid cloud provides. In multi cloud it’s easier to meet compliance requirements as there are more data centres to store data if or when there are compliance requirements different types of data. [8,10]

Many companies have implemented more than one cloud on their infrastructure as can be seen on picture 3. According to HashiCorp’s 2021 State of Cloud

Strategy Survey, 76% of survey respondents use more than one cloud. The larger the organization, the more likely they are to use multiple clouds.[11]



Picture 3. The percentage of companies that has adopted multi-cloud vs. singlecloud in a cloud environment. [11]

When choosing a cloud platform, it is good to check the pricing for different cloud operators. Each provider has vastly different prices for their solutions. Comparing cloud providers computing prices with Linux and Windows machine options, there are great differences in prices.

Prices for three top cloud platforms are shown on upcoming tables 4-6. Tables show monthly cost of on-demand computing with Linux and windows operating system. All systems were configured for 2 vCPU and 8 GB of RAM using operators own pricing calculators. Table 4 shows Amazon web services monthly cost on Europe’s servers.

Table 4 AWS prising for Linux and Windows machine. [13]

Region	Service	Upfront	Monthly (USD)	Configuration summary
Europe (Frankfurt)	Amazon EC2	0	38.90	Operating system (Linux), Quantity (1), Pricing strategy (EC2 Instance Savings Plans 1 Year No Upfront), Storage amount (30 GB), Instance type (t4g.large)
Europe (Frankfurt)	Amazon EC2	0	63.43	Operating system (Windows Server), Quantity (1), Pricing strategy (EC2 Instance Savings Plans 1 Year No Upfront), Storage amount (30 GB), Instance type (t3a.large)

Table 5 Azure pricing for Linux and Windows machine [14]

Region	Service	Upfront	Monthly (USD)	Configuration summary
North Europe	Virtual Machines	0	66.48	1 B2ms (2 vCPUs, 8 GB RAM) x 730 Hours (Pay as you go), Linux, (Pay as you go); 0 managed disks – S4, 100 transaction units; Inter Region transfer type, 5 GB outbound data transfer from North Europe to East Asia
North Europe	Virtual Machines	0	72.466	1 B2ms (2 vCPUs, 8 GB RAM) x 730 Hours (Pay as you go), Windows (License included), OS Only; 0 managed disks – S4, 100 transaction units; Inter Region transfer type, 5 GB outbound data transfer from North Europe to East Asia
North Europe	Virtual Machines	0	66.48	1 B2ms (2 vCPUs, 8 GB RAM) x 730 Hours (Pay as you go), Windows (AHB), OS Only; 0 managed disks – S4, 100 transaction units; Inter Region transfer type, 5 GB outbound data transfer from North Europe to East Asia

Table 6 GCP prising for Linux and Windows machine [15]

Region	Service	Upfront	Monthly (USD)	Configuration summary
Finland	Compute Engine	0	53.86	e2-standard-2 (vCPUs:2, RAM: 8GB) Linux
Finland	Compute Engine	0	121.02	e2-standard-2 (vCPUs:2, RAM: 8GB) Windows
Finland	Persistent Disk	0	3.30	30 GiB

The prices on the tables 4, 5 and 6 show on-demand pricing. Cloud providers also offer spot pricing that is usually up to 90% cheaper than on-demand pricing. It uses providers unused computing space. The downside of spot computing is that these instances can be shut down unexpectedly. For spot priced computing their best use cases are batch workloads that can handle interruption. [16]

3.5 Readiness for the cloud

Readiness for the cloud describes the state where the current service or software is for migration towards cloud.

- Non-cloud
- Cloud-friendly
- Cloud-ready
- Cloud-native

Non-cloud consists mostly of legacy workloads that do not work well towards clouds' new innovative solutions. Not all on-premises deployments can be moved to cloud. If the workloads show signs of ability towards cloud-readiness, then it could be considered to move to cloud. Non-cloud workloads can be expensive to migrate on cloud. Sometimes it's better to retain these on on-premises or thinking to replace the deployment for more modern approach. [1]

Cloud-friendly deployments are not designed for cloud-ready architecture. These deployments can be modified or refactored to be cloud ready. Deployments that can be decoupled form its compute, storage, and network are close to being cloud-friendly workload. [1]

Cloud-ready If on-premises workloads have a cloud-ready architecture which contains observability, fault tolerance, interoperability, portability, security, and scalability, they are ready for cloud migration. [1]

Cloud-native applications are implemented with microservice architecture to implement most of the design principles of cloud-ready characteristics. The applications that are utilizing microservice architecture suits for rehosting them on cloud provider’s cloud. [1]

3.6 Migration strategies

Migration strategies have standardized strategies that give clear understanding of how specific application, service, server could be migrated to cloud. These strategies have become standard for cloud migration and are explained next.

- Rehost
- Re-factor
- Re-platform
- Replace
- Re-architect
- Relocate
- Retain
- Retire
- Re-innovate

Rehost uses cloud providers IaaS or PaaS services to make a virtual machine that runs the specified load. Rehosting can be more expensive than on-premises as it does not utilize cloud providers cloud readiness. This is the easiest way to move application to cloud but without the cost savings that cloud provider give when optimized. [1, 17]

Re-factor Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and databases for [1, 17]

Re-platform Move is an application to cloud and it makes some optimizations to take advantage of clouds' advantages. It does not move everything but utilizes optimisations of a few key parts to make use of cloud benefits. As it is not full refactoring, this deployment does not need a lot of expertise as re-factoring. [1, 17]

Replace is an early stage of cloud migration for enterprises that are at an early stage of their cloud journey. Every step, they are learning and reinventing themselves. Therefore, it is common for enterprises to just drop existing platforms and rebuild their workloads on a public cloud environment. [1, 17]

Re-architect can be used when old service does not meet planned requirements for company's cloud infrastructure. It is rebuilt from scratch to cloud infrastructure utilizing more cloud native ways. [1, 17]

Relocate's strategy is to move on-premises workloads to cloud. Its difference with re-host is ensures some optimization and configuration changes. Relocates goal is just move it quickly to cloud. [1, 17]

Retain Enterprises often have legacy applications with complex dependencies that require major refactoring or are no longer valid for business cases. Retain is a suitable strategy for such workloads where they are identified and kept in the source environment for further processing. [1, 17]

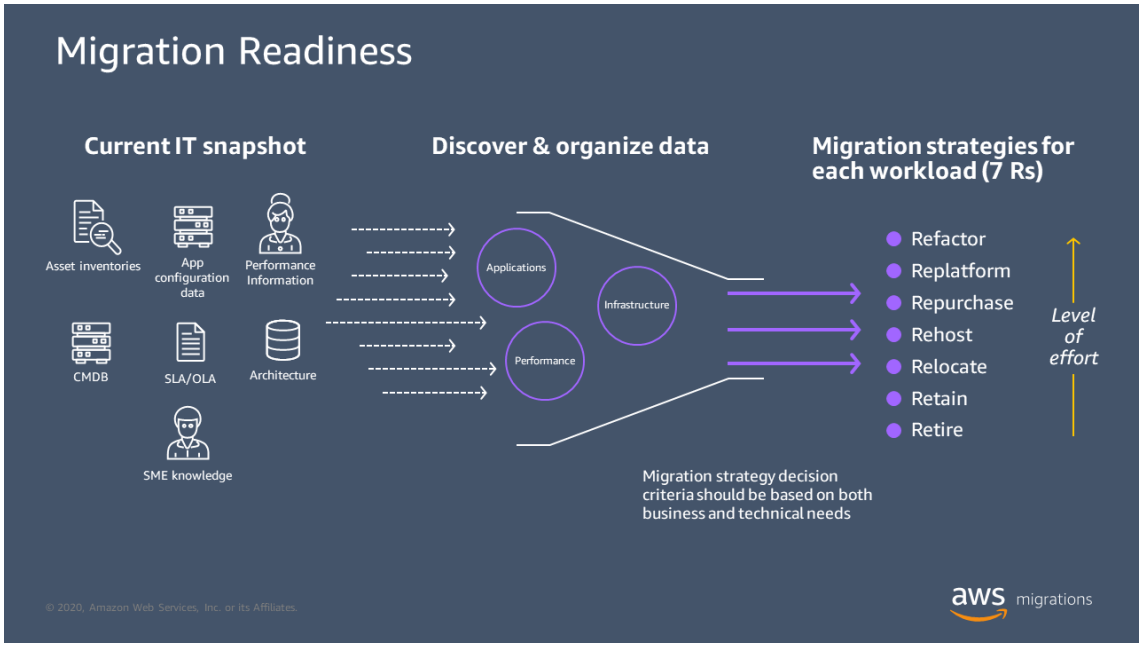
Retire Enterprises adopt this strategy to decommission legacy workloads that no longer have business requirements. In most cases, if an existing workload is not compatible for containerization or refactoring, it can be a suitable candidate for the Retire pattern – for example, they migrated the physical server on-premises to a VM on the cloud. [1, 17]

Re-innovate in some cases is used by enterprises to benefit the opportunity of cloud migration to re-invent themselves. For example, during the discovery

phase of cloud migration organizations might find out that AI SaaS solutions or workflow engines can replace some applications. [1, 17]

With these strategies and cloud readiness you can find out how you would implement cloud migration case by case. It is good think about the cost of moving different pieces of software. Luckily cloud providers have various kinds of calculators for cost of operating on cloud. There does not have to be only one strategy for a service because when the migration happens, it can even change during it.

AWS has seven of these migration strategies in their best practices that can be seen on their migration readiness in picture 4.



Picture 4. AWS migration readiness chart. [16]

It is not unusual to find that more than 10 percent of workloads in an enterprise IT portfolio are no longer useful and can be turned off, thereby creating savings for your organization. [2]

4 Migration project

The purpose of this thesis project was to move a data system to cloud. Firstly, packer, or a Hashicorps product to build automated images was studied. After that the focus of this study was on how to implement ansible for packer for using its automation for installing software packages.

Packer can be easily deployed in production environment and its greatest quality is to make immutable images to multiple cloud and hypervisor providers. All you need is a base image from provider or providers that you use because packer uses providers base image as the start. When choosing and configuring the base image ansible was utilised to install predefined packages.

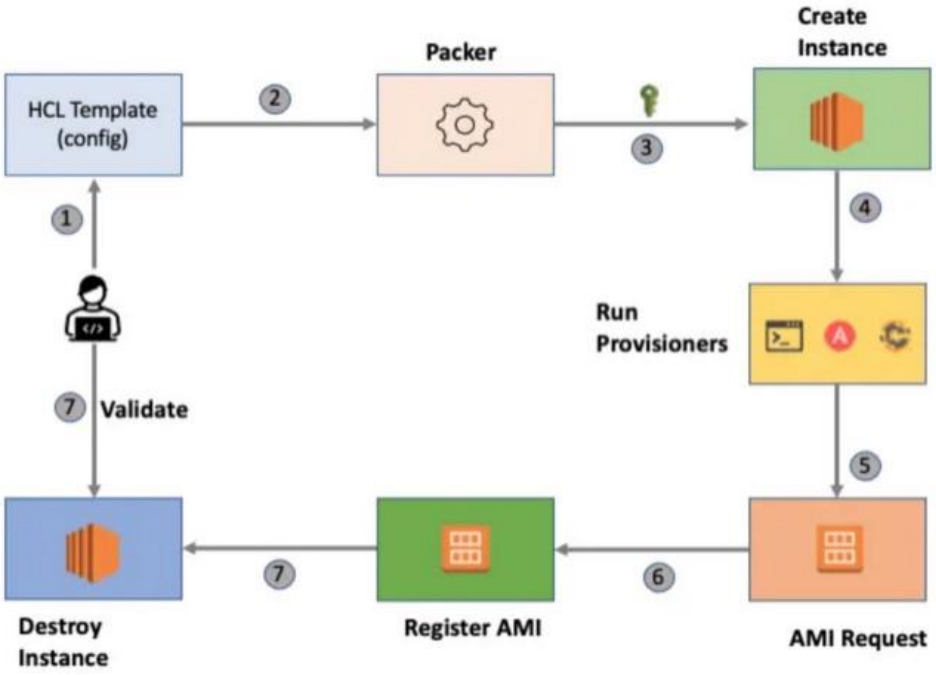
In packer to use AWS ecosystem you need to use Hashicorps plugin called amazon. They have ready built plugins for most cloud and hypervisor providers.

In this project the idea was that the images would easily be updated. The versions for the software used and the provider for the software and further configurations chosen was ansible as it was already used in the company. Ansible was already well built for manually commanding. The biggest problem with using ansible in packer was that in packer ansible did not support group variables. Packer does not support jinja2 macro syntax in ansible provider. These incompatibles with different roles when the trying to implement them in packer are discussed in ansible playbook.

4.1 Packer

Packer software is designed for making machine images for cloud environments. With packer making machine images it is easy as it will be explained bellow. Software supports utilization of modules for providers because it gives great flexibility for deployment.

Packers' operation can be seen as visualized in picture 5. The first step is committing the HCL template which contains configuration steps for machine images. The second step is initializing the needed modules for the preparation when packer has started the building proses. When packer has made the required prerequisites to start build, it creates ssh keys for connecting the instance that it configures. When the instance is created packer starts the provisioners declared on HCL template and runs them to instance. When the provisioner has made the changes, the packer starts the AMI request for registering it. When the registered packer destroys the instance and the AMI stays on registered cloud provider to be implemented for its required use.



Picture 5. Describing graph of packers work model on AWS. [18]

Connecting packer to AWS needs credentials and you can use your own account for it. Packer provides a YAML configuration file for necessary access for creating an image (see Appendix 1). With those permissions packer can create and destroy ec2 instances and make necessary operations for making registered image on AWS image registry.

Packer uses base images to build its own machine images. These images are provider specific as packer uses providers existing images as its base.

To configure what type of instance to use during a building phase packer HCL declares a source block. In source block you start by configuring the first instance built for the packer. With information of underlaying machine image, its storage size and who can access the finished build will be configured. Without specifying the user, the only who can use the finished packer image is the account owner where the packer was connected on. For AWS access control, a YAML file that contains minimum requirements for the packer accessing your enterprise's AWS environment was included (see Appendix 1).

Packers supports various cloud providers and hypervisors for its source to build those images. Officially packer currently supports a list of cloud providers and hypervisors which are listed below.

Here is a list of officially supported cloud providers and hypervisors. These are in Hashicorps management.

Official Providers [19]

- AWS
- Azure
- Docker
- Google cloud Platform
- VMware
- vSphere
- VirtualBox
- QEMU

With community and verified support there are a lot more different cloud providers and hypervisors. Verified plugins are owned and maintained by a third

party that are also a member of the HashiCorp Technology Partner Program. Those owned by communities are from individuals, groups or companies that are not part of the partner program. Hashicorp provides instructions for making custom source providers for their product. Listing 1 shows how plugins are getting enabled for packers to use. [19]

```
packer {
  required_plugins {
    amazon = {
      version = "0.0.1"
      source  = "github.com/hashicorp/amazon"
    }
  }
}
```

Listing 1. A HCL plugin to utilize AWS provider.

Making the packer utilize a selected cloud provider or hypervisor the configuration of the base image must be defined on the packer's source block. The source block defines what communication the packer uses when creating the image and what type of VM is used during the build phase. The first configuration must be done on source block when it comes to first user and its password. On source block the users who can access the finished image should be defined, if not only the account that created the image can utilize it. These source block configurations can be seen on listing 2.[20]

```
ami_name      = "windows"
  communicator  = "winrm"
  instance_type = "t2.small"
  region        = "eu-central-1"
  source_ami_filter {
    filters = {
      name                = "windows-image"
      root-device-type    = "ebs"
      virtualization-type = "hvm"
    }
    most_recent = true
    owners      = ["amazon"]

  }

  tags = {
    build = packer

  }
  ami_users =  [
     xxxxxxxxxx
    ]
  user_data_file = "./bootstrap_win.txt"
  winrm_password = "xxxxxxxxxx"
  winrm_username = "xxxxxxxxxx"


}
```

Listing 2. Example of source block for windows instance on packer

In build block of packers, an HCL file contains all configurations that you want to make when the image is running. The source is also defined on a build block as the packer can build multiple images at the same time. When it has been defined which source is going to be used on a specified build step, the next step is that packers build a block to define its provisioners for building the wanted server.

Provisioners of packer are the software that connect to build a virtual machine and configure it. The provisioners officially supported by the packer are listed below.

Official provisioners [19]

- file
- PowerShell
- Shelll
- Windows Shell
- Ansible
- HashiCups

Provisioners follow the same principle as providers when it's come for their support. Provisioners have official, verified and community tiers of support for the modules. Other popular provisioner but are community supported currently are Puppet and Chef. These provisioner provide the means to installing packages, patching the kernel, creating users, and downloading application code. Build block can look simple when looking it as shown on listing 3. [21]

```
build {
  sources = [
    "source.amazon-ebs.windows"
  ]


  provisioner "powershell" {
    script = "./ConfigureRemotingForAnsible.ps1"
  }

  provisioner "ansible" {
    playbook_file       = "./playbook.yml"
    roles_path          = "./roles"
  }
}
```

Listing 3. Example of packer build block utilizing powershell and ansible.

Additional configuring for packer is utilization of post-processors. Post-processors are run after packer has built the images. With post-processors you

can create artifact of the builds which can be used to build the image. Packer currently supports the post-processors listed below.

Official post-processors [19]

- Vagrant
- vSphere
- docker
- google cloud platform
- amazon

Post-processors follow the same principles as providers and provisioners.

With post-processors the artifice build is in supported form to make local build of the build. For example, the vagrant that is Hashicorps product with utilizing its post-processor it can create working virtual machine of the image build on packer. [22]

4.2 Ansible

The provisioner chosen for packer is ansible for its earlier use in the company to prevent any difficulties. Ansible is currently the only supported provisioner application on packer but there are others that are made available through community support.

Ansible is easy to use because it uses no agents and no additional custom security infrastructure. It is easy to deploy - and most importantly, it uses a very simple YAML language that allows you to describe your automation jobs clearly. [23]

Installation of software with ansible is done in easy-to-understand manner. As can be seen in listing 4, a windows application is implemented on ansible. With ansible managing the software and settings, it is simple to see where the installation files are located and installed.

```
- name: Copy Notepad++ installer to destination
  win_copy:
    src: {{sourcelocation}}/{{ notepad_installer_name }}
    dest: "{{ copy_destination }}"

- name: Install Notepad++
  win_package:
    path: "{{ copy_destination }}{{ notepad_installer_name }}"
    product_id: "{{ notepad_product_id }}"
    state: "{{ notepad_state }}"
    arguments: "{{ notepad_install_arguments }}"
```

Listing 4. The example of ansible role that installs notepad++

Ansible can control all software installations and modify settings on a windows device. When using ansible to make the major configurations the packer HCL only had to configure access for ansible. The software packages needed were stored to s3 for greater accessibility meaning they should not be used on a laptop. To utilize s3 for directory of software was done on ansible as it has greater support and has been on the market longer than packer meaning it was familiar to us. The steps how to get the files from s3 are shown in listing 5.

```
- name: Get list of files from S3
  aws_s3:
    mode: list
    bucket: {{bucket}}
    prefix: "Software/"
    marker: "Software/"
  register: s3_bucket_items
  delegate_to: localhost

- name: Download files from S3
  aws_s3:
    mode: get
    bucket: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    object: "{{ item }}"
    dest: "/Software/{{ item|basename }}"
  with_items: "{{ s3_bucket_items.s3_keys }}"
  delegate_to: localhost
```

Listing 5. Code snippet for downloading installer files from AWS s3 bucket.

Files had to be downloaded to runner so that ansible can then send them to instances was the most straightforward solution for getting files to AWS cloud virtual machines. The way ansible gets the file form s3 is convoluted. As AWS s3 implementation of storing files is object based. Getting software from s3 you need to first list the contents for downloading.

Ansible has created a community for supporting different kinds of deployments. The community creates templates and better solutions to access different vendors ecosystems more straightforward than creating those mechanisms yourself. [24]

4.3 Ci/cd piping

The piping for this project was done with GitHub runners. As a result, when you update the triggering function of the repository packer image which contains the packer configurations and ansible code snippets, this will result in building a pipe packer according to instructions presented in listing 6. The pipe of this packer is easy because the steps are checked to ensure the variables are right. The next step was to initialize it and download the required packer

dependencies before the build. The last step was to build an image specified on the run command.

```
name: Deploy Packer image

on:
  push:
    branches:
      - "main"

jobs:
  deploy:
    name: Deploy with Packer
    runs-on: ubuntu-20.04
    steps:
      - name: Packer - Init
        run: docker-compose run packer init aws-win.pkr.hcl

      - name: Packer - Build
        run: docker-compose run packer build aws-win.pkr.hcl
```

Listing 6. Github runner code snippet to build packer image

To utilize GitHub runner, a Dockerfile with required steps for building packer with ansible had to be built. The source device was Windows. Listing 7 shows the specified packages for it.

```
FROM hashicorp/packer:latest
RUN apk update
RUN apk add --no-cache ansible py-pip openssh \
    && apk add --no-cache --upgrade apk-tools \
    python3 \
    libcurl \
    openssl \
    sshpass \
    curl
RUN pip3 install pywinrm[credssp]
RUN ansible-galaxy collection install community.windows
RUN ansible-galaxy collection install amazon.aws
RUN pip3 install boto3 botocore

ENTRYPOINT ["/bin/packer"]
```

Listing 7. Dockerfile with needed dependencies for packer builder with ansible.

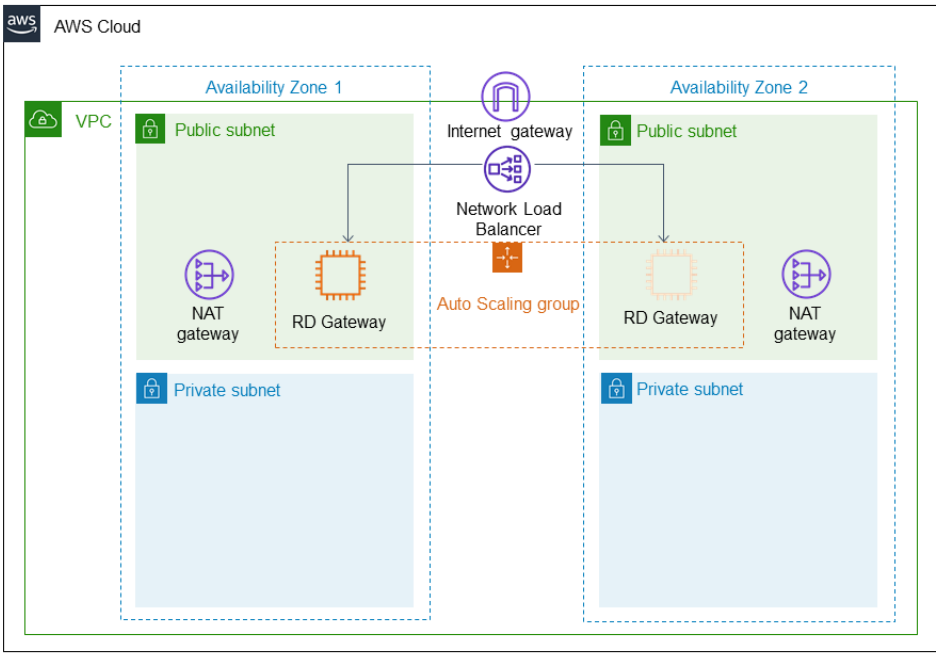
Dockerfile was created which uses packers docker container as its base. This ensured that the packer keeps up to date when using a supported container.

Modifications made on container were installing ansible and support needed for the modules of the project.

4.4 Rd gateway

Rd gateway is Microsoft’s remote desktop protocol gateway meaning a Windows device that controls RDP traffic between public and private subnets on AWS. The Rd gateway uses RDP over HTTPS protocol to establish a secure, encrypted connection between remote users and ec2 instances running Microsoft Windows, without needing to configure a virtual private network. When using Rd gateway, it allows secure connection for inspecting the packer image that was made for the deployment.

AWS provides deployments that have already been made for to use in their cloud environment which is a AWS proprietary iac provisioner. For the environment that it was built cloud formation was not used; instead a terraform for making the infrastructure what Rd gateway needed was utilized. Below there is a complete architecture picture of Rd gateway deployment which can be seen in picture 6. [25]

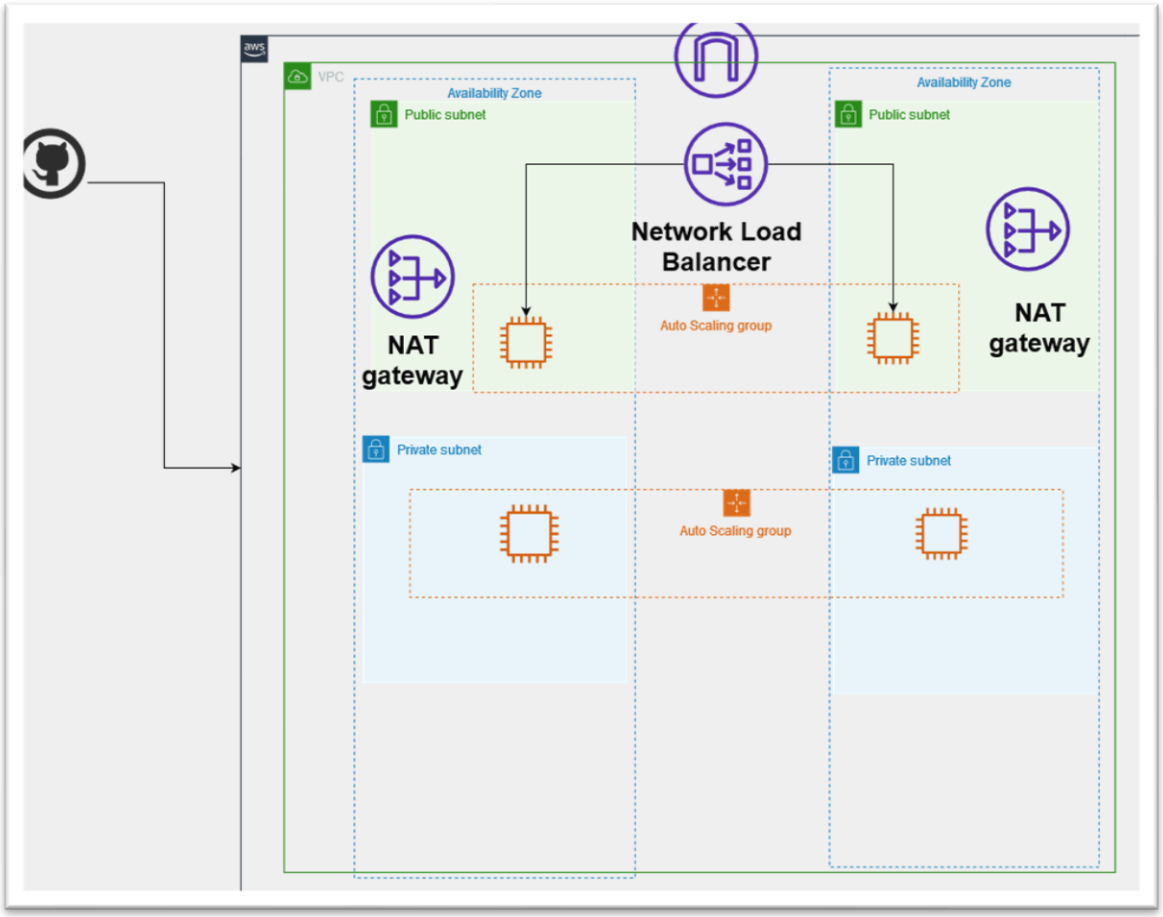


Picture 6. Rd gateway architecture image.[20]

4.5 Scaling

The scaling is configured on terraform to use AWS auto scaling group, which scales ec2 instances on AWS cloud. Utilizing auto scaling group saves significant numbers of resources compared to on-site deployment for its automated detection for growing needs when using applications that are in auto scaling group. The scaling can be triggered from resources utilization or through automating certain limitations when met with scales up or down. Auto scaling group also provides rolling updates trough scaling new ones and commissioning older versions out.

When implemented scaling the full deployment of the project could be shown. The completed architecture diagram can be seen in picture 7. [26]



Picture 7. Architecture image of build system in AWS.

5 Challenges

During the project there were a few problems. The biggest issue was with packer AMI images that had something wrong on ansible code as it would take sometimes more than 40 minutes to build them. During this building time there was little to do as you would not see the specific problem until that was done and it could be something as small as a writing error or the needed variables were not available as the packer did not support jinja2 macros on ansible provider. To fix this problem you needed to move variables from group variables to needed roles of variable folders and find if there were more additional variables still needed.

Rd gateway was added to the project because of a problem that occurred when the implantation of security policies. The security polices and the locking them under private subnets. The access for packer build machine images weren’t reachable for looking what changes were made and it is good for security. But to see the result the need to build gateway that controls access for the machines without making them public to internet.

6 Conclusion

The purpose of this project was to create a migration from a data centre to AWS cloud by using a packer. As a result of this study, the migration was successfully created. However, I did not see the end product to be used in production as my project was to make the pipeline to cloud. Based on this study it can be said that there are many ways to make companies more cloud friendly.

In the future, there will be an increasing need for cloud services. This project provided a good steppingstone for the company’s cloud migration. Its further development would transform the service for cloud native approach to utilize cloud providers greater savings and optimizations.

References

1 Mansura Habiba; Mihai Criveti. 2022. Hybrid Cloud Infrastructure and Operations Explained. Packt Publishing. <https://learning.oreilly.com/library/view/hybrid-cloud-infrastructure/9781803248318/B18191_01_ePub.xhtml#_idParaDest-15>. Accessed 21 October 2022.

2 Evgeniy Altynpara. 2021. Why Migrate To The Cloud: The Basics, Benefits And Real-Life Examples https://www.forbes.com/sites/forbestechcouncil/2021/03/12/why-migrate-to-the-cloud-the-basics-benefits-and-real-life-examples/. Accessed 21 October 2022.

3 Azure. 2022. SLA for Virtual Machines. https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_9/. Accessed 25 October 2022.

4 Google Cloud Platform. 2022. Compute Engine Service Level Agreement (SLA). https://cloud.google.com/compute/sla. Accessed 25 October2022.

5 Amazon Web Services. 2022. AWS Service Level Agreements. https://aws.amazon.com/compute/sla/?did=sla_card&trk=sla_card. Accessed 25 October2022.

6 Redhat. 2022. What is SaaS? https://www.redhat.com/en/topics/cloud-computing/what-is-saas. Accessed 21 October2022.

7 Graeme Messina. 2021. 4 Cloud Deployment Models with Examples: Public, Private, Community, Hybrid. https://www.cbtnuggets.com/blog/certifications/cloud/4-cloud-deployment-models-with-examples-public-private-community-hybrid. Accessed 24 October 2022.

8 Raj Bala; Bob Gill; Dennis Smith; David Wright; Kevin Ji. 2021. Magic Quadrant for Cloud Infrastructure and Platform Services. https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb. Accessed 25 October 2022.

9 Van Baker; Svetlana Sicular; Erick Brethenoux; Arun Batchu; Mike Fang. 2021. Magic Quadrant for Cloud AI Developer Services. https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb. Accessed 25 October2022.

10 Hashicorp. 2022. Scale Your Cloud Operating Model with a Platform Team https://www.hashicorp.com/cloud-operating-model. Accessed 25 October2022.

11 Frederico Hakamine. 2021. What Are the Benefits and Limitations of Multi-Cloud? https://www.okta.com/blog/2021/05/what-are-the-benefits-and-limitations-of-multi-cloud/. Accessed 25 October2022.

12 Acronis. 2020. Top 10 benefits of multicloud. https://www.acronis.com/en-us/cyber-protection-center/posts/top-10-benefits-of-multi-cloud/. Accessed 25 October2022.

13 Amazon Web Services. 2022. AWS Pricing Calculator. https://calculator.aws/#/addService?nc2=pr&refid=ap_card. Accessed 26 October2022.

14 Azure. 2022. Pricing calculator. https://azure.microsoft.com/en-us/pricing/calculator/. Accessed 26 October2022.

15 Google Cloud Platform. 2022. Google Cloud Pricing Calculator. https://cloud.google.com/products/calculator. Accessed 27 October2022.

16 Amazon Web Services. 2022. Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot/pricing/. Accessed 27 October2022.

17 Amazon web Servises. 2022. Best practices for assessing applications to be retired during a migration to the AWS Cloud https://docs.aws.amazon.com/prescriptive-guidance/latest/migration-retiring-applications/overview.html. Accessed 27 October2022.

18 Bibin Wilson. 2021. Immutable VM Image Building Using Hashicorp packer https://devopslearners.com/immutable-vm-image-building-using-hashicorp-packer-5b0189634d43. Accessed 21 October2022.

19 hashicorp. 2022. Plugins https://developer.hashicorp.com/packer/plugins. Accessed 27 October2022.

20 hashicorp. 2022. Builders https://developer.hashicorp.com/packer/docs/builders. Accessed 27 October2022.

21 hashicorp. 2022. Provisioners https://developer.hashicorp.com/packer/docs/provisioners. Accessed 27 October2022.

22 hashicorp. 2022. Post-Processors https://developer.hashicorp.com/packer/docs/post-processors. Accessed 27 October2022.

23 Ansible. 2022 How Ansible works https://www.ansible.com/overview/how-ansible-works. Accessed 27 October2022.

24 Ansible. 2022. Galaxy Documentation. https://galaxy.ansible.com/docs/. Accessed 27 October2022.

25 Amazon Web Services. 2022. Remote Desktop Gateway on AWS https://aws.amazon.com/quickstart/architecture/rd-gateway/#. Accessed 27 October2022.

26 Amazon Web Services. Amazon EC2 Auto Scaling. 2022. https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html. Accessed 27 october 2022.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AttachVolume",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CopyImage",
        "ec2:CreateImage",
        "ec2:CreateKeypair",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSnapshot",
        "ec2:CreateTags",
        "ec2:CreateVolume",
        "ec2:DeleteKeyPair",
        "ec2:DeleteSecurityGroup",
        "ec2:DeleteSnapshot",
        "ec2:DeleteVolume",
        "ec2:DeregisterImage",
        "ec2:DescribeImageAttribute",
        "ec2:DescribeImages",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeRegions",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSnapshots",
        "ec2:DescribeSubnets",
        "ec2:DescribeTags",
        "ec2:DescribeVolumes",
        "ec2:DetachVolume",
```

```
        "ec2:GetPasswordData",
        "ec2:ModifyImageAttribute",
        "ec2:ModifyInstanceAttribute",
        "ec2:ModifySnapshotAttribute",
        "ec2:RegisterImage",
        "ec2:RunInstances",
        "ec2:StopInstances",
        "ec2:TerminateInstances"
      ],
      "Resource": "*"
    }
  ]
}
```