* Packer is an open source tool that allows us to create a custom images across a multitude of platforms
* Mutable means to change while immutable is unchanged
* In a mutable infrastructure when we deploy the server we have to configure certain things such as patching the server, updating the repos etc. This means that this infrastructure changes. This happens in the traditional model. Down the road if you need to make any changes you need to mutate it once again. This works if you have one server. However, the real problem happens when you have multiple servers. Config drift can happen here with some servers not getting the configurations that they need while others do.
* There are automation tools built for this like ansible which helps with preventing configuration drift.
* So to recap in mutable infrastructure we go from development to deployment and then configuration

***IMMUTABLE INFRASTRUCTURE***

* In immutable infrastructure we configure the server before we deploy it.This is where packer comes into play.
* With packer we can build images that already have all the necessary requirements for the server you are trying to deploy.
* Packer can help create a custom AMI
* With an immutable infrastructure (like packer) we can take the application code and configuration and bake them into the image.
* So all we do is deploy the image into the server and the server is production ready
* That’s why its called immutable so we cant change it. However, if we need to make some changes down the road we simply kill that server and deploy a new server.  We can avoid down time in this case by spinning up a load balancer which send traffic to the other servers.

**S*ETTING UP PACKER ON WINDOWS, LINUX AND MACOS***

* To setup packer you go to [packer.io](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Furldefense.com%2Fv3%2F__http%3A%2F%2Fpacker.io__%3B!!CdHzsg!gisqTbsb_uMXUAnWzYhWGNgZ3J2N9xmR3dGBAh5fpGU0h7tTouLZMd3wX1-m9tFnNlV4TGrloHUDREwl3RhsTtSP0HuBsZEgOoc0%24&data=05%7C01%7Cbrigthain.kargong%40gainwelltechnologies.com%7Cdb7338c1c16f46a3861e08da52f5873e%7Cc663f89cef9b418fbd3d41e46c0ce068%7C0%7C0%7C637913511735028723%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=7LngjvKisdyG3ga0UzKuAqoq1Of%2F2oCfABfaIjgOXFc%3D&reserved=0) and download from there
* Installed in mac using homebrew
* To install homebrew go to google and type homebrew. Proceed to the web browser and copy the link in the front page and paste on the CLI and the process should start downloading all the necessary binaries you need.
* After homebrew is installed simply run brew install packer. If the installation for packer don’t work using the above method follow this link to help with installation [https://stackoverflow.com/questions/64963370/error-cannot-install-in-homebrew-on-arm-processor-in-intel-default-prefix-usr](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Furldefense.com%2Fv3%2F__https%3A%2F%2Fstackoverflow.com%2Fquestions%2F64963370%2Ferror-cannot-install-in-homebrew-on-arm-processor-in-intel-default-prefix-usr__%3B!!CdHzsg!gisqTbsb_uMXUAnWzYhWGNgZ3J2N9xmR3dGBAh5fpGU0h7tTouLZMd3wX1-m9tFnNlV4TGrloHUDREwl3RhsTtSP0HuBsbe2iFjy%24&data=05%7C01%7Cbrigthain.kargong%40gainwelltechnologies.com%7Cdb7338c1c16f46a3861e08da52f5873e%7Cc663f89cef9b418fbd3d41e46c0ce068%7C0%7C0%7C637913511735028723%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=hKTHRj0pRZ44yR3YnBf0cs9fuc3tnKlURmrNjA%2Fnszg%3D&reserved=0)

***BUILDING YOUR FIRST AMI***

* First you have to create a directory to store all our source code mine is packer and im using vs code as my text editor
* Packer uses a json format
* First.json will contain all the configurations that you need for packer
* A packer config has 3 building blocks: builders, Provisioners and post processors

***BUILDERS***

* They are responsible for creating a machine and generating images from them across various platforms. Packer supports a ton of platforms
* These are the main components for packers
* Go to the packer home page and select builders on the left pane which shows all the documentations that we will need. Go to [https://www.packer.io/plugins/builders/amazon/ebs](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Furldefense.com%2Fv3%2F__https%3A%2F%2Fwww.packer.io%2Fplugins%2Fbuilders%2Famazon%2Febs__%3B!!CdHzsg!gisqTbsb_uMXUAnWzYhWGNgZ3J2N9xmR3dGBAh5fpGU0h7tTouLZMd3wX1-m9tFnNlV4TGrloHUDREwl3RhsTtSP0HuBsdKJQGBL%24&data=05%7C01%7Cbrigthain.kargong%40gainwelltechnologies.com%7Cdb7338c1c16f46a3861e08da52f5873e%7Cc663f89cef9b418fbd3d41e46c0ce068%7C0%7C0%7C637913511735028723%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=xzjWDN6wysG5BJGkxk0b88IxUVisYMEkOb77KKr%2Fi04%3D&reserved=0) for all the necessary info on AMI creation
* Packer allows you to pass multiple builders so go to your text editor and in in-between the curly braces type the word builders in parenthesis. See example below
* You need to provide keys for your aws environment. To find access information go to your aws account and select your username and go to security credentials. Here you should be able to see your access key and secret access key.
* Access key for me is: AKIAXB5IN5PEKWT5M5OU
* Secret access key is: SY4m3B/C/8Y7xMS0WwM9xUJT7juWPrMChmo/hhzb
* Region is us-east 1
* We are going to hard these now but will move them someplace else later
* Under AMI configuration one requirement is the ami\_name
* There's other properties that you can put in but the name is the only one that’s required all the others are optional
* We have to give packer an ami that we want to tweak to create ours
* Go to your console to get one that you want to tweak
* We need an instance type because packer creates an instance to be able to create the AMI you want. However, this instance is deleted by packer after the AMI is created. So it’s a temporary instance
* Next you provide the ssh\_username for packer to log in to that instance and create the ami. Default user name is usually ec2-user but ubuntu is ubuntu for ubuntu VMs

{

    "builders": [

        {

            "type": "amazon-ebs",

            "access\_key": "AKIAXB5IN5PEKWT5M5OU",

            "secret\_key": "SY4m3B/C/8Y7xMS0WwM9xUJT7juWPrMChmo/hhzb",

            "region": "us-east-1",

            "ami\_name": "bk-ami",

            "source\_ami": "ami-0022f774911c1d690",

            "instance\_type" "t2.micro",

            "ssh\_username" "ec2-user"

        }

    ]

}

* Above is the example for base configuration to create an AMI
* Now go to your terminal in either VS code or the CLI on your local computer and navigate to the folder you are working on. Mine is first.json.
* Now run the command *packer build plus project name*to build the ami and let it run
* In the output during the launching you see that it launches an instance and then later deletes it
* Now if you go to the console under AMI's you will see your newly created AMI and you can use that AMI to launch an EC2-instance
* To terminate an AMI go to the AMI tap, select the AMI and go under actions and deregister the AMI

***PROVISIONERS***

* This is the second aspect of a packer configuration and where the customizabilitycomes into play
* This is where we can install and configure our machine image
* They use built-in and third party software to install and configure the machine image after booting. They prepare the system for use.
* There are several different types of provisioners and it supports automation tools like ansible, chef, puppet etc
* We are using shell provisioners here
* We are using an ubuntu ami here
* Here we will use a provisioner to install nginx into our AMI
* To create a provisioner we run like we are running the builder in json format
* See below for example
* The provisioner has the type and the inline property for shell
* To avoid packer from instantly SSHing into the box upon reboot and start making config changes to the server without the rest of the operating system being up. Its best to set a *sleep x*( in which x is any number of your choosing) so that packer waits for some time for the system to properly be up before it starts making any changes

{

    "builders": [

        {

            "type": "amazon-ebs",

            "access\_key": "AKIAXB5IN5PEKWT5M5OU",

            "secret\_key": "SY4m3B/C/8Y7xMS0WwM9xUJT7juWPrMChmo/hhzb",

            "region": "us-east-1",

            "ami\_name": "Bk-ubuntu-ami-project-2",

            "source\_ami": "ami-09d56f8956ab235b3",

            "instance\_type": "t2.micro",

            "ssh\_username": "ubuntu"

        }

    ],

    "provisioners": [

        {

            "type": "shell",

            "inline":[

                "sleep 30","sudo apt update","sudo apt install nginx -y"]

        }

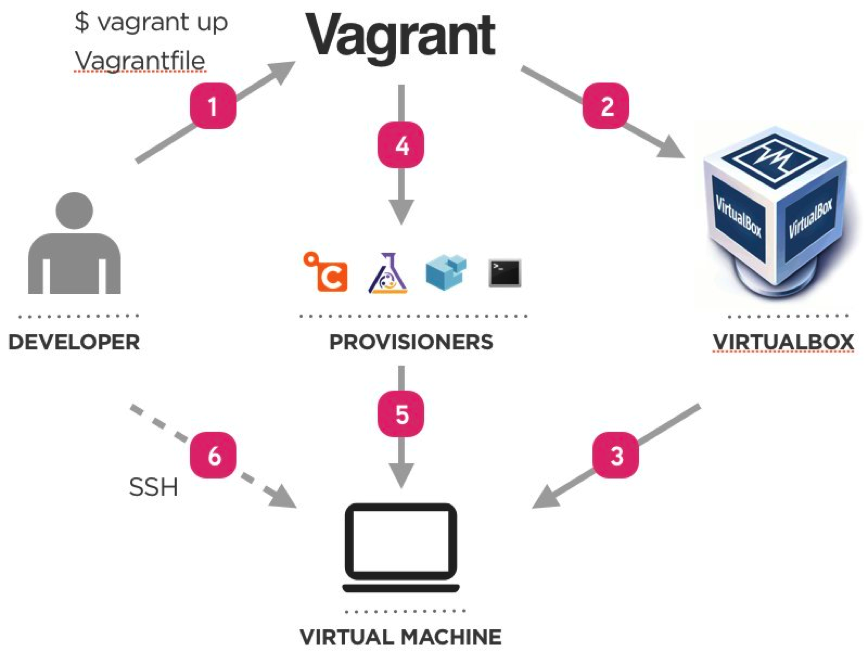
    ]

}

* Make sure that you bake as much as possible in the image to avoid doing any manual configurations
* Make sure to setup firewalls, web applications etc
* Ufw stands for uncomplicated firewall
* Instead of having to type multiple commands like the one in project 2 in the inline command you can use a script which makes everything more neat. So in this case you create a new file where you will input the script. In our case its called setup.sh
* In our example the setup.sh script is in the same directory as the json file hence we can just pass in the script name. however if the script was stored anywhere else in our OS we will have to pass in the full path
* After running the command and provisioning the AMI, create an instance from that AMI and check nginx status by typing systemctl status nginx. It should say nginx is active
* You can go to the nginx browser by editing the security group for that instance and adding port 80 and 443
* Now copy the public IP address and paste it in a web browser and you should see the nginx welcome page
* To not see the default web page but see our own custom web page go to the etc/nginx/sites-availabledirectory
* Now cat into the default file and there you will see where the html file is stored which is usually root/var/www/html
* Now go to that directory
* Now to have our own website there instead of the welcome nginx page we will use a provisioner that will allow us to copy an html file from our local machine unto the EC2 instance
* We will use the file provisionerin this case
* Here we will be working in project 4
* We can have an array of provisioners so here we will be adding the file provisioner to the project
* Now create another file for the index.html file creation
* To create a boiler plate for an html file in VS code simply type html and select the html:5 which gives you a template for an html file
* In this case the first try failed cause the ubuntu user didn’t have permissions in the var/www/index directory. However, all users have full access in the /tmp directory so we can either use that or we can change ownership in the var/www/html directory to ubuntu
* The file provisioner doesn’t give us sudo privileges so after creating the index file in the /tmp directory we can use the shell provisioner to cp the file from the tmp directory to the var/www/html directory

***POST-PROCESSORS***

* Post processors run after the image is built by the builder and provisioned by the provisioner(s). They are optional, and they can be used to upload artifacts, re-package, or more.
* We can create a vagrant boxes with post processors
* We used manifestand vagrantpost processors
* The manifest post-processor writes a JSON file with a list of all of the artifacts packer produces during a run. If your Packer template includes multiple builds, this helps you keep track of which output artifacts (files, AMI IDs, docker containers, etc.) correspond to each build.
* The manifest post-processor is invoked each time a build completes and *updates* data in the manifest file. Builds are identified by name and type, and include their build time, artifact ID, and file list.
* With manifest every time a build is run the new build artifacts will be added to the manifest file rather than replacing it.
* The Packer Vagrant post-processor takes a build and converts the artifact into a valid [Vagrant](https://nam10.safelinks.protection.outlook.com/?url=https%3A%2F%2Furldefense.com%2Fv3%2F__https%3A%2F%2Fwww.vagrantup.com%2F__%3B!!CdHzsg!gisqTbsb_uMXUAnWzYhWGNgZ3J2N9xmR3dGBAh5fpGU0h7tTouLZMd3wX1-m9tFnNlV4TGrloHUDREwl3RhsTtSP0HuBsdafBbCV%24&data=05%7C01%7Cbrigthain.kargong%40gainwelltechnologies.com%7Cdb7338c1c16f46a3861e08da52f5873e%7Cc663f89cef9b418fbd3d41e46c0ce068%7C0%7C0%7C637913511735028723%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=lQw07ZG9MbWt0u%2BF%2FV%2B9FfLg9lua11hSbIgBg2JRNmg%3D&reserved=0) box, if it can. This lets you use Packer to automatically create arbitrarily complex Vagrant boxes, and is in fact how the official boxes distributed by Vagrant are created.
* Vagrant is **an open-source tool that allows you to create, configure, and manage boxes of virtual machines through an easy to use command interface**. Essentially, it is a layer of software installed between a virtualization tool (such as VirtualBox, Docker, Hyper-V) and a VM



***USER-VARIABLES (See project 7)***

* They help your template to be further configured with variables from the command-line, environment variables, vault or files.
* Most often the variables are placed in the template however you can pass in an empty string in the template and pass in the value when running the packer build command. So its going to be packer build -var "description=Webserver" first.json.This passes on the value at run time.
* You can pass in multiple variables at run time and its going to look like packer build -var "description=mywebser" -var "name=girl" First.json
* For templates with multiple variables referencing them on the command line or inputting them in the main template can be cumbersome so its best to create a separate file just for the variables and reference that file
* The variable file will look like this

{

    "description": "myWebserver"

}

* They way you reference it at run time is by running packer build -var-file=variable.json first.json
* In this case variable.json is the name of the variable file

***ENVIRONMENT VARIABLES (See project8)***

* These are great cause they are not really stored anywhere. We can store our access keys and secret keys here and pass them into our packer config
* For windowsyou run  setx plus value (access or secret key)on the command line and this saves the values
* This will be saved in the environment variables
* To do so on a linux machine you run export plus value ex export ACCESS\_KEY=DBSKJDNWKEJMWDKL
* To view it you run printenv | grep -I access\_key
* This is helpful as our credential are not passed into the template giving people access to them
* Passing environment variables on the command line will look something like this

 "access\_key": "{{env `ACCESS\_KEY`}}",

        "secret\_key": "{{env `SECRET\_KEY`}}"

    },

* So instead of "user"as in user variable we input "env"
* The sensitive-variablesparameter in the template is to prevent the access key and secret key from being provided in the packer logs hence giving people access to them

AWS AUTHENTICATION

* The following methods of authentication are supported by packer
* Static credentials: These credentials are those that are being hard coded into the template
* Environment variables: As mentioned above. Here you first have to set values in your local either by running the setx for windows or export for linux and passing the values. Here you are not setting up the access and secret values as environment variables and then calling them as user variables in the builders section. Here there's no value provided in the template and packer automatically will find them in your local computer once exported.
* Shared credentials: Here you can use an AWS credentials file to specify your credentials. The default location is HOME/.aws/credentials on Linux and OS X, or %USERPROFILE%.aws\credentials for windows users. Packer will check in this location if it fails to detect the credentials inline or in the environment
* EC2 Role: You can assign a specific role if you are trying to run it on an instance

 WORKING PACKER TEMPLATE:

{

"builders": [

{

"boot\_command": [

"<tab> text ks=http://{{.HTTPIP}}:{{.HTTPPort}}/ks-{{user `profile`}}.cfg<enter><wait>"

],

"boot\_wait": "10s",

"disk\_size": "{{user `disk\_size`}}",

"format": "ova",

"guest\_additions\_mode": "disable",

"guest\_os\_type": "RedHat\_64",

"headless": true,

"http\_directory": "http",

"iso\_checksum": "07b94e6b1a0b0260b94c83d6bb76b26bf7a310dc78d7a9c7432809fb9bc6194a",

"iso\_url": "<https://mirrors.edge.kernel.org/centos/7.9.2009/isos/x86_64/CentOS-7-x86_64-Minimal-2009.iso>",

"output\_directory": "output-{{user `profile`}}",

"shutdown\_command": "sudo -S shutdown -P now",

"ssh\_password": "{{user `ssh\_password`}}",

"ssh\_port": 22,

"ssh\_timeout": "10000s",

"ssh\_username": "{{user `ssh\_username`}}",

"type": "virtualbox-iso",

"vboxmanage": [

[

"modifyvm",

"{{.Name}}",

"--memory",

"{{user `memory`}}"

],

[

"modifyvm",

"{{.Name}}",

"--cpus",

"{{user `cpus`}}"

]

],

"vm\_name": "centos7-ec2-{{user `profile`}}"

}

],

"description": "Bare CentOS 7 prepped for AMI import",

"variables": {

"cpus": "4",

"disk\_size": "8192",

"memory": "6122",

"profile": "xfs",

"ssh\_password": "centos",

"ssh\_username": "centos",

"aws\_access\_key": "AKIAXB5IN5PEKWT5M5OU",

"aws\_secret\_key": "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"

}

}