Image Building Pipeline with Hashicorp Packer

Alvin Chua

Overview

High level infrastructure diagram

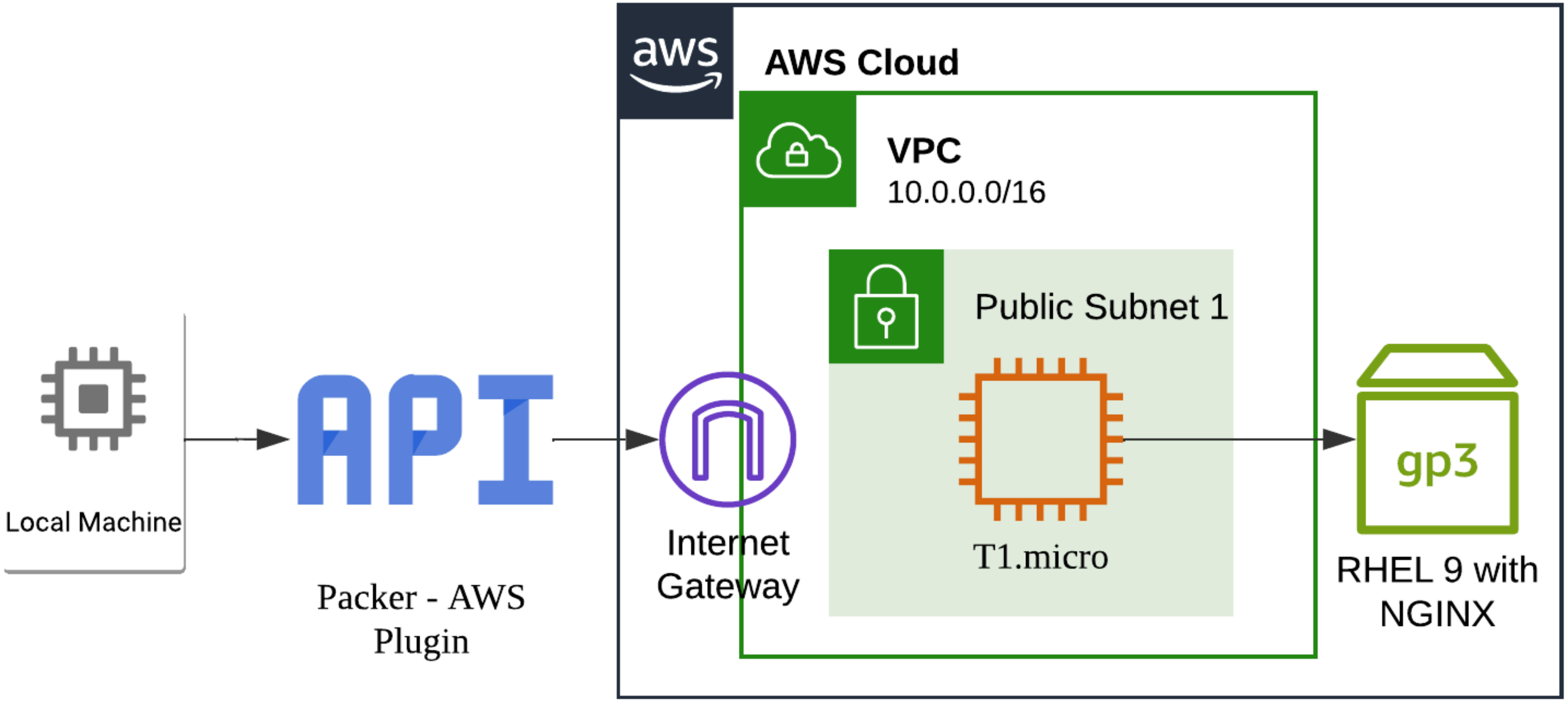


Figure 1 - High level infrastructure diagram of the image pipeline

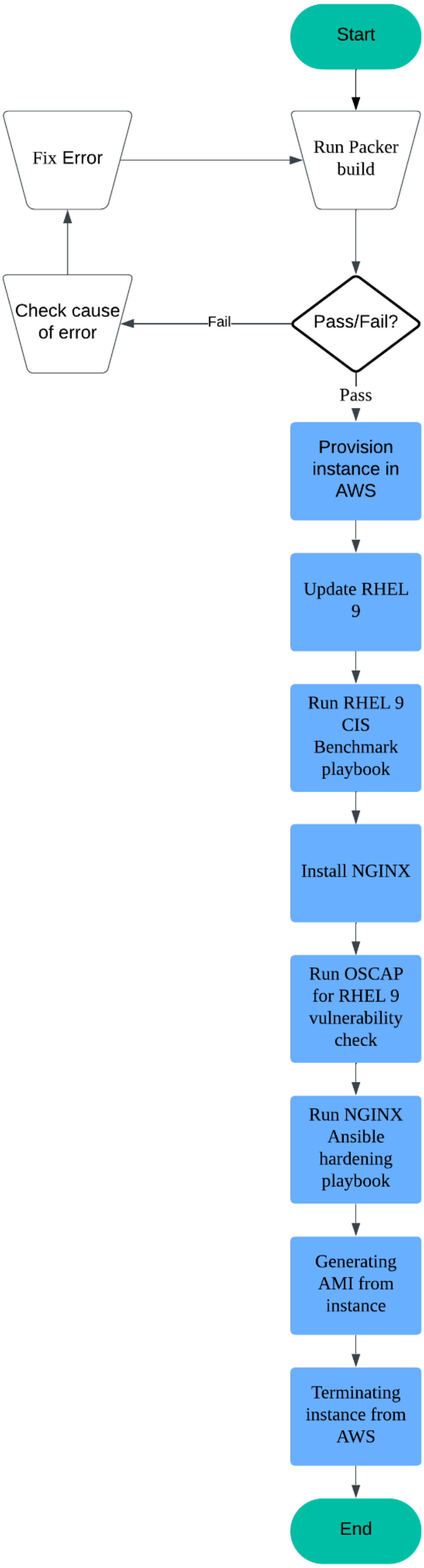

Start
Fix Error
Run Packer build
Check cause of error
Fail
Pass/Fail?
Pass
Provision instance in AWS
Update RHEL 9
Run RHEL 9 CIS Benchmark playbook
Install NGINX
Run OSCAP for RHEL 9 vulnerability check
Run NGINX Ansible hardening playbook
Generating AMI from instance
Terminating instance from AWS
End

Figure 2 - Flowchart for the image pipeline

Implementation

Packer was used to generate an immutable image of NGINX installed on Red Hat Enterprise Linux 9 (RHEL 9). The pipeline would update the RHEL 9 image from the AWS Marketplace before hardening it in accordance to CIS Level 2 RHEL 9 benchmark. After the hardening of the operating system, the installation and configuration of NGINX would follow.

After the configuration of NGINX, the compliance check of the operating system with OpenSCAP (OSCAP) would begin. Following the compliance check, the hardening of NGINX configurations would begin. Once the hardening of NGINX is done, Packer would generate an AMI before terminating the EC2 instance.

After successfully generating an AMI through the pipeline, a manual process would be required to launch an EC2 instance from the AMI. This instance was then verified to ensure that NGINX was still functioning as expected.

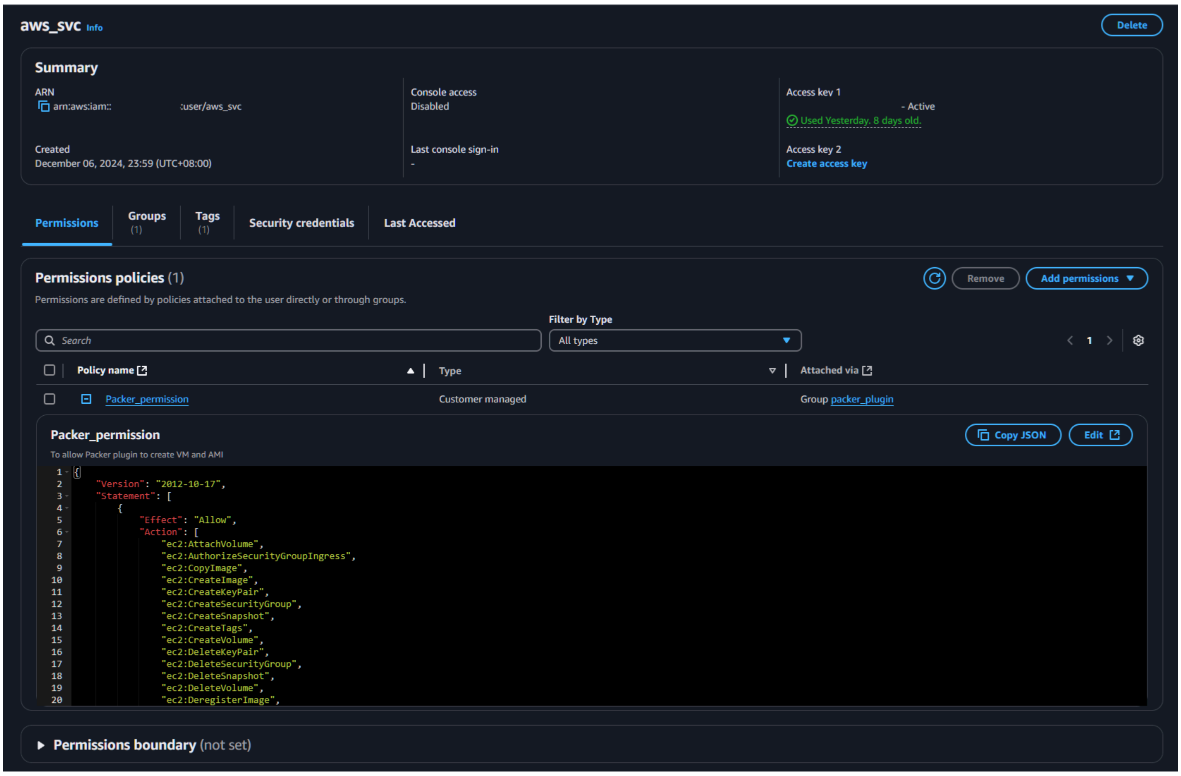
Security Implementations

Identity and Access Management (IAM)

Creating service account for Packer

A dedicated service account "aws_svc" was created to grant Packer permissions for provisioning, modifying, and deleting EC2 instances and AMIs.

The access and secret keys were generated after the service account had been created successfully. The access and secret keys were configured to be stored as environment variables on my local machine using AWS CLI version 2, to allow Packer access to the service account that was created.



aws_svc Info

Summary

ARN
arn:aws:iam::[redacted]:user/aws_svc

Created
December 06, 2024, 23:59 (UTC+08:00)

Console access
Disabled

Last console sign-in
-

Access key 1
[redacted] - Active
Used Yesterday. 8 days old.

Access key 2
Create access key

Permissions policies (1)

Permissions are defined by policies attached to the user directly or through groups.

Filter by Type: All types

Policy name	Type	Attached via
Packer_permission	Customer managed	Group packer_plugin

Packer_permission

To allow Packer plugin to create VM and AMI

```
{
    "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",
```

Permissions boundary (not set)

Figure 3 - Screenshot of the service account and permissions used for Packer to access AWS

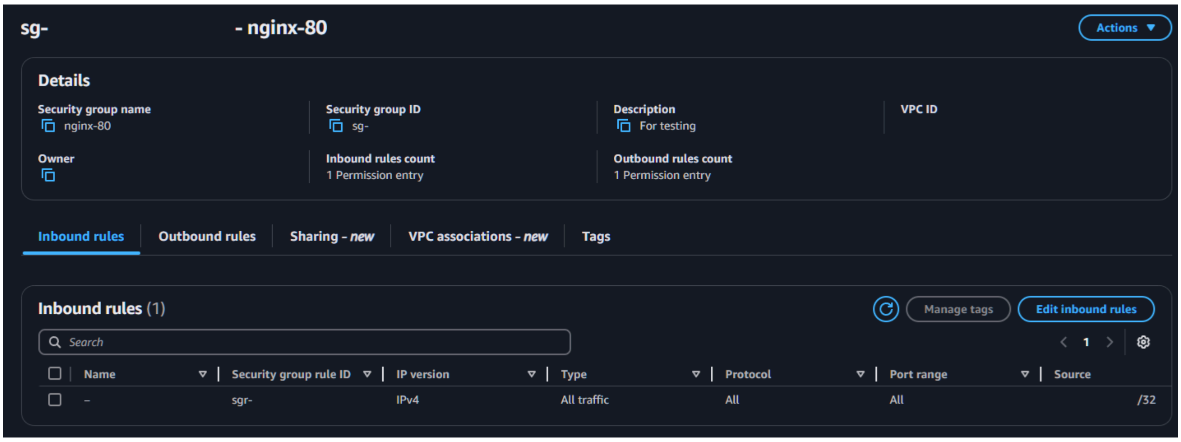
Creating Key pair for accessing EC2 instance using SSH

A key pair was generated to allow my local machine to connect to an EC2 instance with the AMI that has been built successfully using Packer. After creating the key pair, the private key can be downloaded for use on my local machine. The file path of the private key can be used when establishing an SSH connection using the following command: “*`ssh -i /path/to/private-key.pem username@hostname`*”

Network Security

Creating Network Security Group

A security group was created with an inbound rule to allow only my local machine’s IP address range to enter the network that the EC2 instance was provisioned in.



sg- - nginx-80

Details

Security group name	Security group ID	Description	VPC ID
nginx-80	sg-	For testing	
Owner	**Inbound rules count**	**Outbound rules count**	
	1 Permission entry	1 Permission entry	

Inbound rules (1)

□	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source
□	–	sgr-	IPv4	All traffic	All	All	/32

Figure 4 - Screenshot of the inbound rule for the network security group created

Security Best Practices

CIS Level 2 RHEL 9 Hardening

The hardening of RHEL 9 was executed using an Ansible playbook. This ensured that the operating system’s configurations adhered to the latest CIS level 2 benchmark guidelines.

Open SCAP (OSCAP) Compliance Check

An OSCAP compliance check was conducted following the hardening of RHEL 9 to validate that the system meets the required security standards and was aligned with the implemented hardening measures.

NGINX Hardening

The hardening of NGINX was executed using an Ansible playbook as well. This would ensure that the NGINX configurations were aligned with the latest security standards.