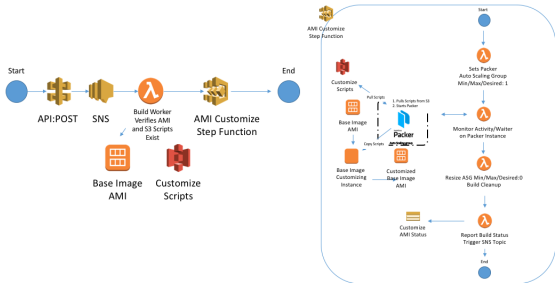
Custom AMI

Introduction

Custom AMI is feature of our Automated Safeguards for AWS. This feature enables our customers to use an automated build process to be able to create custom AMIs by providing configuration management bundles to be layered on top of standard ClearDATA base AMIs.

Architecture



As shown above, the following components are deployed in customer account:

- An S3 bucket to store the default customization scripts
- An API Gateway that can be used to trigger AMI customizations - This API Gateway will start a Step Function, allowing for a dashboard that can be used to troubleshoot any issues.
- Other elements that are required by the process above include an SNS topic, orchestration lambda, Step function, & Packer AMI.

S3 bucket structure

The S3 bucket created in the customer account can be used by the customer to store customization scripts that will be automatically called upon whenever a new Base AMI is dropped into the account. The following structure:

- This default bucket naming and structure is as follows:
    - bucket name: {aws_account_id}-os-customizations
        - Example: 123456789101-os-customizations
    - Object structure: {region}/{base-image-os}/{script-name}
        - Example: us-east-1/amazon-linux/2-2017.12.0/customize.sh
            - This would indicate a bundle to be executed using a the ClearDATA hardened base AMI that is based on the amazon-linux version 2-2017.12.0 operating system
            - Please note that the Custom AMI feature supports shell scripts for linux and derivates, and PowerShell scripts for Windows
            - The script name must be customize.sh for a Linux based OS, or customize.ps1 for a Windows OS

Customers can also create their own S3 bucket to have more customization options and allow for more than one customized AMI to be created per OS.

SNS Topic

Customers can get notifications when a new Base AMI is drooped into their account by subscribing to the SNS topic: ClearDATA-Custom-Image-Topic

API Gateway

The exact URL for the API gateway will be defined when it is created - Example: https://abcdefghij.execute-api.us-east-1.amazonaws.com

Image customization is triggered by issuing a POST command to the {api-url}/prod/build API endpoint - The payload of the POST containing the following information in JSON format:

- "ami_name" - name of new custom ami
- "custom_script_name" - name of script to invoke at s3_url
- "instance_type" - instance size used for customization process, e.g., m4-large
- "os_style" - operating system type, `windows` or `linux` only
- "s3_url" - url to the s3 object containing the customization script
- "source_ami_id" - source ami id
- "ssh_username" - optional (default mappings outlined below)
- "subnet_id" - optional
- "vpc_id" - optional

SSH User Mappings

Custom AMI Builder follows the defaults from Amazon Web Services for default SSH user mappings. Below is a listing of these defaults from the AWS documentation:

Amazon Linux 2 or the Amazon Linux AMI, the user name is ec2-user.

Centos AMI, the user name is centos.
Debian AMI, the user name is admin or root.
RHEL AMI, the user name is ec2-user or root.
Ubuntu AMI, the user name is ubuntu.
Otherwise, if ec2-user and root don't work, check with the AMI provider.

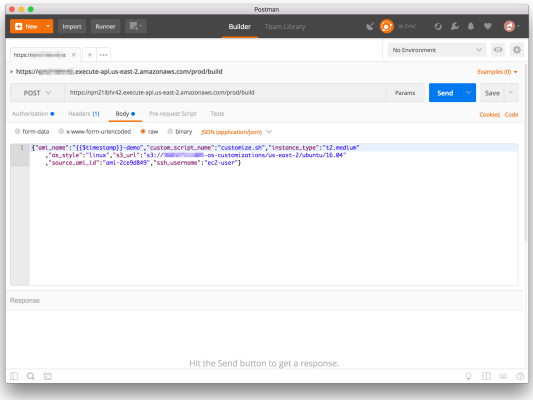
Sample Payload

This payload is to build a Windows 2016 Amazon Machine Image.

```
{
"ami_name": "windows_2016_custom_ami",
"custom_script_name":"customize.ps1",
"instance_type":"c4.large",
"os_style":"windows",
"s3_url":"s3://XXXXXXXXXXXX-os-customizations/us-east-1/windows/2016",
"source_ami_id":"ami-0cee167fbf4cc266f"
}
```

Please note IAM authentication is required to use the gateway.

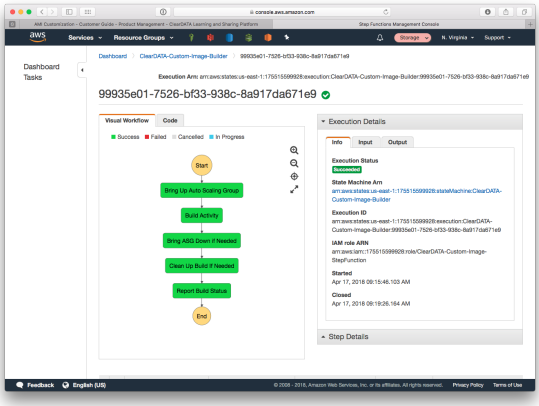
The example below shows a sample call when using Postman per https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-use-postman-to-call-api.html

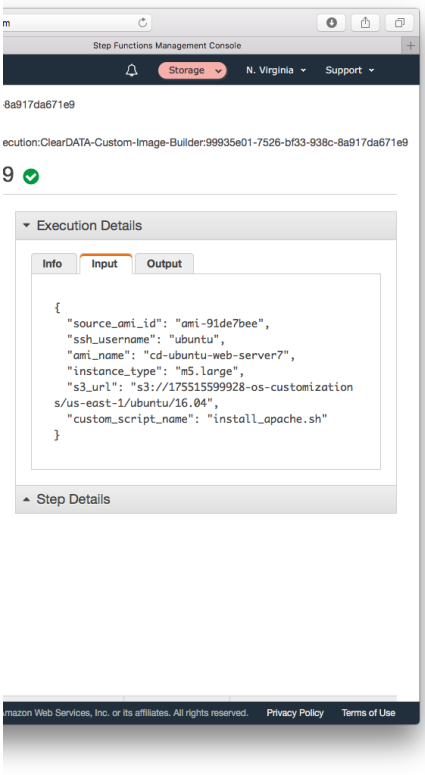


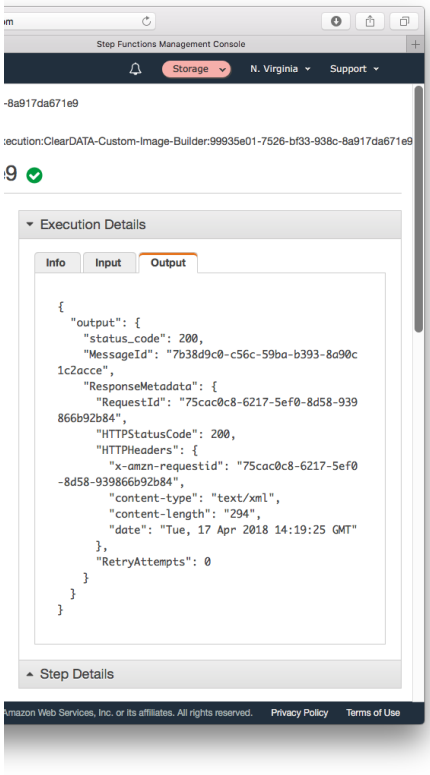
An other option that allows integrating is to call the API from a lambda per https://docs.aws.amazon.com/apigateway/latest/developerguide/call-api-with-api-gateway-lambda-authorization.html

Troubleshooting Tips

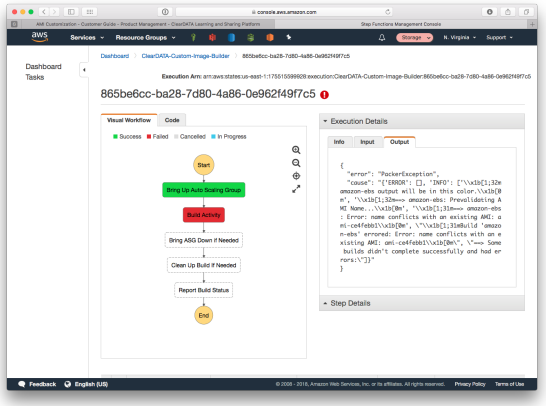
The call to the API gateway triggers the step function that was deployed in the account (ClearDATA-Custom-Image-Builder) - The Step function management console can then be used for both status updates and troubleshooting. As an example, this view of the console shows a successful execution:







While the example below shows a failed execution as the name of the target AMI conflicts with one that was already built



Stopping the State Function

If you stop the ClearDATA-Custom-Image-Builder state machine it may leave certain components running, such as instances in the autoscaling group. If you stop the Step Function during an execution, you will need to modify the ASG named ClearDATA-Custom-Image-ASG. The Step Function will adjust the ASG with a Desired/Min/Max count of 1, and if the function is stopped prior to spinning down the ASG the instance may remain online and future executions will not work. If the Step Function is stopped, you should modify the ASG to set Desired/Min/Max with a count of 0.

Supported Operating Systems

amazon-linux/2-2017.12.0
amazon/linux/2017.03.0
amazon/linux/2017.09.1
centos6
centos7
debian8.8/jessie
rhel6
rhel7
ubuntu/18.04
ubuntu/16.04
windows/2012Wsql2012
windows/2012Wsql2014
windows/2012Wsql2016-ent
windows/2012Wsql2016-std
windows/2012r2
windows/2016
windows/2016Wsql2016-ent
windows/2016Wsql2016-std

Example Customize Scripts

Powershell

```
<#
                        #### Custom AMI Builder Windows PowerShell Example
####
In order for the Custom AMI Script for Windows to function properly inside
```

```
the builder you must ensure
that your exit codes produce the desired result.

Should you choose not to build exit codes to your script.  The Builder
will continue on and produce an AMI
from your custom script.  This would result in an AMI that may or may not
have the desired customizations
you seek for your build.

#>

$username = "temp_user"
$password =  ConvertTo-SecureString  -AsPlainText "password@123!" -Force

<#
Function below is used to determine which version of PowerShell your
customize.ps1 script is running on.
This is important as you can only use PowerShell Modules that are present
in the PS release version.
#>
Function ps_version {
        $version = $PSVersionTable.PSVersion.Major
        return $version
        }

Function createuser  {
    $version = ps_Version
    if ($version -le 4) {
        write-host("Using PS $version modules")
        $BSTR = [System.Runtime.InteropServices.Marshal]::
SecureStringToBSTR($password)
        $UnsecurePassword = [System.Runtime.InteropServices.Marshal]::
PtrToStringAuto($BSTR)
        try {
            write-host "Creating local user $username"
            net user $username $UnsecurePassword /add
            }
        catch {
            Write-Host "Error!!! $?"
            exit(1)
            }
        try{
            write-host "Adding newly created user $username to
Administrators Group"
            net localgroup administrators $username /add
            }
        catch {
            write-host("Unable to create user!!!")
            exit(1)
            }
        write-host "User $username was created and placed in the
Administrators Group Successfully..."
        exit(0)
        }
    elseif ($version -ge 5) {
        write-host("Using PS $version modules")
        try {
            write-host "Creating local user $username"
            New-LocalUser "$username" -Password $password -FullName
"$username" -Description "One time use password"
            }
        catch {
            write-host "ERROR!!! $?"
```

```
            exit(1)
            }
        try
            {
            Write-Host "Adding newly created user $username to
Administrators Group"
            Add-LocalGroupMember -Group "Administrators" -Member
"$username"
            }
        catch {
            write-host("Unable to create user!!!")
            exit(1)
              }
        write-host "User $username was created and placed in the
Administrators Group Successfully..."
        exit(0)
    }
}

createuser
```

Bash

```
customize.sh
#!/usr/bin/env bash
echo Installing Apache Web Server...
apache_script="tmp/install_apache.sh"
chmod +x ${apache_script}
sudo ./${apache_script}

install_apache.sh
#!/usr/bin/env bash
echo "Y" | yum install httpd
echo 'Listen 8080' >> /etc/httpd/conf/httpd.conf
semanage port -a -t http_port_t -p tcp 8080
systemctl enable httpd
```