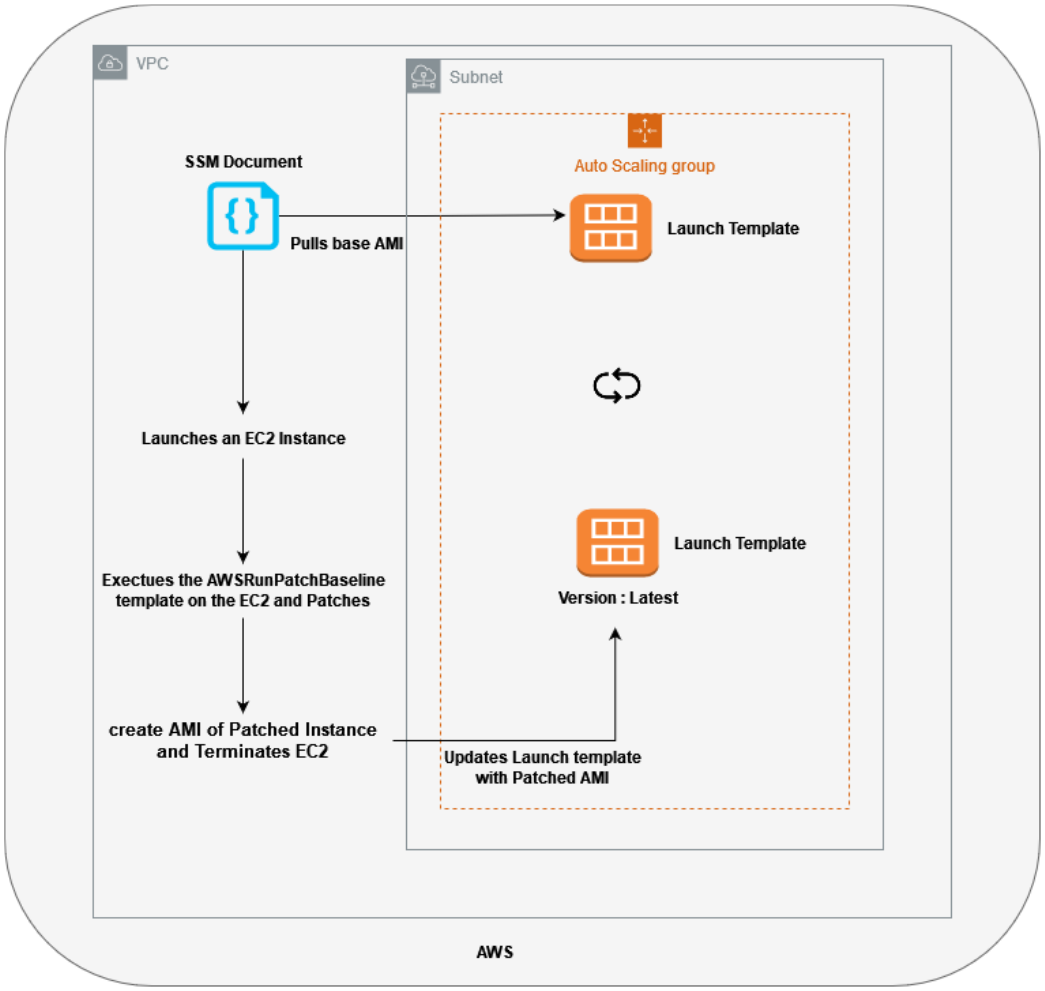
How To Automate Patching for Auto Scaling Groups

Wiki to walk you through on how to deploy a patching solution for AWS Auto Scaling Groups.

-Works with ASGs created in Terraform,

-patching is done by SSM runbook (automation document).

-To make sure that terraform picks the changes made to the ASG by the runbook, we use a terraform data source to pass the AMI ID used in the launch templates. The new AMI will have a tag called "*Version*" with a value of "*Latest*". The data source will use this value to pull the AMI ID for the template.



Steps:

1. Create Role for SSM

- Role should have the following trust policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "ssm.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- Role should have the follwoing permission sets:

```
{
    "Statement": [
        {
            "Action": [
                "iam:*",
                "iam:PassRole",
                "ec2:*",
                "ssm:*",
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:StartInstanceRefresh",
                "autoscaling:UpdateAutoScalingGroup"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Sid": "VisualEditor0"
        }
    ],
    "Version": "2012-10-17"
}
```

2. Create SSM PatchAMIAndUpdateASG runbook using the `"aws_ssm_document" terraform resource.` Create a file with following code to use with the SSM resource:

```
description: Systems Manager Automation Demo - Patch AMI and Update ASG
schemaVersion: '0.3'
assumeRole: '{{ AutomationAssumeRole }}'
outputs:
  - createImage.ImageId
parameters:
  AutomationAssumeRole:
    type: String
    description: '(Required) The ARN of the role that allows Automation to perform the actions on your behalf.
    default: ''
  SourceAMI:
    type: String
    description: (Required) The ID of the AMI you want to patch.
  SubnetId:
    type: String
    description: (Required) The ID of the subnet where the instance from the SourceAMI parameter is launched.
  SecurityGroupIds:
    type: StringList
    description: (Required) The IDs of the security groups to associate with the instance launched from the Sou
  NewAMI:
    type: String
    description: (Optional) The name of of newly patched AMI.
```

```
    default: 'patchedAMI-{{global:DATE_TIME}}'
  TargetASG:
    type: String
    description: (Required) The name of the Auto Scaling group you want to update.
  InstanceProfile:
    type: String
    description: (Required) The name of the IAM instance profile you want the source instance to use.
  SnapshotId:
    type: String
    description: (Optional) The snapshot ID to use to retrieve a patch baseline snapshot.
    default: ''
  RebootOption:
    type: String
    description: '(Optional) Reboot behavior after a patch Install operation. If you choose NoReboot and patche
    allowedValues:
      - NoReboot
      - RebootIfNeeded
    default: RebootIfNeeded
  Operation:
    type: String
    description: (Optional) The update or configuration to perform on the instance. The system checks if patche
    allowedValues:
      - Install
      - Scan
    default: Install
mainSteps:
  - name: startInstances
    action: 'aws:runInstances'
    timeoutSeconds: 1200
    maxAttempts: 1
    onFailure: Abort
    inputs:
      ImageId: '{{ SourceAMI }}'
      InstanceType: m5.large
      MinInstanceCount: 1
      MaxInstanceCount: 1
      IamInstanceProfileName: '{{ InstanceProfile }}'
      SubnetId: '{{ SubnetId }}'
      SecurityGroupIds: '{{ SecurityGroupIds }}'
  - name: verifyInstanceManaged
    action: 'aws:waitForAwsResourceProperty'
    timeoutSeconds: 600
    inputs:
      Service: ssm
      Api: DescribeInstanceInformation
      InstanceInformationFilterList:
        - key: InstanceIds
          valueSet:
            - '{{ startInstances.InstanceIds }}'
      PropertySelector: '$.InstanceInformationList[0].PingStatus'
      DesiredValues:
        - Online
    onFailure: 'step:terminateInstance'
  - name: installPatches
    action: 'aws:runCommand'
    timeoutSeconds: 7200
    onFailure: Abort
    inputs:
```

```
      DocumentName: AWS-RunPatchBaseline
      Parameters:
        SnapshotId: '{{SnapshotId}}'
        RebootOption: '{{RebootOption}}'
        Operation: '{{Operation}}'
      InstanceIds:
        - '{{ startInstances.InstanceIds }}'
  - name: stopInstance
    action: 'aws:changeInstanceState'
    maxAttempts: 1
    onFailure: Continue
    inputs:
      InstanceIds:
        - '{{ startInstances.InstanceIds }}'
      DesiredState: stopped
  - name: createImage
    action: 'aws:createImage'
    maxAttempts: 1
    onFailure: Continue
    inputs:
      InstanceId: '{{ startInstances.InstanceIds }}'
      ImageName: '{{ NewAMI }}'
      NoReboot: false
      ImageDescription: Patched AMI created by Automation
  - name: terminateInstance
    action: 'aws:changeInstanceState'
    maxAttempts: 1
    onFailure: Continue
    inputs:
      InstanceIds:
        - '{{ startInstances.InstanceIds }}'
      DesiredState: terminated
  - name: updateASG
    action: 'aws:executeScript'
    timeoutSeconds: 300
    maxAttempts: 1
    onFailure: Abort
    inputs:
      Runtime: python3.8
      Handler: update_asg
      InputPayload:
        TargetASG: '{{TargetASG}}'
        NewAMI: '{{createImage.ImageId}}'
      Script: |-
        from __future__ import print_function
        import datetime
        import json
        import time
        import boto3

        # create auto scaling and ec2 client
        asg = boto3.client('autoscaling')
        ec2 = boto3.client('ec2')

        def update_asg(event, context):
            print("Received event: " + json.dumps(event, indent=2))

            target_asg = event['TargetASG']
```

```
            new_ami = event['NewAMI']

            # get object for the ASG we're going to update, filter by name of target ASG
            asg_query = asg.describe_auto_scaling_groups(AutoScalingGroupNames=[target_asg])
            if 'AutoScalingGroups' not in asg_query or not asg_query['AutoScalingGroups']:
                return 'No ASG found matching the value you specified.'

            # gets details of an instance from the ASG that we'll use to model the new launch template after
            source_instance_id = asg_query.get('AutoScalingGroups')[0]['Instances'][0]['InstanceId']
            instance_properties = ec2.describe_instances(
                InstanceIds=[source_instance_id]
            )
            source_instance = instance_properties['Reservations'][0]['Instances'][0]

            # create list of security group IDs
            security_groups = []
            for group in source_instance['SecurityGroups']:
                security_groups.append(group['GroupId'])

            # create a list of dictionary objects for block device mappings
            mappings = []
            for block in source_instance['BlockDeviceMappings']:
                volume_query = ec2.describe_volumes(
                    VolumeIds=[block['Ebs']['VolumeId']]
                )
                volume_details = volume_query['Volumes']
                device_name = block['DeviceName']
                volume_size = 10
                volume_type = 'gp2'
                device = {'DeviceName': device_name, 'Ebs': {'VolumeSize': volume_size, 'DeleteOnTermination':
                mappings.append(device)

            # create new launch template using details returned from instance in the ASG and specify the newly
            time_stamp = time.time()
            time_stamp_string = datetime.datetime.fromtimestamp(time_stamp).strftime('%m-%d-%Y_%H-%M-%S')
            new_template_name = f'{new_ami}_{time_stamp_string}'
            version_description = f'My Launch Template'
            launch_template_name = asg_query['AutoScalingGroups'][0]['LaunchTemplate']['LaunchTemplateName']
            try:
                response = ec2.create_launch_template_version(
                    LaunchTemplateName=launch_template_name,
                    SourceVersion='$Latest',
                    VersionDescription=version_description,
                    LaunchTemplateData={
                        'ImageId': new_ami,
                        'InstanceType': source_instance['InstanceType'],
                        'IamInstanceProfile': {
                            'Arn': source_instance['IamInstanceProfile']['Arn']
                        },
                        'KeyName': source_instance['KeyName'],
                        'SecurityGroupIds': security_groups
                    }
                )
                new_version = response['LaunchTemplateVersion']['VersionNumber']

            except Exception as e:
                return f'Exception caught: {str(e)}'

```

```
            else:
                ## Update the tag on the old AMI
                tag_name = 'Version'
                tag_value = '0.1'

                response = ec2.describe_images(
                    Filters=[
                        {
                            'Name': 'tag:Version',
                            'Values': ['Latest']
                        }
                    ]
                )

                if len(response['Images']) == 0:
                    print(f"No AMI found with Version = 'Latest'")
                else:
                    ami_id = response['Images'][0]['ImageId']  # assuming there's only one match

                    current_tags = response['Images'][0]['Tags']

                    updated_tags = [{'Key': tag_name, 'Value': tag_value}]
                    for tag in current_tags:
                        if tag['Key'] != tag_name:
                            updated_tags.append(tag)

                    ec2.create_tags(Resources=[ami_id], Tags=updated_tags)

                ## Update Launch Template
                response = ec2.describe_launch_template_versions(LaunchTemplateName=launch_template_name, Versi
                ami_id = response['LaunchTemplateVersions'][0]['LaunchTemplateData']['ImageId']

                print(f"The Auto Scaling Group: {target_asg} is using the launch template: {launch_template_nam

                ## Make latest version the default version on launch template
                response = ec2.modify_launch_template(
                    LaunchTemplateName=launch_template_name,
                    DefaultVersion=str(new_version)
                )

                tags = [
                  {'Key': "Owner", 'Value':"Sandbox"},
                  {'Key': "Version", 'Value':"Latest"},
                  {'Key': "Env", 'Value':"Dev"}

                ]

                response = ec2.create_tags(
                    Resources=[ami_id],
                    Tags=tags
                )

                ## update ASG to use new launch template
                asg.update_auto_scaling_group(
                    AutoScalingGroupName=target_asg,
                    LaunchTemplate={
                        'LaunchTemplateName': launch_template_name,
                        'Version': str(new_version)
```

```
                    }
                )

                # Start instance refresh on Autoscaling group
                response = asg.start_instance_refresh(
                    AutoScalingGroupName=target_asg,
                    Strategy='Rolling',
                    Preferences={
                      'InstanceWarmup': 300,
                      'MinHealthyPercentage': 50
                    }
                )


                return f'Updated ASG {target_asg} to use launch template version: {new_version} which uses AMI
```

This document will create an instance using the same AMI used by the ASG, proceed to update/apply patching on the new instance, it will then stop it, create a new AMI, update the launch template with the new AMI thus creating a new launch template version. Lastly, it will update the ASG to use the new version of the launch template.

3. Give or make sure that the IAM role used by the EC2 instance of the Auto Scaling Group have SSM permissions.

4. **TESTING:**

-In the navigation pane, choose **Automation**, and then choose **Execute automation**.

-In the **Choose document** page, choose the **Owned by me** tab.

-Search for the runbook Document you created, and select the button in the card.

-Choose **Next**.

-Choose **Simple execution**.

-Specify values for the input parameters. Be sure the `SubnetId` and `SecurityGroupIds` you specify allow access to the public Systems Manager endpoints, or your interface endpoints for Systems Manager.

-Choose **Execute**.

-After automation completes, in the Amazon EC2 console, choose Auto Scaling, and then choose Launch Templates. Select the launch template used by your ASG and verify that you see the new launch template version, and that it uses the new AMI.

1. gdgb
2. dgb

Resources:

Updating AMIs for Auto Scaling groups - AWS Systems Manager