Security: Automatically remove unused old security groups using AWS Lambda

Maintaining security groups across multiple VPCs, regions, accounts is quite an time consuming effort. I decided to automate this, with an lambda.

1) Launch an Ec2 instance in the Region and provide the Access keys and secret keys

2) Check your file at ~/.aws/config and set the AWS Region properly
3) Create a bash shell in the Ec2 linux instance and change the Vpc

```
   #!/bin/bash
set -ex

vpc_to_spam="vpc-b33cafc9a" ---> Replace the Vpc based on the region
for i in {1..3}
    do
       val="${RANDOM}"
       aws ec2 create-security-group \
           --vpc-id  "${vpc_to_spam}" \
           --group-name "sg_group_name_${val}" \
           --description "My security group_${val}"
    done
```

This will create a dummy Security Groups for testing
3) Go to the Cloudformation and create a stack as i created below in the json

```
   {
  "Resources": {
    "SecurityGroupJanitorServiceRoleCF49BC8D": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                "Service": "lambda.amazonaws.com"
              }
            }
          ],
          "Version": "2012-10-17"
        },
        "ManagedPolicyArns": [
          {
            "Fn::Join": [
              "",
              [
                "arn:",
                {
                  "Ref": "AWS::Partition"
                },
                ":iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
              ]
            ]
          }
        ]
      },
      "Metadata": {
        "aws:cdk:path":
```

```
"remove-unused-security-groups/SecurityGroupJanitor/ServiceRole/Resource"
      }
    },
    "SecurityGroupJanitorServiceRoleDefaultPolicyF289DD19": {
      "Type": "AWS::IAM::Policy",
      "Properties": {
        "PolicyDocument": {
          "Statement": [
            {
              "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroupReferences",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeStaleSecurityGroups",
                "ec2:DeleteSecurityGroup"
              ],
              "Effect": "Allow",
              "Resource": "*",
              "Sid":
"AllowLambdaToDescribeSecurityGroupsInstancesAndDeleteSecurityGroups"
            }
          ],
          "Version": "2012-10-17"
        },
        "PolicyName": "SecurityGroupJanitorServiceRoleDefaultPolicyF289DD19",
        "Roles": [
          {
            "Ref": "SecurityGroupJanitorServiceRoleCF49BC8D"
          }
        ]
      },
      "Metadata": {
        "aws:cdk:path":
"remove-unused-security-groups/SecurityGroupJanitor/ServiceRole/DefaultPolicy/
Resource"
      }
    },
    "SecurityGroupJanitor9CF4173D": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "Code": {
          "ZipFile": "# -*- coding: utf-8 -*-\n\"\"\"\n.. module: CleanUp unused
security groups in a region using lambda\n    :platform: AWS\n    :copyright: (c)
2019 PaulZeedup.,\n    :license: Apache, see LICENSE for more details.\n..
moduleauthor:: Mystique\n.. contactauthor:: paulzeedup@github issues\n\"\"\"\n\
nimport boto3\nfrom botocore.exceptions import ClientError\nimport logging\n\n#
Initialize Logger\nlogger = logging.getLogger()\nlogger.setLevel(logging.INFO)\n\
ndef set_global_vars():\n    global_vars = {'status': False}\n    try:\n
global_vars['Owner']                    = \"Paulzeedup\"\n
global_vars['Environment']              = \"Prod\"\n
global_vars['region_name']              = \"us-east-1\"\n
global_vars['tag_name']                 = \"serverless_security_group_janitor\"\n
global_vars['exclude_sgs']               = set([\"sg-04f19c7e87bdae9ef\", \"sg-
f8ed38a1\", \"sg-0266f091db5460f56\"])\n        global_vars['status']
= True\n    except Exception as e:\n        logger.error(\"Unable to set Global
Environment variables. Exiting\")\n        global_vars['error_message']
= str(e)\n    return global_vars\n\ndef get_unused_sgs(excluded_sgs: set) -> dict:\
n    resp_data = {'status': False, 'unused_sg_ids':set, 'error_message': ''}\n
try:\n        ec2 = boto3.resource('ec2')\n        #TODO: Use pagination\n
all_sgs = list(ec2.security_groups.all())\n        insts =
list(ec2.instances.all())\n        all_sg_ids = set([sg.id for sg in all_sgs])\n
# Make a list of SGs associated with instances\n        inst_sg_ids=[]\n        for
inst in insts:\n            for sg_id in inst.security_groups:\n
inst_sg_ids.append( sg_id.get('GroupId') )\n    except Exception as e:\n
resp_data['error_message'] = str(e)\n    # Lets unique the list\n    inst_sg_ids =
set(inst_sg_ids)\n    # Unused Security Groups\n    unused_sg_ids = all_sg_ids -
inst_sg_ids\n    # Let removed the excluded ones\n    resp_data['unused_sg_ids'] =
unused_sg_ids - excluded_sgs\n    resp_data['status'] = True\n    return resp_data\
n\ndef janitor_for_security_groups(unused_sgs: dict) -> dict:\n    \"\"\"\
n    :param unused_sg_ids: Set of unqiue security group ids to be deleted\
n    :param type: set\n\n    :return: resp_data Return a dictionary of data\
n    :rtype: json\n    \"\"\"\n    sg_deleted = {'status': False,
'TotalSecurityGroupsDeleted':'','SecurityGroupIds': [] }\n    ec2 =
boto3.client('ec2')\n    for sg_id in unused_sgs.get('unused_sg_ids'):\n
logging.info(f\"Attempting to delete security group: {sg_id}\")\n        try:\n
resp=ec2.delete_security_group(GroupId=sg_id)\n            response_code =
resp['ResponseMetadata']['HTTPStatusCode']\n            if response_code >= 400:\n
# logging.error(f\"ERROR: {resp}\")\n                raise ClientError(resp)\n
#TODO: Can VPC ID too for more context if requred.\n
sg_deleted.get('SecurityGroupIds').append({'SecurityGroupId': sg_id})\n
except ClientError as e:\n            logging.error(f\"ERROR: {str(e.response)}\")\
n            sg_deleted['error_message'] = f\"Unable to delete Security Group with
id:{sg_id}. ERROR:{str(e)}\"\n    # Get the count of security groups deleted\n
if sg_deleted['SecurityGroupIds']:\n        sg_deleted['status'] = True\n
sg_deleted['TotalSecurityGroupsDeleted'] = len( sg_deleted['SecurityGroupIds'] )\n
else:\n        sg_deleted['TotalSecurityGroupsDeleted'] = 0\n    return sg_deleted\
n\ndef lambda_handler(event, context):\n\n    global_vars = set_global_vars()\n
resp_data = {\"status\": False, \"error_message\" : '' }\n\n    if not
global_vars.get('status'):\n        resp_data['error_message'] =
global_vars.get('error_message')\n        return resp_data\n\n    unused_sgs =
get_unused_sgs(global_vars.get('exclude_sgs'))\n    if not
unused_sgs.get('status'):\n        resp_data['error_message'] =
unused_sgs.get('error_message')\n        return resp_data\n\n    deleted_sgs =
janitor_for_security_groups(unused_sgs)\n    resp_data = deleted_sgs\n    return
resp_data\n\nif __name__ == '__main__':\n    lambda_handler(None, None)"
        },
        "Handler": "index.lambda_handler",
        "Role": {
          "Fn::GetAtt": [
            "SecurityGroupJanitorServiceRoleCF49BC8D",
            "Arn"
          ]
        },
        "Runtime": "python3.7",
        "FunctionName": "SecurityGroupJanitor",
        "Timeout": 10
      },
      "DependsOn": [
        "SecurityGroupJanitorServiceRoleDefaultPolicyF289DD19",
        "SecurityGroupJanitorServiceRoleCF49BC8D"
      ],
      "Metadata": {
        "aws:cdk:path":
"remove-unused-security-groups/SecurityGroupJanitor/Resource"
      }
    },

"SecurityGroupJanitorAllowEventRuleremoveunusedsecuritygroupsdeleteunusedsgs4D4E205
```

```
4D78F7CF2": {
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:InvokeFunction",
        "FunctionName": {
          "Fn::GetAtt": [
            "SecurityGroupJanitor9CF4173D",
            "Arn"
          ]
        },
        "Principal": "events.amazonaws.com",
        "SourceArn": {
          "Fn::GetAtt": [
            "deleteunusedsgs824006AC",
            "Arn"
          ]
        }
      },
      "Metadata": {
        "aws:cdk:path":
"remove-unused-security-groups/SecurityGroupJanitor/AllowEventRuleremoveunusedsecur
itygroupsdeleteunusedsgs4D4E2054"
      }
    },
    "deleteunusedsgs824006AC": {
      "Type": "AWS::Events::Rule",
      "Properties": {
        "ScheduleExpression": "rate(7 days)",
        "State": "ENABLED",
        "Targets": [
          {
            "Arn": {
              "Fn::GetAtt": [
                "SecurityGroupJanitor9CF4173D",
                "Arn"
              ]
            },
            "Id": "Target0"
          }
        ]
      },
      "Metadata": {
        "aws:cdk:path": "remove-unused-security-groups/delete_unused_sgs/Resource"
      }
    }
  }
}
```

This will create a stack and Lambda functions
In the lambda functions this will also create a cloudwatch events
Pass the variable in the lambda functions and test it