Analysis and Proposed Solution for Default VPC Deletion in AWS Accounts

July 24, 2025

1 Objective

The objective is to replace the default VPC deletion logic in an existing custom Python automation framework for provisioning AWS accounts with an AWS-native solution. The solution must automate default VPC deletion across all regions in new accounts, integrate with the existing framework, and address whether Service Control Policies (SCPs) can prevent default VPC creation. The rest of the framework remains unchanged.

2 Analysis

2.1 Current Approach

- **Custom Python Framework**: Automates AWS account provisioning and deletes default VPCs using Boto3, handling dependencies like subnets and internet gateways.
- **Scope**: Replace only the default VPC deletion logic with an AWS-native solution, preserving other provisioning tasks.
- **Question**: Why extract the deletion logic if it's already automated?

2.2 Benefits of Moving to an AWS-Native Solution

Moving default VPC deletion to an AWS-native solution (e.g., AWS CloudFormation with Lambda) offers:

- **Reduced Maintenance**: AWS manages Lambda and CloudFormation, minimizing script upkeep compared to custom Python code.
- **Infrastructure-as-Code (IaC)**: CloudFormation provides version-controlled, auditable deployments, unlike imperative scripts.
- **Scalability**: AWS StackSets enable deployment across multiple accounts via AWS Organizations.
- **Reliability**: Lambda's built-in error handling and CloudWatch logging improve robustness.

- **Integration**: Seamlessly integrates with your framework via API calls or event triggers.
- **Compliance**: Combines with SCPs to restrict default VPC usage.

2.3 Trade-Offs

- **Setup Effort**: Initial creation of CloudFormation templates and Lambda functions.
- **Learning Curve**: Familiarity with CloudFormation/Lambda, though these are standard AWS tools.

2.4 Why Keep the Custom Framework?

- **Functionality**: If stable, your scripts handle deletion well.
- **Flexibility**: Custom logic for unique requirements.
- **Minimal Disruption**: No need to change a working system.

2.5 Can SCPs Prevent Default VPC Creation?

SCPs cannot prevent default VPC creation:

- **AWS-Managed Process**: AWS creates default VPCs in all regions during account provisioning, outside IAM actions SCPs control.
- **SCP Limitations**: SCPs restrict user/role actions (e.g., `ec2:RunInstances`) but not AWS's internal logic.
- **SCP Role**: Can restrict *usage* of default VPCs post-creation (e.g., deny resource deployment).

2.6 Evaluation of AWS-Native Alternatives

1. **CloudFormation with Lambda** (Recommended): IaC, scalable, robust, integrates with existing framework.
2. **AWS CLI Scripts**: Simple but not IaC, manual maintenance.
3. **Systems Manager Automation**: Less flexible than Lambda for region iteration.
4. **Control Tower**: Limited to landing zone region, not all regions.
5. **AWS SDK (Boto3)**: Similar to current approach, not IaC.

2.7 Key Requirements

- Automate default VPC deletion in all regions.
- Integrate with existing Python/PowerShell framework.
- Use AWS-native tools.

- Scale to multiple accounts.
- Ensure security via IAM roles.
- Leverage existing SCPs for compliance.
- Handle dependencies to avoid errors.

3 Proposed Solution

Use **AWS CloudFormation with a Lambda-backed custom resource** to replace default VPC deletion logic, triggered by your existing framework. An SCP restricts default VPC usage, leveraging your existing SCPs.

3.1 Solution Components

1. **Lambda Function**: Python-based, deletes default VPCs across all regions, handling dependencies.
2. **CloudFormation Template**: Deploys Lambda and IAM role, triggers deletion.
3. **Integration**: Trigger stack creation from your framework via AWS SDK, SNS, or EventBridge.
4. **StackSets (Optional)**: Deploy to multiple accounts in AWS Organizations.
5. **SCP**: Deny resource creation in default VPCs.

3.2 Implementation Steps

1. Create and deploy CloudFormation stack with Lambda function (artifacts below).
2. Integrate with your framework using a Boto3 `create_stack` call or SNS/EventBridge trigger.
3. Optionally use StackSets for multi-account deployment.
4. Apply SCP to restrict default VPC usage.
5. Test in a single account and monitor via CloudWatch.

3.3 Artifacts

3.3.1 Lambda Function Code

This Python Lambda function deletes default VPCs across all regions.

```
import boto3
import cfnresponse

def delete_default_vpc(region):
    ec2 = boto3.client('ec2', region_name=region)
    vpcs = ec2.describe_vpcs(Filters=[{'Name': 'isDefault', 'Values'
        : ['true']}])
    if not vpcs['Vpcs']:
```

```
        return f"No default VPC in {region}"

    vpc_id = vpcs['Vpcs'][0]['VpcId']
    ec2.create_tags(Resources=[vpc_id], Tags=[{'Key': 'isDefault', '
       Value': 'true'}])
    igws = ec2.describe_internet_gateways(Filters=[{'Name': '
       attachment.vpc-id', 'Values': [vpc_id]}])
    for igw in igws.get('InternetGateways', []):
        igw_id = igw['InternetGatewayId']
        ec2.detach_internet_gateway(InternetGatewayId=igw_id, VpcId=
           vpc_id)
        ec2.delete_internet_gateway(InternetGatewayId=igw_id)
    subnets = ec2.describe_subnets(Filters=[{'Name': 'vpc-id', '
       Values': [vpc_id]}])
    for subnet in subnets.get('Subnets', []):
        ec2.delete_subnet(SubnetId=subnet['SubnetId'])
    ec2.delete_vpc(VpcId=vpc_id)
    return f"Deleted default VPC {vpc_id} in {region}"

def lambda_handler(event, context):
    try:
        request_type = event['RequestType']
        if request_type in ['Create', 'Update']:
            regions = boto3.client('ec2').describe_regions()['
               Regions']
            for region in regions:
                delete_default_vpc(region['RegionName'])
            cfnresponse.send(event, context, cfnresponse.SUCCESS,
               {})
        else:
            cfnresponse.send(event, context, cfnresponse.SUCCESS,
               {})
    except Exception as e:
        cfnresponse.send(event, context, cfnresponse.FAILED, {'Error
           ': str(e)})
```

3.3.2 CloudFormation Template

This YAML template deploys the Lambda function and IAM role.

```
Resources:
  DeleteVPCFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.lambda_handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import boto3
          import cfnresponse

```

```
          def delete_default_vpc(region):
              ec2 = boto3.client('ec2', region_name=region)
              vpcs = ec2.describe_vpcs(Filters=[{'Name': 'isDefault
                 ', 'Values': ['true']}])
              if not vpcs['Vpcs']:
                  return f"No default VPC in {region}"

              vpc_id = vpcs['Vpcs'][0]['VpcId']
              ec2.create_tags(Resources=[vpc_id], Tags=[{'Key': '
                 isDefault', 'Value': 'true'}])
              igws = ec2.describe_internet_gateways(Filters=[{'Name
                 ': 'attachment.vpc-id', 'Values': [vpc_id]}])
              for igw in igws.get('InternetGateways', []):
                  igw_id = igw['InternetGatewayId']
                  ec2.detach_internet_gateway(InternetGatewayId=
                     igw_id, VpcId=vpc_id)
                  ec2.delete_internet_gateway(InternetGatewayId=
                     igw_id)
              subnets = ec2.describe_subnets(Filters=[{'Name': 'vpc-
                 id', 'Values': [vpc_id]}])
              for subnet in subnets.get('Subnets', []):
                  ec2.delete_subnet(SubnetId=subnet['SubnetId'])
              ec2.delete_vpc(VpcId=vpc_id)
              return f"Deleted default VPC {vpc_id} in {region}"

          def lambda_handler(event, context):
              try:
                  request_type = event['RequestType']
                  if request_type in ['Create', 'Update']:
                      regions = boto3.client('ec2').describe_regions
                         ()['Regions']
                      for region in regions:
                          delete_default_vpc(region['RegionName'])
                      cfnresponse.send(event, context, cfnresponse.
                         SUCCESS, {})
                  else:
                      cfnresponse.send(event, context, cfnresponse.
                         SUCCESS, {})
              except Exception as e:
                  cfnresponse.send(event, context, cfnresponse.
                     FAILED, {'Error': str(e)})
      Runtime: python3.9
      Timeout: 300
  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
```

```
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: VPCDeletePolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - ec2:DescribeRegions
                  - ec2:DescribeVpcs
                  - ec2:DescribeInternetGateways
                  - ec2:DescribeSubnets
                  - ec2:DetachInternetGateway
                  - ec2:DeleteInternetGateway
                  - ec2:DeleteSubnet
                  - ec2:DeleteVpc
                  - ec2:CreateTags
                Resource: '*'
              - Effect: Allow
                Action:
                  - logs:CreateLogGroup
                  - logs:CreateLogStream
                  - logs:PutLogEvents
                Resource: '*'
  CustomResource:
    Type: AWS::CloudFormation::CustomResource
    Properties:
      ServiceToken: !GetAtt DeleteVPCFunction.Arn
```

3.3.3 SCP to Restrict Default VPC Usage

This JSON SCP denies resource creation in tagged default VPCs.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "ec2:RunInstances",
        "rds:CreateDBInstance",
        "elasticloadbalancing:CreateLoadBalancer"
      ],
      "Resource": "arn:aws:ec2:*:*:vpc/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/isDefault": "true"
        }
      }
    }
```

```
  ]
}
```

3.3.4 Integration with Existing Framework

This Python snippet triggers the CloudFormation stack.

```python
import boto3

def trigger_vpc_deletion(account_id, region='us-east-1'):
    cf_client = boto3.client('cloudformation', region_name=region)
    stack_name = f"DeleteDefaultVPC-{account_id}"
    try:
        cf_client.create_stack(
            StackName=stack_name,
            TemplateBody=open('delete_default_vpc.yaml').read(),
            Capabilities=['CAPABILITY_IAM']
        )
        cf_client.get_waiter('stack_create_complete').wait(StackName
            =stack_name)
        print(f"Default VPC deletion stack deployed for account {
            account_id}")
    except cf_client.exceptions.AlreadyExistsException:
        print(f"Stack {stack_name} already exists")
    except Exception as e:
        print(f"Error deploying stack: {e}")

# Call after account provisioning
trigger_vpc_deletion(new_account_id)
```

3.4 Benefits

- **Lower Maintenance**: AWS manages Lambda and CloudFormation.
- **IaC**: CloudFormation ensures consistency and auditability.
- **Scalability**: StackSets for multi-account deployment.
- **Reliability**: Lambda's error handling and CloudWatch logging.
- **Compliance**: Integrates with existing SCPs.
- **Minimal Disruption**: Replaces only deletion logic.

3.5 Considerations

- **Permissions**: Ensure IAM roles have `ec2:Describe*`, `ec2:Delete*`, `ec2:CreateTags`, and `cloudformation:CreateStack`.
- **Dependencies**: Add checks for resources (e.g., EC2 instances) if needed.
- **Regions**: Uses `describe-regions` for full coverage.

- **Restoring VPCs**: Requires AWS Support to restore deleted VPCs.
- **SCP Tagging**: Remove tagging logic if not needed.

4 Conclusion

SCPs cannot prevent default VPC creation but can restrict usage. Replacing the default VPC deletion logic with **AWS CloudFormation and Lambda** reduces maintenance, aligns with AWS best practices, and integrates with your existing Python/PowerShell framework via a simple trigger. The provided artifacts ensure a robust, scalable solution with minimal disruption.