I'm not a robot

tags {Name = "test"}}###CloudFormationAWSTemplateFormatVersion: '2010-09-09'Resources:InstanceSecurityGroup:Type: 'AWS::EC2::SecurityGroup'Properties:GroupDescription: Enable access to AutomateSecurityGroupIngress:- IpProtocol: tcpFromPort: '22'ToPort: '22'CidrIp: "0.0.0.0/0"VpcId: "vpc-xxxxx"Debug Output2017-11-02T10:35:07.946-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: ----------2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: 2017/11/02 10:35:08 [DEBUG] [aws-sdk-go] DEBUG: Response cloudformation/CreateStack Details:2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: ---[RESPONSE]--------------------------------------2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: HTTP/1.1 400 Bad Request2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: Connection: close2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: Content-Length: 3022017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: Content-Type: text/xml2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: Date: Thu, 02 Nov 2017 15:35:06 GMT2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: X-Amzn-Requestid: 671462e3-bfe3-11e7-8eb2-57d99139006e2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4:2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4:2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: ----------2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: 2017/11/02 10:35:08 [DEBUG] [aws-sdk-go] 2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: 2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: Sender2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: ValidationError2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: Template format error: unsupported structure.2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: 2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: 671462e3-bfe3-11e7-8eb2-57d99139006e2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: 2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: 2017/11/02 10:35:08 [DEBUG] [aws-sdk-go] DEBUG: Validate Response cloudformation/CreateStack failed, not retrying, error ValidationError: Template format error: unsupported structure.2017-11-02T10:35:08.674-0500 [DEBUG] plugin.terraform-provider-aws_v1.2.0_x4: status code: 400, request id: 671462e3-bfe3-11e7-8eb2-57d99139006e2017/11/02 10:35:08 [TRACE] root: eval: *terraform.EvalWriteState While trying to create a stack, I'm keeping getting this error:operation error CloudFormation: CreateStack, https response error StatusCode: 400, RequestID: xxxx-xxxx-xxxx, api error ValidationError: Template format error: unsupported structure.Since the same template file works fine with aws-cli, I guess it's something to do with the TemplateBody field.I also try the validate template api, and I get the same error again while cli works well.Expected Behaviorno error messageCurrent Behavioroperation error CloudFormation: CreateStack, https response error StatusCode: 400, RequestID: xxxx-xxxx-xxxx, api error ValidationError: Template format error: unsupported structure.Reproduction Stepsokaws cloudformation validate-template --template-body file:///absolute/path/template.yamlfailedpackage main import ("context" "log" "github.com/aws/aws-sdk-go-v2/aws" "github.com/aws/aws-sdk-go-v2/config" "github.com/aws/aws-sdk-go-v2/service/cloudformation") func main() { cfg, _ := config.LoadDefaultConfig(context.TODO()) client := cloudformation.NewFromConfig(cfg) _, err := client.ValidateTemplate(context.TODO(), &cloudformation.ValidateTemplateInput{ TemplateBody: aws.String("file:///absolute/path/template.yaml"), }) if err != nil { log.Fatal(err)}}Possible SolutionNo responseAdditional Information/ContextNo responseAWS Go SDK V2 Module Versions Usedgithub.com/aws/aws-sdk-go-v2 v1.18.1github.com/aws/aws-sdk-go-v2/config v1.18.27github.com/aws/aws-sdk-go-v2/credentials v1.13.26github.com/aws/aws-sdk-go-v2/service/cloudformation v1.30.0 August 2017 This stumped me for a few hours yesterday while wanting to deploy a cloudformation stack to AWS via PowerShell using New-CFNStack instead of the console, I kept getting the following error: Test-CFNTemplate : Template format error: unsupported structure. Not overly helpful, I checked the path was correct, used relative and full file paths and tried with file:// before the file name but the error was still the same. To check that there wasnt an error in my cloudformation yaml file, I deployed it successfully via the AWS console and thought Id try the Test-CFNTemplate command that accepted the -TemplateBody parameter but still failed with the same error message as New-CFNStack Next I tried: Test-CFNTemplate -TemplateBody (get-content .\amazon-linux-vm.yaml) This gave the error:Test-CFNTemplate : Cannot convert 'System.Object[]' to the type 'System.String' required by parameter 'TemplateBody'. What gives here?Get-Content reads the file in and creates a list of objects for each line. If you were to save get content to a variable, then access the first item, youd get the whole first line of the file returned.$template = Get-Content -Path .\amazon-linux-vm.yaml$template[0]AWSTemplateFormatVersion: 2010-09-09Use the -Raw switchThe Raw switch, although not well documented at present, will read in the whole file as one big string object instead of a separate object per line. To demonstrate this:$template1 = Get-Content -Path .\amazon-linux-vm.yaml$template1[0]A See this post about Get-Content and why its not your friend on powershell.org To read the file without line breaks, use the -Raw switch and the cmdlet should now work correctly. Test-CFNTemplate -TemplateBody (Get-Content .\amazon-linux-vm.yaml -Raw) The Test-CFNTemplate will now run correctly and youll be able to use New-CFNStack to deploy your resources via cloudformation with PowerShell making it easy to change the values of the parameters you pass in. Tags: aws, cloudformation, powershell Categories: aws Updated: August 25, 2017 This is unbelievably confusing and a very poor design choice. While aws cloudformation validate-template --template-body S3_Bucket.template fails with the extremely cryptic error message A client error (ValidationError) occurred when calling the ValidateTemplate operation: Template format error: unsupported structure. The following command aws cloudformation validate-template --template-body file://S3_Bucket.template --region us-west-1 succeeds. In addition, the error message is extremely misleading, as it implies that parsing was attempted and failed, not that the document was not found/failed to be loaded. There should either be A more informative error message: for example requiring the protocol as a prefix and erring when one is not specified, or;The --template-body argument should be interpreted as a local file URI by default.Choose one of the following solutions based on the error message that you receive:For "JSON not well-formed" or "YAML not well-formed" errors, see the Validate template syntax section.For "Unresolved resource dependencies [XXXXXXXX] in the Resources block of the template" errors, see the Validate logical IDs and parameters section.For "Unrecognized parameter type: XXXXXXXX" or "Invalid template parameter property 'XXXXXXXX'" errors, see the Validate parameter definitions section.For "Every Condition member must be a string" errors, see the Confirm that Conditions is specified as a string section.For "Unrecognized resource types: [XXXXXXXX]" errors, see the Verify the availability of your resource type section.For "The [environmental resource] 'XXXXXXXX' does not exist" errors, see the Verify that your resource exists outside the stack, or validate dependencies for resources in the same stack section.For "Invalid template property or properties [XXXXXXXX]" errors, see the Verify template properties section.For "Invalid policy syntax" or "MalformedPolicy" errors, see the Verify policy syntax for any IAM policy related resources section.ResolutionIf you receive errors when you run AWS Command Line Interface (AWS CLI) commands, then see Troubleshoot AWS CLI errors. Also, make sure that you're using the most recent AWS CLI version.Validate template syntaxTo follow proper JSON or YAML syntax in your CloudFormation template, consider the following: Validate logical IDs and parametersConfirm that resource logical IDs and parameters are defined in your template.In the following JSON and YAML templates, test is referenced for the ImageId property. However, neither template includes a resource logical ID nor a parameter namedtest. These templates return the following error: "Unresolved resource dependencies [test] in the Resources block of the template." For more information on resource definitions and their syntax, see Resources.Example JSON (incorrect):{ "Resources" : { "EC2Instance01" : { "Type" : "AWS::EC2::Instance", "Properties" : { "ImageId" : {"Ref": "test"} } } }}Example YAML (incorrect):Resources: EC2Instance01: Type: AWS::EC2::Instance Properties: ImageId: !Ref testTo resolve this issue, add a resource logical ID that's named test. Or, create a parameter that's named test where the reference returns the ImageId value. The following example JSON and YAML templates include a parameter with the name test and ImageId as the value.Example JSON (correct):{ "Parameters": { "test": { "Type": "String", "Default": "ami-xxx" } }, "Resources" : { "EC2Instance01" : { "Type" : "AWS::EC2::Instance", "Properties" : { "ImageId" : {"Ref": "test"} } } }}Example YAML (correct):Parameters: test: Type: String Default: ami-xxxResources: EC2Instance01: Type: 'AWS::EC2::Instance' Properties: ImageId: !Ref testValidate parameter definitions In the following example JSON and YAML templates, the default value for ParameterC has the intrinsic function Fn::Sub. This intrinsic function causes the validation error: "Every Default member must be a string."Example JSON (incorrect):{ "Parameters": { "ParameterA": { "Type": "String", "Default": "abc" }, "ParameterB": { "Type": "String", "Default": "def" }, "ParameterC": { "Type": "String", "Default": { "Fn::Sub": "${ParameterA}-${ParameterB}" } } }, "Resources": { "MyS3Bucket": { "Type": "AWS::S3::Bucket", "Properties": { "BucketName": { "Ref": "ParameterC" } } } }}Example YAML (incorrect):Parameters: ParameterA: Type: String Default: abc ParameterB: Type: String Default: def ParameterC: Type: String Default: !Sub '${ParameterA}-${ParameterB}'Resources: MyS3Bucket: Type: 'AWS::S3::Bucket' Properties: BucketName: !Ref ParameterCConfirm that Conditions is specified as a stringIn your CloudFormation template, specify Conditions as a string.The following example JSON and YAML templates specify the condition in the resource EC2RouteA as a list of strings instead of a single string. These templates result in the following validation error: "Every Condition member must be a string."Example JSON (incorrect):{ "Conditions": { "ConditionA": { "Fn::Not": [{ "Fn::Equals": ["", "Sample"] }] }, "ConditionB": { "Fn::Not": [{ "Fn::Equals": ["", "Sample"] }] } }, "Resources": { "EC2RouteA": { "Type": "AWS::EC2::Route", "Condition": ["ConditionA", "ConditionB"], "Properties": { ... } } }}Example YAML (incorrect):Conditions: ConditionA: !Not - !Equals - '' - Sample ConditionB: !Not - !Equals - '' - SampleResources: EC2RouteA: Type: 'AWS::EC2::Route' Condition: - ConditionA - ConditionB Properties:To resolve this error, add ConditionAandB to the Conditions section of your template, and then use ConditionAandB as the condition for the EC2RouteA resource. See the following example JSON and YAML templates.Example JSON (correct):{ "Conditions": { "ConditionA": { "Fn::Not": [{ "Fn::Equals": ["", "Sample"] }] }, "ConditionB": { "Fn::Not": [{ "Fn::Equals": ["", "Sample"] }] }, "ConditionAandB": { "Fn::And": [{ "Condition": "ConditionA" }, { "Condition": "ConditionB" }] } }, "Resources": { "EC2RouteA": { "Type": "AWS::EC2::Route", "Condition": "ConditionAandB", "Properties": { ... } } }}Example YAML (correct):Conditions: ConditionA: Fn::Not: - Fn::Equals: - '' - Sample ConditionAandB: Fn::And: - Condition: ConditionA - Condition: ConditionBResources: EC2RouteA: Type: AWS::EC2::Route Condition: ConditionAandB Properties:Verify the availability of your resource type1. Verify that your resource is available in your AWS Region.Not all resource types are available in every AWS Region. Templates that include resource types that aren't available in your AWS Region result in the following error: "Unrecognized resource types: [XXXXXXXX]."2. If your template consists of any serverless resources, then include a Transform declaration. See the following example JSON and YAML templates.Example JSON:{ "Transform": "AWS::Serverless-2016-10-31", #Please make sure to include this. "Resources": { "MyServerlessFunctionLogicalID": { "Type": "AWS::Serverless::Function", "Properties": { "Handler": "index.handler", "Runtime": "nodejs8.10", "CodeUri": "s3://testBucket/mySourceCode.zip" } } }}Example YAML:Transform: AWS::Serverless-2016-10-31 #Please make sure to include this.Resources: MyServerlessFunctionLogicalID: Type: AWS::Serverless::Function Properties: Handler: index.handler Runtime: nodejs8.10 CodeUri: 's3://testBucket/mySourceCode.zip'Verify that your resource exists outside the stack, or validate dependencies for resources in the same stackIf you're hardcoding a resource or Amazon Resource Name (ARN) into one of your stack's resources for one that's outside of the CloudFormation stack, then verify the following:The resource name or ARN is correct.The resource exists.The resource exists in the same AWS Region as the stack. Consider that some resources accept properties across AWS Regions or accounts.For example, an AWS::EC2::Instance resource in your stack that specifies a security group (sg-1234567890) fails if:The security group doesn't exist.The security group doesn't exist in the stack's AWS Region.As a result, you receive the error message: "The sg-1234567890 does not exist." See the following example:LinuxInstance: Type: AWS::EC2::Instance Properties: SubnetId: !Ref ServerSubnetID KeyName: !Ref EC2KeyPairName SecurityGroupIds: sg-1234567890 #Verify template propertiesUse only permitted template properties in your CloudFormation template.The following example JSON and YAML templates set the bucket resource on the same level as theResources section. This returns the following error: "Template validation error: Invalid template property or properties [Bucket]." This error is caused when the CloudFormation template validator sees the bucket resource as a section-level specification. A section-level specification isn't allowed as a template property.Example JSON (incorrect):{ "Resources": { "WaitCondition": { "Type": "AWS::CloudFormation::WaitCondition" } }, # "Bucket": { "Type": "AWS::S3::Bucket", "Properties": { "Name": "BucketName" } }}Example YAML (incorrect):Resources: WaitCondition: Type: AWS::CloudFormation::WaitConditionBucket: # Type: AWS::S3::Bucket Properties: Name: BucketNameTo resolve this issue, correct the formatting so that the bucket resource is specified inside the Resources section. See the following example JSON and YAML templates that are correctly formatted.Example JSON (correct):{ "Resources": { "WaitCondition": { "Type": "AWS::CloudFormation::WaitCondition" }, "Bucket": { "Type": "AWS::S3::Bucket", "Properties": { "Name": "BucketName" } } }}Example YAML (correct):Resources: WaitCondition: Type: 'AWS::CloudFormation::WaitCondition' Bucket: Type: 'AWS::S3::Bucket' Properties: Name: BucketNameVerify policy syntax for any IAM policy-related resourcesIf you're creating an Identity and Access Management (IAM) policy resource or related configuration in your resource properties, verify that the policy is valid with this structure base.{ "Resources": { "Policy": { "Type": "AWS::IAM::Policy", "Properties": { "PolicyName": "IamPolicyName", "PolicyDocument": { "Version": "2012-10-17", "Statement": [{ "Effect": "effect", "Action": [":", ""], "Resource": "desiredResourceARN", "Condition": { "ConditionConfiguration": { "conditionKey": ["values"] }, "ConditionConfiguration2": "" } }] } } } }}Note: Replace with a service name of your choice. Replace with the API action for your selected service. For more information, seeIAM JSON policy.Integrate your JSON policy document with a YAML formatYou might want to integrate a JSON policy document with a YAML format template for provisioning CloudFormation. This requires you to change how the document appears in the template.After integration, the policy elements look similar to what's shown below:Resources: Policy: Type: 'AWS::IAM::Policy' Properties: PolicyName: IamPolicyName PolicyDocument: Version: 2012-10-17 Statement: - Effect: effect Action: - ':' - Resource: desiredResourceARN Condition: ConditionConfiguration: conditionKey: - values ConditionConfiguration2: Hi everyone,I'm trying to deploy multi-region StackSets but get the following error:Template format error: Unrecognized resource types: [AWS::SES::Template]The resource type definition is correct according to the CloudFormation documentation and previous tests I executed with standard CloudFormation stacks. Using simple stacks within the region of choice the template the resources get provisioned successfully. But why is CloudFormation StackSet unable to deploy the AWS::SES::Template resource?The region of choice is eu-west-1 if this makes any difference.Would be great to get some help and if it is a bug to address this to the AWS support team.Best regards and thanks in advance!NewestMost votesMost commentsWhich regions are you deploying the Stack Set to? If you try to deploy to a region which doesn't has SES or support that resource, you'll get that error. You should limit your Stack Set deployment to just supported regions. You could also (if there are other resources in the template you still want to deploy) use a Condition to only deploy the Template in a suitable region(s).For reference, I quickly put together a minimal template with the AWS::SES::Template resource in, and checked the region support with the cfn-lint tool:% cat ses.yamlAWSTemplateFormatVersion: 2010-09-09Resources: Template: Type: AWS::SES::Template Properties: Template: HtmlPart: Testing SubjectPart: Testing TemplateName: Testing TextPart: Testing% cfn-lint -r ALL_REGIONS -- ses.yamlE3001 Invalid or unsupported Type AWS::SES::Template for resource Template in af-south-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in ap-east-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in ap-northeast-2ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in ap-northeast-3ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in ap-south-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in ap-southeast-2ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in ap-southeast-3ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in ca-central-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in cn-north-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in cn-northwest-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in eu-central-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in eu-west-2ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in sa-east-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in us-gov-east-1ses.yaml:5:5E3001 Invalid or unsupported Type AWS::SES::Template for resource Template in us-gov-west-1ses.yaml:5:5As per the documentation, the resource type is correct, and also the region you selected is supported. Please refer to link {1}But Before you perform "Multi-region Deployment With AWS Cloudformation Stacksets", you need to set up permissions for cloud formation stacksets.AWS CloudFormation StackSets requires specific permissions to be able to deploy stacks in multiple AWS accounts across multiple AWS Regions. It needs an administrator role that is used to perform StackSets operations, and an execution role to deploy the actual stacks in target accounts.These roles require specific naming conventions:***AWSCloudFormationStackSetAdministrationRole ***for the administrator role, and **AWSCloudFormationStackSetExecutionRole for the execution role.Note: StackSets execution will fail if either of these roles are missing.The ***AWSCloudFormationStackSetAdministrationRole ***should be created in the account where you are creating the StackSet.The ***AWSCloudFormationStackSetExecutionRole ***should be created in each target account where you wish to deploy the stack.{1} hope this helps.answered 3 years agolg...Thank you for your support. The error is not permission related though. For verification: I already deployed SSM parameters using the same setup, even the same template file. The only new thing to the CloudFormation template file are the AWS::SES::Template resources, which CloudFormation StackSet doesn't seem to like or recognize.I already get the error message when I upload the CloudFormation template via AWS Management Console > CloudFormation > StackSets. Same happens programmatically.answered 3 years agolg... Reddit and its partners use cookies and similar technologies to provide you with a better experience. By accepting all cookies, you agree to our use of cookies to deliver and maintain our services and site, improve the quality of Reddit, personalize Reddit content and advertising, and measure the effectiveness of advertising. By rejecting non-essential cookies, Reddit may still use certain cookies to ensure the proper functionality of our platform. For more information, please see our Cookie Notice and our Privacy Policy. You can author CloudFormation templates in JSON or YAML formats. Both formats serve the same purpose but offer distinct advantages in terms of readability and complexity. JSON JSON is a lightweight data interchange format that's easy for machines to parse and generate. However, it can become cumbersome for humans to read and write, especially for complex configurations. In JSON, the template is structured using nested braces {} and brackets [] to define resources, parameters, and other components. Its syntax requires explicit declaration of every element, which can make the template verbose but ensures strict adherence to a structured format. YAML YAML is designed to be more human-readable and less verbose than JSON. It uses indentation rather than braces and brackets to denote nesting, which can make it easier to visualize the hierarchy of resources and parameters. YAML is often preferred for its clarity and ease of use, especially when dealing with more complex templates. However, YAML's reliance on indentation can lead to errors if the spacing is not consistent, which requires careful attention to maintain accuracy. Template structure CloudFormation templates are divided into different sections, and each section is designed to hold a specific type of information. Some sections must be declared in a specific order, and for others, the order doesn't matter. However, as you build your template, it can be helpful to use the logical order shown in the following examples because values in one section might refer to values from a previous section. When authoring templates, don't use duplicate major sections, such as the Resources section. Although CloudFormation might accept the template, it will have an undefined behavior when processing the template, and might incorrectly provision resources, or return inexplicable errors. The following example shows the structure of a JSON-formatted template with all available sections. { "AWSTemplateFormatVersion" : "version date" "Description" : "JSON string", "Metadata" : { template metadata }, "Parameters" : { set of parameters }, "Rules" : { set of rules }, "Mappings" : { set of mappings }, "Conditions" : { set of conditions }, "Transform" : { set of transforms }, "Resources" : { set of resources }, "Outputs" : { set of outputs } } The following example shows the structure of a YAML-formatted template with all available sections. ---AWSTemplateFormatVersion: version date Description: String Metadata: template metadata Parameters: set of parameters Rules: set of rules Mappings: set of mappings Conditions: set of conditions Transform: set of transforms Resources: set of resources Outputs: set of outputs In JSON-formatted templates, comments are not supported. JSON, by design, doesn't include a syntax for comments, which means you can't add comments directly within the JSON structure. However, if you need to include explanatory notes or documentation, you can consider adding metadata. For more information, see Metadata attribute. In YAML-formatted templates, you can include inline comments by using the # symbol. The following example shows a YAML template with inline comments. AWSTemplateFormatVersion: 2010-09-09Description: A sample CloudFormation template with YAML comments.# Resources sectionResources: MyEC2Instance: Type: AWS::EC2::Instance Properties: # Linux AMI ImageId: ami-1234567890abcdef0 InstanceType: t2.micro KeyName: MyKey BlockDeviceMappings: - DeviceName: /dev/sdm Ebs: VolumeType: io1 Iops: 200 DeleteOnTermination: false VolumeSize: 20 Specifications CloudFormation supports the following JSON and YAML specifications: JSON CloudFormation follows the ECMA-404 JSON standard. For more information about the JSON format, see . YAML CloudFormation supports the YAML Version 1.1 specification with a few exceptions. CloudFormation doesn't support the following features: The binary, omap, pairs, set, and timestamp tags Aliases Hash merges For more information about YAML, see . Learn more For each resource you specify in your template, you define its properties and values using the specific syntax rules of either JSON or YAML. For more information about the template syntax for each format, see CloudFormation template sections. AWS CloudFormation is a powerful service designed to help you manage your infrastructure as code (IaC). However, as with any coding task, you might encounter errors while working with CloudFormation templates. Debugging these errors can be frustrating, particularly when you're in the middle of developing or deploying your infrastructure. This blog post aims to guide you through common CloudFormation template errors, their potential causes, and how to fix them effectively.What is CloudFormation?Before diving into the debugging process, lets briefly discuss what AWS CloudFormation is. AWS CloudFormation allows you to define your cloud resources in a declarative manner using JSON or YAML templates. This means you get to describe what your infrastructure should look like, and CloudFormation takes care of the how.Why Use CloudFormation?Using AWS CloudFormation provides numerous benefits:Infrastructure as Code (IaC): Improve consistency and reduce human error.Version Control: Just like application code, infrastructure can be versioned and rolled back.Automation: Simplify the process of resource provisioning and management.Template Reusability: Share templates with teams and utilize them in multiple environments.Understanding Common CloudFormation ErrorsNow, let's delve into common CloudFormation template errors you might encounter.1. JSON vs. YAML Format IssuesCloudFormation templates can be written in either JSON or YAML format. Mixing these formats within the same template or incorrectly formatting them will lead to validation errors.Example Template:Resources: MyBucket: Type: AWS::S3::Bucket Properties: BucketName: MyUniqueBucketNameCommon Issue:If the above snippet were written in JSON but failed to correctly format the syntax, you might get an error like:Error: Unable to parse JSON template. Invalid JSON format.Solution: Always ensure consistency in your format. Using a linter can help catch these issues early. YAML Lint is a popular choice for validating YAML.2. Incorrect Resource TypesAnother common pitfall is specifying resource types incorrectly. All resources listed in a template must be valid CloudFormation resource types.Example:{ "Resources": { "MyBucket": { "Type": "AWS::S3::Bkt", // Incorrect resource type "Properties": { "BucketName": "MyUniqueBucketName" } } }}Error Message:"An error occurred (ValidationError) when calling the CreateStack operation: Template format error: Unrecognized resource types: [AWS::S3::Bkt]"Solution: Make sure to consult the AWS Resource Types Reference to verify resource types.3. Missing Required PropertiesOmitting required properties for resources defined in the template can lead to deployment failures.Example:Resources: MyBucket: Type: AWS::S3::Bucket Properties: {} # Required property omitted Error Message:"An error occurred (ValidationError) when calling the CreateStack operation: Template validation failed: The Resource of type 'AWS::S3::Bucket' must contain the property 'BucketName'."Solution: Always refer to the documentation for the specific resource you are using to identify required properties.4. Dependency IssuesSometimes, resources depend on each other. If you declare these dependencies incorrectly, CloudFormation may fail to create stacks.Example:Resources: MyBucket: Type: AWS::S3::Bucket Properties: BucketName: MyUniqueBucketName MyBucketPolicy: Type: AWS::S3::BucketPolicy Properties: Bucket: !Ref MyBucket PolicyDocument: Version: "2012-10-17" Statement: - Effect: "Allow" Principal: "*" Action: "s3:GetObject" Resource: !Sub "${MyBucket.Arn}/*"Error Message:"An error occurred (ValidationError) when calling the CreateStack operation: Template contains circular dependency between resources."Solution: Use the DependsOn attribute to define an explicit ordering if needed. Here's an example:Resources: MyBucket: Type: AWS::S3::Bucket Properties: BucketName: MyUniqueBucketName MyBucketPolicy: Type: AWS::S3::BucketPolicy DependsOn: MyBucket # This is the key fix. Properties: ...5. Parameter Type MismatchesParameters are defined at the start of your template and can accept inputs when stacks are created. Incorrect usage of parameter types can lead to errors.Example:Parameters: InstanceType: Type: String # Should be: Type: 'AWS::EC2::InstanceType'Error Message:"An error occurred (ValidationError) when calling the CreateStack operation: Template format error: Parameter instance 'InstanceType' value must be of type list and value cannot be null."Solution: Always ensure the parameters match the expected CloudFormation parameter types. Check the AWS CloudFormation Parameters documentation for details.6. Intrinsic Function ErrorsIntrinsic functions such as !Ref, !GetAtt, and !Sub allow dynamic references in your templates. Improper usage can lead to failures.Example:Outputs: BucketName: Value: !Ref "MyBucket" # Correct usage Description: "The name of the S3 Bucket"Error Message:"An error occurred (ValidationError) when calling the CreateStack operation: Template format error: Invalid intrinsic function or reference."Solution: Review the correctness of your intrinsic functions. Make sure placeholders are appropriately used.Debugging TipsUse the AWS CloudFormation Designer: AWS provides a graphical interface to help visualize CloudFormation templates. This can help in identifying misconfigurations.Review the Events Tab: When troubleshooting, check the Events tab in the AWS CloudFormation console for detailed error messages.Use AWS CLI for Validation: You can validate a CloudFormation template using the AWS CLI:aws cloudformation validate-template --template-body file://template.yamlLeverage Stack Overflow & AWS Forums: If you're stuck, theres a high chance someone else has encountered a similar issue.Final ThoughtsDebugging CloudFormation template errors can be daunting. However, with proper strategies validating syntax, understanding intrinsic functions, and correctly defining resource types and parameters you can efficiently troubleshoot and ensure successful deployments.By understanding common pitfalls and knowing how to resolve them, youll save yourself time and avoid potential frustration. For a deeper dive into infrastructure as code principles, check AWS CloudFormation User Guide and begin writing effective CloudFormation templates today!Feel free to comment below if you have any specific questions or experiences you would like to share regarding CloudFormation errors! Happy coding!

Template format error unsupported structure. Aws cloudformation unsupported structure. Aws cloudformation template format error unsupported structure. Cfn template format error unsupported structure.