**TDIU Report Service: CloudFormation Implementation Guide**

**Introduction to CloudFormation**

AWS CloudFormation provides a way to model and manage your AWS resources using "infrastructure as code." For the TDIU Report Service, CloudFormation offers several key advantages:

1. **Complete Documentation**: Maintains a single source of truth for all AWS resources
2. **Consistent Updates**: Ensures all new services follow the same architecture patterns
3. **Version Control**: Allows tracking of infrastructure changes over time
4. **Simplified Sharing**: Provides an easy way to communicate current system status between work sessions
5. **Multi-Service Support**: Facilitates adding new revenue streams as planned in the business model

**Step-by-Step Implementation Process**

**Step 1: Create the Initial Template File**

1. Open a text editor (VS Code, Notepad++, etc.)
2. Create a new file named tdiu-infrastructure.yaml
3. Start with the basic CloudFormation structure:

yaml

AWSTemplateFormatVersion: '2010-09-09'

Description: 'TDIU Report Service Infrastructure'

Resources:

*# Resources will be defined here*

**Step 2: Document Existing S3 Buckets**Add your existing S3 buckets to the template:

yaml

Resources:

DocumentStorageBucket:

Type: AWS::S3::Bucket

Properties:

BucketName: tdiu-document-storage BucketEncryption:

ServerSideEncryptionConfiguration: - ServerSideEncryptionByDefault: SSEAlgorithm: AES256 VersioningConfiguration: Status: Enabled

PublicAccessBlockConfiguration: BlockPublicAcls: true BlockPublicPolicy: true IgnorePublicAcls: true

RestrictPublicBuckets: true

TemplatesBucket:

Type: AWS::S3::Bucket

Properties:

BucketName: tdiu-templates BucketEncryption:

ServerSideEncryptionConfiguration: - ServerSideEncryptionByDefault: SSEAlgorithm: AES256 VersioningConfiguration: Status: Enabled

PublicAccessBlockConfiguration: BlockPublicAcls: true BlockPublicPolicy: true IgnorePublicAcls: true

RestrictPublicBuckets: true

CompletedReportsBucket:

Type: AWS::S3::Bucket

Properties:

BucketName: tdiu-completed-reports BucketEncryption:

ServerSideEncryptionConfiguration: - ServerSideEncryptionByDefault: SSEAlgorithm: AES256 VersioningConfiguration:

Status: Enabled

PublicAccessBlockConfiguration: BlockPublicAcls: true

BlockPublicPolicy: true
  
IgnorePublicAcls: true
  
RestrictPublicBuckets: true

**Step 3: Document Existing Lambda Functions** Add your Lambda functions to the template:

yaml

GenerateUploadUrlFunction:

Type: AWS::Lambda::Function

Properties:

FunctionName: TDIU-GenerateUploadUrl

Runtime: python3.9

Handler: index.handler

Role: !GetAtt LambdaExecutionRole.Arn

Code:

ZipFile: |

def handler(event, context):

# Function code would go here

return {

'statusCode': 200,

'body': 'This is a placeholder'

}

*### Step 4: Document IAM Role and CloudTrail*Add the JAM role and CloudTrail configuration:

```yaml

*# IAM Role for Lambda Functions*

LambdaExecutionRole:

Type: AWS::IAM::Role

Properties:

RoleName: TDIU-LambdaExecutionRole

AssumeRolePolicyDocument:

Version: '2012-10-17'

Statement:

- Effect: Allow

Principal:

Service: [lambda.amazonaws.com](http://lambda.amazonaws.com)

Action: sts:AssumeRole

ManagedPolicyArns:

- arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole

- arn:aws:iam::aws:policy/AmazonS3FullAccess

- arn:aws:iam::aws:policy/AmazonBedrockFullAccess

*# CloudTrail*

ComplianceTrail:

Type: AWS::CloudTrail::Trail

Properties:

TrailName: TDIU-Compliance-Trail

IsLogging: true

S3BucketName: !Ref TrailBucket
  
IncludeGlobalServiceEvents: true
  
IsMultiRegionTrail: true
  
EnableLogFileValidation: true

TrailBucket:

Type: AWS::S3::Bucket

Properties:

BucketName: tdiu-cloudtrail-logs

VersioningConfiguration:

Status: Enabled

**Step 5: Save the Complete Template**



Save the complete template file ( tdiu-in-Frastructure.yaml ) with all resources defined.

**Importing Existing Resources**

Once your template is created, you'll need to import your existing resources:

**Step 1: Navigate to CloudFormation in AWS Console**

1. Log in to the AWS Management Console
2. Navigate to CloudFormation
3. Click "Create stack" > "With existing resources (import resources)"

**Step 2: Upload Your Template**

1. Choose "Upload a template file"
2. Upload your tdiu-in-Frastructure.yaml file
3. Click "Next"

**Step 3: Specify Stack Details**

1. Enter a stack name (e.g., "TDIU-Report-Service")
2. Click "Next"

**Step 4: Import Resources**

1. For each resource in your template, CloudFormation will ask you to identify the corresponding

existing resource

1. Select each resource from the dropdown menu:

|  |  |
| --- | --- |
|  | For S3 buckets, select the matching bucket names  For Lambda functions, select the matching function names  For IAM roles, select the matching role name  For CloudTrail, select the matching trail |

**Step 5: Review and Import**

1. Review the import preview
2. Click "Import resources" to create the stack

**Detecting and Resolving Drift**

Once your resources are imported, you should check for drift:

**Step 1: Select Your Stack**

1. In the CloudFormation console, select your stack
2. Click "Stack actions" > "Detect drift"

**Step 2: Review Drift Results**

1. Wait for the drift detection to complete
2. Review any detected drift (differences between your template and actual resources)

**Step 3: Update Your Template**If drift is detected:

1. Note the properties that are different
2. Update your CloudFormation template to match the actual resources
3. Save the updated template

**Adding New Resources to CloudFormation**

As you expand your infrastructure, add new resources to your CloudFormation template:

**Example: Adding Cognito User Pool**

yaml

CognitoUserPool:

Type: AWS::Cognito::UserPool
  
Properties:

UserPoolName: TDIU-UserPool AdminCreateUserConfig:

AllowAdminCreateUserOnly: true AutoVerifiedAttributes:

- email

MfaConfiguration: "ON"

Policies:

PasswordPolicy:

MinimumLength: 12

RequireLowercase: true
  
RequireNumbers: true
  
RequireSymbols: true
  
RequireUppercase: true

CognitoUserPoolClient:

Type: AWS::Cognito::UserPoolClient

Properties:

ClientName: TDIU-App-Client

GenerateSecret: false

UserPoolId: !Ref CognitoUserPool

ExplicitAuthFlows:

- ALLOW\_USER\_PASSWORD\_AUTH

- ALLOW\_REFRESH\_TOKEN\_AUTH

**Example: Adding API Gateway**

yaml

ApiGateway:

Type: AWS::ApiGateway::RestApi

Properties:

Name: TDIU-API

Description: API for TDIU Report Service

EndpointConfiguration:

Types:

- REGIONAL

**Updating the Stack**

After adding new resources:

1. In the CloudFormation console, select your stack
2. Click "Update"
3. Choose "Replace current template"
4. Upload your updated template
5. Follow the wizard to update the stack

**Sharing Your Infrastructure Status**

To share your infrastructure status for a new Claude chat:

**Step 1: Export Your Template**

1. In the CloudFormation console, select your stack
2. Click on the "Template" tab
3. Click "View in Designer"
4. In the Designer, click the "Template" tab (bottom panel)
5. Copy the entire template

**Step 2: Use the Template in Your Session Start Prompt**Use this template for starting new chats:

I'm working on the TDIU Report Service project, an AWS-based HIPAA-compliant service for generating reports for veterans' attorneys. Here's my current infrastructure as defined in CloudFormation:

[PASTE YOUR CLOUDFORMATION TEMPLATE HERE]

Previous accomplishments:

1. [List 2-3 key things completed in previous sessions]

Current focus:

I'd like to [your specific goal for this session].

Based on my current infrastructure and the project plans, please guide me through the next steps to accomplish this goal.

**Best Practices for CloudFormation Management**

1. **Version Control**: Store your template in a version control system (e.g., Git)
2. **Parameter Usage**: Use parameters for values that might change
3. **Resource Naming**: Use consistent naming conventions
4. **Documentation**: Add comments to explain complex configurations
5. **Modular Templates**: Consider splitting into multiple templates as complexity grows
6. **Regular Updates**: Keep your template updated as you make changes

**Advanced CloudFormation Features**

As your infrastructure grows, you may want to use these advanced features:

1. **Parameters**

yaml

Parameters:

Environment:

Type: String

Default: Dev

AllowedValues:

- Dev

- Prod

Description: Environment type

Resources:

DocumentStorageBucket:

Type: AWS::S3::Bucket

Properties:

BucketName: !Sub "tdiu-document-storage-${Environment}"

1. **Outputs**

yaml

Outputs:

DocumentStorageBucketName:

Description: Name of the document storage bucket

Value: !Ref DocumentStorageBucket

ApiEndpoint:

Description: API Gateway endpoint URL

Value: !Sub "https://${ApiGateway}.execute-api.${AWS::Region}.amazonaws.com/prod"

**3. Nested Stacks**

As you add more services, you may want to use nested stacks:

yaml

Resources:

CoreInfrastructure:

Type: AWS::CloudFormation::Stack

Properties:

TemplateURL: <https://s3.amazonaws.com/bucket/core-infrastructure.yaml>

DocumentAnalysisService:

Type: AWS::CloudFormation::Stack

Properties:

TemplateURL: <https://s3.amazonaws.com/bucket/document-analysis.yaml>

Parameters:

CoreStackName: !Ref AWS::StackName

**Conclusion**

CloudFormation provides a powerful way to document and manage your AWS infrastructure. By maintaining your template as you build the TDIU Report Service, you'll have a clear record of your resources and can easily share your current status between work sessions.

This approach will become increasingly valuable as you add more services to your platform, ensuring consistent implementation and documentation of your growing infrastructure.

DocumentProcessorFunction:

Type: AWS::Lambda::Function

Properties:

FunctionName: TDIU-DocumentProcessor

Runtime: python3.9

Handler: index.handler

Role: !GetAtt LambdaExecutionRole.Arn

Code:

ZipFile: |

def handler(event, context):

# Function code would go here

return {

'statusCode': 200,

'body': 'This is a placeholder'

}': 'This is a placeholder'
  
}

CreateCaseFunction:

Type: AWS::Lambda::Function

Properties:

FunctionName: TDIU-CreateCase

Runtime: python3.9

Handler: index.handler

Role: !GetAtt LambdaExecutionRole.Arn

Code:

ZipFile: |

def handler(event, context):

# Function code would go here

return {

'statusCode': 200,

'body