TDIU Report Service: AWS Implementation & CloudFormation Strategy

Core Architecture Overview

The TDIU Report Service is built on AWS with a focus on security, scalability, and compliance. The architecture leverages AWS Bedrock (Claude AI) for document analysis and AWS services for secure document handling and processing.

Key Design Principles

1. **Security-First Design**: Enterprise-grade security with military-grade encryption
2. **HIPAA Compliance**: Full compliance through BAAs and comprehensive audit trails
3. **Scalable Architecture**: Design that grows from 20 to 200+ reports monthly
4. **Service Extensibility**: Core infrastructure that supports multiple service offerings
5. **Infrastructure as Code**: CloudFormation for complete system documentation and management

CloudFormation Implementation

CloudFormation provides a comprehensive "infrastructure as code" approach for managing all AWS resources. This ensures consistent deployment, simplified documentation, and easy sharing of the current system state between work sessions.

Benefits for the Project

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
5. **Multi-Service Support**: Facilitates adding new revenue streams

Current CloudFormation Template

```
yaml
```

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: 'TDIU Report Service Infrastructure'

Resources:
  # S3 Buckets
  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

  # Lambda Functions
  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'
              }

  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': 'This is a placeholder'
              }

  DocumentProcessorFunction:
```

```yaml
    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'
              }

  # 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
            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
```

AWS Services Implementation

1. Storage Layer (AWS S3)

Three secure buckets have been established:

1. **tdiu-document-storage**: For client-uploaded documents
    - Encryption: AES-256 at rest
    - Versioning: Enabled
    - Public Access: Blocked
2. **tdiu-templates**: For report templates
    - Encryption: AES-256 at rest
    - Versioning: Enabled
    - Public Access: Blocked
3. **tdiu-completed-reports**: For finalized reports
    - Encryption: AES-256 at rest
    - Versioning: Enabled
    - Public Access: Blocked

2. Processing Layer (AWS Lambda)

Three Lambda functions handle document processing:

1. **TDIU-GenerateUploadUrl**: Generates pre-signed URLs for secure client uploads
    - Runtime: Python 3.9
    - Triggers: Client request via API Gateway
    - Permissions: S3 access for URL generation
2. **TDIU-CreateCase**: Manages case creation and organization
    - Runtime: Python 3.9
    - Triggers: Document upload events
    - Permissions: S3 access for document organization

3. **TDIU-DocumentProcessor**: Handles document analysis with Claude AI
    - Runtime: Python 3.9
    - Triggers: Document upload events
    - Permissions: S3 access and Bedrock for AI processing

3. Security Layer

1. **AWS CloudTrail**: Comprehensive audit logging for HIPAA compliance
    - Trail Name: TDIU-Compliance-Trail
    - Coverage: Multi-region with global service events
    - Log Validation: Enabled
2. **IAM Roles**: Properly scoped permissions
    - LambdaExecutionRole: Access to S3, Bedrock, and basic execution
3. **Encryption**: AES-256 encryption for all data at rest

4. Client Portal (Planned)

The client portal will be implemented using:

1. **AWS Amplify**: For secure frontend hosting
2. **Amazon Cognito**: For user authentication
3. **API Gateway**: For secure backend communication

Multi-Service Technical Implementation

The architecture is designed to support multiple services:

Phase 1: TDIU Report Service (Current)

- Core infrastructure described in CloudFormation template
- Document storage and processing flows
- Initial AI integration

Phase 2: Document Analysis Service

- Enhanced AI processing for document analysis
- Additional Lambda functions for organization
- Extended S3 structure for document libraries

Phase 3: VA Brief Templates

- Template storage in S3
- Template customization Lambda functions
- DynamoDB for template metadata

Phase 4: Medical Terms Translation

- Specialized AI prompts for medical translation
- Term processing Lambda functions
- Medical terminology database in DynamoDB

Deployment & Management Strategy

CloudFormation Management

1. **Import Existing Resources**: Resources already created manually have been documented in CloudFormation
2. **Deploy New Resources**: All new resources will be deployed through CloudFormation
3. **Track Changes**: Template updates will be version-controlled
4. **Service Expansion**: Each new service will extend the CloudFormation template

Session Management with CloudFormation

For each new work session:

1. **Export Current Template**: Copy the CloudFormation template showing current resources
2. **Use Template in Session Start**: Include template in the initial prompt to Claude
3. **Document Session Changes**: Update template with new resources at end of session

Current Status & Next Steps

Implemented Components

- AWS account with proper security measures
- S3 buckets with appropriate security configurations
- CloudTrail for compliance logging
- Initial Lambda functions
- Claude 3.7 Sonnet access via AWS Bedrock
- CloudFormation template documenting current resources

Next Implementation Priorities

1. Complete client portal for document uploads
2. Finalize Claude AI integration via AWS Bedrock
3. Implement document analysis workflow
4. Test end-to-end system with sample documents
5. Prepare for initial service launch

Session Prompt Template

For starting new work sessions, use this prompt template:

```
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, e.g., "set up the client portal for
document uploads" or "integrate Claude AI via AWS Bedrock"].

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