Building a Secure Code Analysis Application with AWS Bedrock and CloudFormation

Enterprise code analysis and understanding often requires secure, efficient solutions that can handle sensitive codebases while providing intelligent insights. In this blog post, we'll explore how to build and deploy a secure code analysis application using AWS Bedrock's Claude 3 Sonnet model. This solution enables enterprise users to analyze code stored in S3 buckets through a chat interface, while maintaining security and compliance requirements.

We've provided a one-click CloudFormation template to get you started quickly. Before deploying the template, here's what you need to do:

Prerequisites

1. **Region Selection**
   - Open AWS Console
   - Switch to the `us-east-1` (N. Virginia) region as Claude 3 Sonnet is currently available here
2. **Enable Bedrock Access**
   - Go to Amazon Bedrock in the AWS Console
   - Click on "Model access" in the left sidebar
   - Find "Anthropic Claude 3 Sonnet"
   - Select the checkbox and click "Manage model access"
   - Click "Enable" for the model
   - Wait for the model status to show as "Available"
3. **Deployment Parameters** You'll need:
   - Your AWS Account ID (12-digit number)
   - A unique name for your S3 bucket (must be globally unique)

Quick Start Deployment

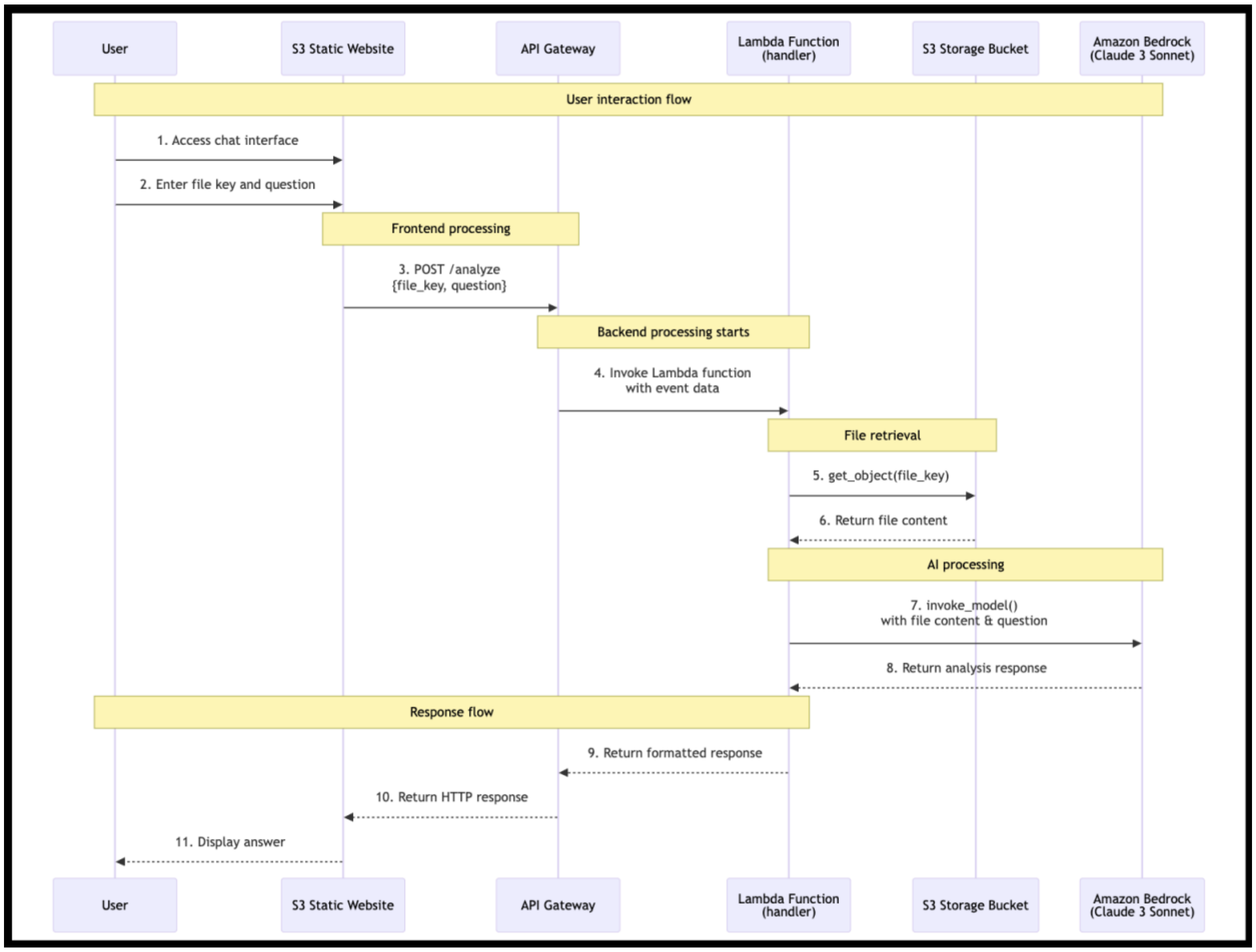
1. Download the provided CloudFormation template
2. Go to AWS CloudFormation console in `us-east-1` region
3. Click "Create stack" → "With new resources (standard)"
4. Upload the template file
5. Enter the required parameters:
   - AWS Account ID
   - S3 Bucket Name
6. Click through the next screens, acknowledge IAM resource creation
7. Click "Create stack"

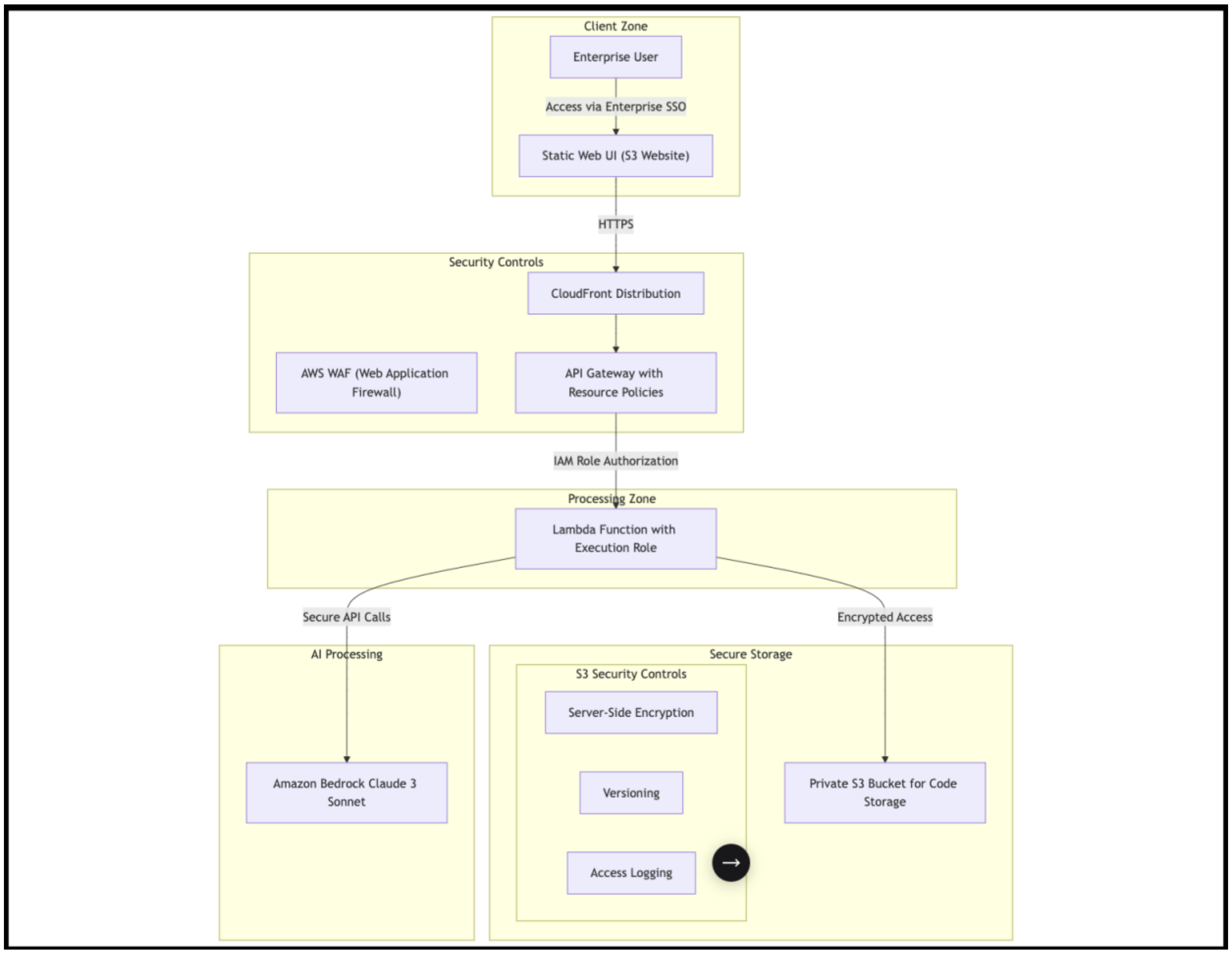
The deployment typically takes 3-5 minutes. Once complete, you'll see two URLs in the Outputs tab:

- API Endpoint URL
- Chat Interface URL

Architecture Overview

Let's start by understanding the high-level architecture of our solution:


User
S3 Static Website
API Gateway
Lambda Function (handler)
S3 Storage Bucket
Amazon Bedrock (Claude 3 Sonnet)
User interaction flow
1. Access chat interface
2. Enter file key and question
Frontend processing
3. POST /analyze {file_key, question}
Backend processing starts
4. Invoke Lambda function with event data
File retrieval
5. get_object(file_key)
6. Return file content
AI processing
7. invoke_model() with file content & question
8. Return analysis response
Response flow
9. Return formatted response
10. Return HTTP response
11. Display answer
User
S3 Static Website
API Gateway
Lambda Function (handler)
S3 Storage Bucket
Amazon Bedrock (Claude 3 Sonnet)



Key Components

1. User Interface

The application provides a simple yet effective chat interface hosted on S3. Users can specify the file they want to analyze and ask questions about the code.

Here's a snippet of the chat interface HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Code Analysis Chat</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    #chat-container { max-width: 800px; margin: 0 auto; }
    #messages { height: 400px; overflow-y: auto; border: 1px solid #ccc; padding: 10px; margin-bottom: 10px; }
    #input-container { display: flex; gap: 10px; }
```

```
        #question-input { flex-grow: 1; padding: 5px; }
    </style>
</head>
<body>
    <div id="chat-container">
        <h1>Code Analysis Chat</h1>
        <div id="file-input">
            <input type="text" id="file-key" placeholder="Enter S3 file key">
        </div>
        <div id="messages"></div>
        <div id="input-container">
            <input type="text" id="question-input" placeholder="Ask a question about the code...">
            <button onclick="sendQuestion()">Send</button>
        </div>
    </div>
    <script>
        // API integration code here
    </script>
</body>
</html>
```

2. Lambda Function for Code Analysis

The core of our application is a Lambda function that: - Retrieves code from S3 - Processes questions using Bedrock - Returns analyzed responses

Here's the key part of the Lambda function:

```
def handler(event, context):
    try:
        # Parse the incoming event body
        if isinstance(event.get('body'), str):
            body = json.loads(event['body'])
        else:
            body = event.get('body', {})

        file_key = body.get('file_key')
```

```
question = body.get('question')

# Initialize Bedrock client
bedrock = boto3.client(
    service_name='bedrock-runtime',
    region_name='us-east-1'  # Bedrock is available in specific regions
)

# Get file content from S3
file_content = ""
if file_key:
    s3 = boto3.client('s3')
    bucket = os.environ['BUCKET_NAME']
    response = s3.get_object(Bucket=bucket, Key=file_key)
    file_content = response['Body'].read().decode('utf-8')

# Prepare message for Claude
messages = [{
    "role": "user",
    "content": f"""Here is the code from the file:
    {file_content if file_content else 'No file content provided'}

    Question:
    {question}

    Please analyze the code and answer the question."""
}]

# Call Bedrock
response = bedrock.invoke_model(
    modelId='anthropic.claude-3-sonnet-20240229-v1:0',
    contentType='application/json',
    accept='application/json',
    body=json.dumps({
        "messages": messages,
        "max_tokens": 2048,
        "temperature": 0.7,
            "anthropic_version": "bedrock-2023-05-31"
        })
    )

    # Return response
    response_body = json.loads(response['body'].read().decode())
    assistant_message = response_body['content'][0]['text']

    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
            'message': assistant_message
        })
    }
except Exception as e:
    print(f"Error in Lambda: {str(e)}")
    return {
        'statusCode': 500,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
            'error': f"Error processing request: {str(e)}"
        })
    }
```

3. Security Features

API Gateway Resource Policies

We implement strict access controls using API Gateway resource policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "execute-api:Invoke",
      "Resource": "arn:aws:execute-api:region:account:*/*",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-xxx"
        }
      }
    },
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "execute-api:Invoke",
      "Resource": "arn:aws:execute-api:region:account:*/*",
      "Condition": {
        "NotIpAddress": {
          "aws:SourceIp": ["10.0.0.0/8"]
        }
      }
    }
  ]
}
```

IAM Roles and Policies

Careful attention to least-privilege access:

```
- PolicyName: BedrockAccess
  PolicyDocument:
    Version: '2012-10-17'
    Statement:
      - Effect: Allow
        Action:
```

```
    - "bedrock-runtime:InvokeModel"
  Resource: "arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-3-sonnet-20240229-v1:0"
```

Deployment Process

The entire infrastructure is deployed using CloudFormation, ensuring consistent and repeatable deployments. Here are some key considerations:

1. **Regional Availability**: Bedrock's Claude 3 Sonnet is currently available in specific regions (e.g., us-east-1). If deploying in other regions, configure the Lambda function to make cross-region calls.

2. **Model Access**: Before deployment, ensure you have enabled access to Claude 3 Sonnet in the Bedrock console.

3. **Security Setup**: Configure:

   - VPC settings
   - Allowed IP ranges
   - S3 bucket policies
   - API Gateway resource policies

Best Practices and Lessons Learned

1. **Security First**
   - Implement least-privilege access
   - Enable encryption at rest and in transit
   - Use VPC endpoints where possible
   - Implement proper logging and monitoring
2. **Error Handling**
   - Implement comprehensive error handling in Lambda
   - Add proper CORS headers
   - Handle file reading errors gracefully
3. **Performance**
   - Consider Lambda timeout settings
   - Optimize file reading for large codebases
   - Implement proper caching strategies
4. **Monitoring**
   - Set up CloudWatch alarms
   - Monitor API Gateway metrics
   - Track Bedrock usage and costs

Common Issues and Solutions

1. **Cross-Region Access**

```
bedrock = boto3.client(
  service_name='bedrock-runtime',
  region_name='us-east-1'  # Explicitly set region
)
```

2. **CORS Issues**

```
  'headers': {
  'Content-Type': 'application/json',
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Headers': 'Content-Type',
  'Access-Control-Allow-Methods': 'OPTIONS,POST'
}
```

3. **File Access Issues**

   - Implement proper error handling for S3 access
   - Handle different file encodings
   - Consider file size limits

Conclusion

This application demonstrates how to build a secure, enterprise-ready code analysis solution using AWS services. The combination of CloudFormation for infrastructure, Bedrock for AI capabilities, and various AWS security services creates a robust and scalable solution.

Key takeaways: - Security should be built-in from the start - Proper error handling is crucial - Consider regional availability of services - Use infrastructure as code for consistent deployments

Next Steps

Consider these enhancements:

1. Add authentication using Cognito
2. Implement caching for frequently accessed files
3. Add support for multiple file types
4. Implement rate limiting
5. Add custom WAF rules

Resources

- AWS Bedrock Documentation
- API Gateway Resource Policies
- CloudFormation Best Practices

GitHub Repo to download the one-click CloudFormation template

https://github.com/rineeshmr-org/code-analysis-tool.git