How to Use the Bedrock Claude/Titan API

This guide will walk you through using the API to interact with Anthropic Claude models and Amazon Titan Embedding models via AWS Bedrock.

API Overview

The API provides a unified interface to:

- Generate text with Claude 3.5 Sonnet or Claude 3 Haiku models
- Create text embeddings with Amazon Titan Embedding V2

Base URL: `https://quchnti6xu7yzw7hfzt5yjqtvi0kafsq.lambda-url.eu-central-1.on.aws/`

Authentication

All requests require an API key passed in the request body:

```
{
  "api_key": "syn-d4fc12c6-7d45-4241-830c-02d30d373c68"
}
```

Available Models

The API supports three models, specified via the `model_id` parameter:

- `claude-3.5-sonnet`: Claude's most capable model with strong reasoning abilities
- `claude-3-haiku`: Faster, more cost-effective Claude model
- `amazon-embedding-v2`: Amazon's text embedding model for creating vector representations

Making Requests

Basic Request Structure

All requests are POST requests to the base URL with a JSON body containing:

```
{
  "api_key": "YOUR_API_KEY",
  "prompt": "Your input text here",
  "model_id": "model-id-here",
  "model_params": {
    "max_tokens": 1024,
    "temperature": 0.7
  }
}
```

The `model_params` object is optional and primarily used for Claude models.

Example 1: Claude 3.5 Sonnet Text Generation

```python
import requests
import json

url = "https://quchnti6xu7yzw7hfzt5yjqtvi0kafsq.lambda-url.eu-central-1.on.aws/"

payload = {
    "api_key": "YOUR_API_KEY",
    "prompt": "Write a short story about a robot who discovers music.",
    "model_id": "claude-3.5-sonnet",
    "model_params": {
        "max_tokens": 500,
        "temperature": 0.7
    }
}

headers = {"Content-Type": "application/json"}

response = requests.post(url, headers=headers, data=json.dumps(payload))
result = response.json()

print(result["response"]["content"][0]["text"])
```

Example 2: Claude 3 Haiku for Quick Answers

```javascript
const fetch = require('node-fetch');

const url = 'https://quchnti6xu7yzw7hfzt5yjqtvi0kafsq.lambda-url.eu-central-1.on.aws/'

const payload = {
    api_key: 'YOUR_API_KEY',
    prompt: 'What is the capital of France?',
    model_id: 'claude-3-haiku',
    model_params: {
        max_tokens: 50,
        temperature: 0.2
    }
};

fetch(url, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload)
})
.then(response => response.json())
.then(data => {
    console.log(data.response.content[0].text);
})
.catch(error => console.error('Error:', error));
```

Example 3: Creating Text Embeddings

```
import requests
import json

url = "https://quchnti6xu7yzw7hfzt5yjqtvi0kafsq.lambda-url.eu-central-1.on.aws/"

payload = {
    "api_key": "YOUR_API_KEY",
    "prompt": "The quick brown fox jumps over the lazy dog.",
    "model_id": "amazon-embedding-v2"
}

headers = {"Content-Type": "application/json"}

response = requests.post(url, headers=headers, data=json.dumps(payload))
result = response.json()

# Access the embedding vector
embedding_vector = result["response"]["embedding"]
token_count = result["response"]["inputTextTokenCount"]

print(f"Embedding dimension: {len(embedding_vector)}")
print(f"Input token count: {token_count}")
```

cURL Example

```
curl -X POST \
  'https://quchnti6xu7yzw7hfzt5yjqtvi0kafsq.lambda-url.eu-central-1.on.aws/' \
  -H 'Content-Type: application/json' \
  -d '{
    "api_key": "YOUR_API_KEY",
    "prompt": "Explain quantum computing in simple terms.",
    "model_id": "claude-3.5-sonnet",
    "model_params": {
      "max_tokens": 300,
      "temperature": 0.5
    }
  }'
```

Understanding Responses

Claude Model Responses

Claude models return a response with this structure:

```
{
  "response": {
    "id": "msg_01Xg9PAA2Q8N7a56T4tW9e5x",
    "type": "message",
    "role": "assistant",
    "content": [
      {
        "type": "text",
        "text": "Quantum computing is a new kind of computing that..."
      }
    ],
    "model": "claude-3-5-sonnet-20240620",
    "stop_reason": "end_turn",
    "stop_sequence": null,
    "usage": {
      "input_tokens": 10,
      "output_tokens": 50
    }
  }
}
```

The actual generated text is in `response.content[0].text`.

Titan Embedding Responses

Embedding models return a vector representation:

```
{
  "response": {
    "embedding": [0.123, -0.456, 0.789, ...],
    "inputTextTokenCount": 8
  }
}
```

Error Handling

The API returns standard HTTP status codes:

- `200`: Success
- `400`: Bad request (missing/invalid parameters)
- `401`: Unauthorized (invalid API key)

Error responses have this format:

```
{
  "error": "Missing required parameter: prompt"
}
```

Best Practices

1. **Choose the right model**:
    - Use Claude 3.5 Sonnet for complex reasoning and creative tasks
    - Use Claude 3 Haiku for quick answers and lower-complexity tasks
    - Use Amazon Embedding V2 for vector representations
2. **Tune parameters for Claude models**:
    - Lower `temperature` (0.1-0.3) for more deterministic/factual responses
    - Higher `temperature` (0.7-0.9) for more creative responses
    - Set appropriate `max_tokens` based on expected response length
3. **Handle rate limits**:
    - Implement retry logic with exponential backoff
    - Consider batching requests when possible
4. **Secure your API key**:

   - Never expose your API key in client-side code
   - Use environment variables to store the key

5. **Validate responses**:

   - Check for expected fields before accessing them
   - Have error handling for unexpected response formats

This API provides a streamlined way to access powerful AI models through AWS Bedrock for text generation and embedding creation tasks.