Complete AWS Bedrock Production Application with Authentication

Step-by-Step Guide

Prerequisites

- AWS Account with appropriate permissions
- AWS CLI installed and configured
- Basic understanding of AWS services

PART 1: SETUP AMAZON BEDROCK

Step 1: Enable Amazon Bedrock Models

1. Go to AWS Console → **Amazon Bedrock**
2. Navigate to **Model access** in the left sidebar
3. Click **Manage model access**
4. Select models:
   - ✅ **Anthropic Claude 3.5 Sonnet** (recommended)
   - ✅ Any other models you want
5. Click **Request model access**
6. Wait for approval (usually instant)

PART 2: SETUP AUTHENTICATION (COGNITO)

Step 2: Create Cognito User Pool

1. Go to **Amazon Cognito** console
2. Click **Create user pool**

Configure sign-in experience:

- Sign-in options: ✅ **Email**

- Click **Next**

Configure security requirements:

- Password policy: **Cognito defaults**
- MFA: **No MFA** (can enable later)
- User account recovery: ✅ **Email only**
- Click **Next**

Configure sign-up experience:

- Self-service sign-up: ✅ **Enable self-registration**
- Attribute verification: ✅ **Send email message, verify email address**
- Required attributes:
  - ✅ **name**
  - ✅ **email** (auto-selected)
- Click **Next**

Configure message delivery:

- Email provider: **Send email with Cognito**
- Click **Next**

Integrate your app:

- User pool name: `BedrockAppUserPool`
- Hosted UI: ❌ **Don't use**
- App client name: `BedrockWebClient`
- Client secret: ❌ **Don't generate**
- Click **Next**

3. Review and click **Create user pool**

📝 **SAVE THESE VALUES:**

- **User Pool ID**: (e.g., `us-east-1_ABC123xyz`)
- **App Client ID**: (Go to App integration tab → App clients → Copy Client ID)

PART 3: SETUP IAM ROLES AND PERMISSIONS

Step 3: Create IAM Policy for Bedrock

1. Go to **IAM → Policies → Create Policy**
2. Choose **JSON** editor
3. Paste this policy:

```
json

{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "BedrockInvoke",
   "Effect": "Allow",
   "Action": [
    "bedrock:InvokeModel",
    "bedrock:InvokeModelWithResponseStream"
   ],
   "Resource": [
    "arn:aws:bedrock:*::foundation-model/*"
   ]
  }
 ]
}
```

4. Click **Next**
5. Name: `BedrockInvokePolicy`
6. Click **Create policy**

Step 4: Create IAM Policy for Cognito

1. **IAM → Policies → Create Policy**
2. Choose **JSON** editor
3. Paste this policy:

```
json
```

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "cognito-idp:GetUser"
   ],
   "Resource": "arn:aws:cognito-idp:*:*:userpool/*"
  }
 ]
}
```

4. Name: `CognitoGetUserPolicy`
5. Click **Create policy**

Step 5: Create Lambda Execution Role

1. Go to **IAM → Roles → Create Role**
2. Select **AWS Service → Lambda**
3. Click **Next**
4. Attach these policies:
   - ✅ `BedrockInvokePolicy` (created above)
   - ✅ `CognitoGetUserPolicy` (created above)
   - ✅ `AWSLambdaBasicExecutionRole` (AWS managed)
5. Click **Next**
6. Role name: `BedrockLambdaExecutionRole`
7. Click **Create role**

PART 4: CREATE LAMBDA FUNCTION

Step 6: Create Lambda Function

1. Go to **Lambda → Create function**
2. Choose **Author from scratch**
3. Function name: `BedrockContentGenerator`

4. Runtime: **Python 3.12**
5. Architecture: **x86_64**
6. Execution role: **Use an existing role** → Select `BedrockLambdaExecutionRole`
7. Click **Create function**

Step 7: Add Lambda Function Code

1. In the Lambda console, scroll to **Code source**
2. Replace all code with this:

```python
```

```
import json
import boto3
import os

bedrock_runtime = boto3.client(
    service_name='bedrock-runtime',
    region_name='us-east-1'  # Change to your region
)

cognito_client = boto3.client('cognito-idp', region_name='us-east-1')

def verify_token(access_token):
    """Verify Cognito access token and return user info"""
    try:
        response = cognito_client.get_user(AccessToken=access_token)

        # Extract user attributes
        user_attributes = {}
        for attr in response['UserAttributes']:
            user_attributes[attr['Name']] = attr['Value']

        return {
            'valid': True,
            'username': response['Username'],
            'email': user_attributes.get('email', ''),
            'name': user_attributes.get('name', ''),
            'user_id': user_attributes.get('sub', '')
        }
    except Exception as e:
        print(f"Token verification failed: {str(e)}")
        return {'valid': False, 'error': str(e)}

def lambda_handler(event, context):
    # Handle OPTIONS request for CORS
    if event.get('httpMethod') == 'OPTIONS':
        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type,Authorization',
                'Access-Control-Allow-Methods': 'POST,OPTIONS'
            },
            'body': ''
```

```
    }

try:
    # Extract authorization token
    headers = event.get('headers', {})
    auth_header = headers.get('Authorization') or headers.get('authorization')

    if not auth_header:
        return {
            'statusCode': 401,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({'error': 'No authorization token provided'})
        }

    # Remove 'Bearer ' prefix if present
    access_token = auth_header.replace('Bearer ', '')

    # Verify token
    user_info = verify_token(access_token)
    if not user_info['valid']:
        return {
            'statusCode': 401,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({'error': 'Invalid or expired token'})
        }

    # Parse request body
    body = json.loads(event['body']) if 'body' in event else event

    prompt = body.get('prompt', '')
    # Use cross-region inference profile instead of direct model ID
    model_id = body.get('model_id', 'us.anthropic.claude-3-5-sonnet-20241022-v2:0')
    max_tokens = body.get('max_tokens', 2048)
    temperature = body.get('temperature', 0.7)

    if not prompt:
        return {
            'statusCode': 400,
```

```
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({'error': 'Prompt is required'})
    }

# Prepare request for Claude
request_body = {
    "anthropic_version": "bedrock-2023-05-31",
    "max_tokens": max_tokens,
    "temperature": temperature,
    "messages": [
        {
            "role": "user",
            "content": prompt
        }
    ]
}

# Invoke Bedrock
response = bedrock_runtime.invoke_model(
    modelId=model_id,
    body=json.dumps(request_body)
)

# Parse response
response_body = json.loads(response['body'].read())
generated_text = response_body['content'][0]['text']

return {
    'statusCode': 200,
    'headers': {
        'Content-Type': 'application/json',
        'Access-Control-Allow-Origin': '*'
    },
    'body': json.dumps({
        'generated_text': generated_text,
        'model_used': model_id,
        'tokens_used': response_body.get('usage', {}),
        'user': {
            'email': user_info['email'],
            'name': user_info['name']
        }
```

```
        })
    }

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

3. Click **Deploy**

Step 8: Configure Lambda Settings

1. Go to **Configuration → General configuration → Edit**
2. Memory: **512 MB**
3. Timeout: **30 seconds**
4. Click **Save**

Step 9: Test Lambda Function (Optional)

1. Click **Test** tab
2. Create new test event with this JSON:

```json
{
  "httpMethod": "POST",
  "headers": {
    "Authorization": "Bearer test-token"
  },
  "body": "{\"prompt\": \"Write a short story about AI\", \"max_tokens\": 500}"
}
```

3. Click **Test** (it will fail authentication, which is expected)

PART 5: CREATE API GATEWAY

Step 10: Create REST API

1. Go to **API Gateway → Create API**
2. Choose **REST API → Build**
3. Choose **New API**
4. API name: `BedrockContentAPI`
5. Endpoint Type: **Regional**
6. Click **Create API**

Step 11: Create Resource and Method

1. Select **/ → Actions → Create Resource**
2. Resource Name: `generate`
3. Resource Path: `/generate`
4. Enable CORS: ✅ **Yes**
5. Click **Create Resource**
6. Select `/generate` → **Actions → Create Method → POST**
7. Setup:
    - Integration type: **Lambda Function**
    - Use Lambda Proxy integration: ✅ **Check this!**
    - Lambda Region: Select your region
    - Lambda Function: `BedrockContentGenerator`
8. Click **Save**
9. Click **OK** to grant permissions

Step 12: Enable CORS

1. Select `/generate` resource
2. **Actions → Enable CORS**
3. In the CORS configuration:
    - Access-Control-Allow-Headers: `Content-Type,Authorization`
    - Access-Control-Allow-Methods: Check **POST** and **OPTIONS**
4. Click **Enable CORS and replace existing CORS headers**

5. Confirm

Step 13: Deploy API

1. **Actions** → **Deploy API**
2. Deployment stage: **[New Stage]**
3. Stage name: `prod`
4. Click **Deploy**

📝 **SAVE THIS VALUE:**

- **Invoke URL**: Copy the URL (e.g., `https://abc123.execute-api.us-east-1.amazonaws.com/prod`)

PART 6: CREATE WEB FRONTEND

Step 14: Create S3 Bucket

1. Go to **S3** → **Create bucket**
2. Bucket name: `bedrock-app-web-YOUR-NAME` (must be globally unique)
3. Region: **Same as your API**
4. ❌ **Uncheck** "Block all public access"
5. ✅ Acknowledge the warning
6. Click **Create bucket**

Step 15: Enable Static Website Hosting

1. Select your bucket → **Properties** tab
2. Scroll to **Static website hosting** → **Edit**
3. Static website hosting: **Enable**
4. Index document: `index.html`
5. Click **Save changes**

📝 **SAVE THIS VALUE:**

- Note the **Bucket website endpoint** URL

Step 16: Add Bucket Policy

1. Go to **Permissions** tab → **Bucket policy** → **Edit**
2. Replace `YOUR-BUCKET-NAME` and paste:

```json
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "PublicReadGetObject",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "s3:GetObject",
   "Resource": "arn:aws:s3:::YOUR-BUCKET-NAME/*"
  }
 ]
}
```

3. Click **Save changes**

Step 17: Create index.html File

Create a file named `index.html` on your computer with this content:

```html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AI Content Generator - AWS Bedrock</title>
    <script src="https://cdn.jsdelivr.net/npm/amazon-cognito-identity-js@6.3.6/dist/amazon-cognito-identity.min.js"></script>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            min-height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
            padding: 20px;
        }

        .container {
            background: white;
            border-radius: 20px;
            box-shadow: 0 20px 60px rgba(0,0,0,0.3);
            padding: 40px;
            max-width: 800px;
            width: 100%;
        }

        .auth-container {
            max-width: 450px;
        }

        h1 {
            color: #333;
            margin-bottom: 10px;
            font-size: 2.5em;
        }
```

```
h2 {
    color: #333;
    margin-bottom: 20px;
    font-size: 1.8em;
}

.subtitle {
    color: #666;
    margin-bottom: 30px;
    font-size: 1.1em;
}

.form-group {
    margin-bottom: 20px;
}

label {
    display: block;
    margin-bottom: 8px;
    color: #333;
    font-weight: 600;
}

input[type="text"],
input[type="email"],
input[type="password"],
textarea {
    width: 100%;
    padding: 12px 15px;
    border: 2px solid #e0e0e0;
    border-radius: 10px;
    font-size: 16px;
    font-family: inherit;
    transition: border-color 0.3s;
}

input:focus,
textarea:focus {
    outline: none;
    border-color: #667eea;
}

textarea {
    resize: vertical;
```

```
    min-height: 120px;
}

button {
    width: 100%;
    padding: 15px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    border: none;
    border-radius: 10px;
    font-size: 18px;
    font-weight: 600;
    cursor: pointer;
    transition: transform 0.2s, box-shadow 0.2s;
    margin-bottom: 10px;
}

button:hover:not(:disabled) {
    transform: translateY(-2px);
    box-shadow: 0 10px 20px rgba(102, 126, 234, 0.4);
}

button:disabled {
    opacity: 0.6;
    cursor: not-allowed;
}

.secondary-btn {
    background: white;
    color: #667eea;
    border: 2px solid #667eea;
}

.secondary-btn:hover:not(:disabled) {
    background: #f8f9ff;
}

.link-btn {
    background: none;
    color: #667eea;
    text-decoration: underline;
    padding: 10px;
    font-size: 16px;
}
```

```
.link-btn:hover {
    transform: none;
    box-shadow: none;
}

.hidden {
    display: none;
}

.error-message {
    background: #fee;
    color: #c33;
    padding: 12px;
    border-radius: 8px;
    margin-bottom: 20px;
    border-left: 4px solid #f44;
}

.success-message {
    background: #efe;
    color: #3c3;
    padding: 12px;
    border-radius: 8px;
    margin-bottom: 20px;
    border-left: 4px solid #4f4;
}

.user-info {
    background: #f8f9fa;
    padding: 15px;
    border-radius: 10px;
    margin-bottom: 20px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.user-details {
    color: #555;
}

.settings {
    display: grid;
```

```
    grid-template-columns: 1fr 1fr;
    gap: 20px;
    margin-bottom: 20px;
}

.output {
    margin-top: 30px;
    padding: 20px;
    background: #f8f9fa;
    border-radius: 10px;
    border-left: 4px solid #667eea;
    display: none;
}

.output.show {
    display: block;
}

.output-text {
    color: #555;
    line-height: 1.8;
    white-space: pre-wrap;
}

.loading {
    display: none;
    text-align: center;
    margin-top: 20px;
}

.loading.show {
    display: block;
}

.spinner {
    border: 4px solid #f3f3f3;
    border-top: 4px solid #667eea;
    border-radius: 50%;
    width: 40px;
    height: 40px;
    animation: spin 1s linear infinite;
    margin: 0 auto;
}
```

```
        @keyframes spin {
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }

        .verification-notice {
            background: #fff3cd;
            color: #856404;
            padding: 12px;
            border-radius: 8px;
            margin-bottom: 20px;
            border-left: 4px solid #ffc107;
        }

        .meta-info {
            margin-top: 15px;
            padding-top: 15px;
            border-top: 1px solid #ddd;
            font-size: 0.9em;
            color: #666;
        }

        @media (max-width: 768px) {
            .settings {
                grid-template-columns: 1fr;
            }
            h1 {
                font-size: 2em;
            }
        }
    </style>
</head>
<body>
    <!-- Login/Signup Forms -->
    <div id="authContainer" class="container auth-container">
        <!-- Login Form -->
        <div id="loginForm">
            <h2>🔐 Sign In</h2>
            <p class="subtitle">Welcome back!</p>

            <div id="loginError" class="error-message hidden"></div>

            <div class="form-group">
                <label for="loginEmail">Email</label>
```

```
        <input type="email" id="loginEmail" placeholder="your.email@example.com" required>
    </div>

    <div class="form-group">
        <label for="loginPassword">Password</label>
        <input type="password" id="loginPassword" placeholder="Enter your password" required>
    </div>

    <button onclick="login()">Sign In</button>
    <button class="link-btn" onclick="showSignup()">Don't have an account? Sign up</button>
</div>

<!-- Signup Form -->
<div id="signupForm" class="hidden">
    <h2> Create Account</h2>
    <p class="subtitle">Join us today!</p>

    <div id="signupError" class="error-message hidden"></div>
    <div id="signupSuccess" class="success-message hidden"></div>

    <div class="form-group">
        <label for="signupName">Full Name</label>
        <input type="text" id="signupName" placeholder="John Doe" required>
    </div>

    <div class="form-group">
        <label for="signupEmail">Email</label>
        <input type="email" id="signupEmail" placeholder="your.email@example.com" required>
    </div>

    <div class="form-group">
        <label for="signupPassword">Password</label>
        <input type="password" id="signupPassword" placeholder="Min 8 characters" required>
    </div>

    <button onclick="signup()">Create Account</button>
    <button class="link-btn" onclick="showLogin()">Already have an account? Sign in</button>
</div>

<!-- Verification Form -->
<div id="verificationForm" class="hidden">
    <h2> Verify Email</h2>
    <p class="subtitle">Check your email for the verification code</p>
```

```
        <div id="verifyError" class="error-message hidden"></div>

        <div class="verification-notice">
            We've sent a verification code to <strong id="verifyEmail"></strong>
        </div>

        <div class="form-group">
            <label for="verificationCode">Verification Code</label>
            <input type="text" id="verificationCode" placeholder="Enter 6-digit code" required>
        </div>

        <button onclick="verifyEmail()">Verify Email</button>
        <button class="secondary-btn" onclick="resendCode()">Resend Code</button>
    </div>
</div>

<!-- Main App (hidden until logged in) -->
<div id="appContainer" class="container hidden">
    <div class="user-info">
        <div class="user-details">
            <strong>👤 <span id="userName"></span></strong><br>
            <small id="userEmail"></small>
        </div>
        <button class="secondary-btn" onclick="logout()" style="width: auto; padding: 10px 20px;">Logout</button>
    </div>

    <h1>🤖 AI Content Generator</h1>
    <p class="subtitle">Powered by AWS Bedrock & Claude</p>

    <div class="form-group">
        <label for="prompt">Enter your prompt:</label>
        <textarea
            id="prompt"
            placeholder="Example: Write a blog post about cloud computing..."
        ></textarea>
    </div>

    <div class="settings">
        <div class="form-group">
            <label for="maxTokens">Max Tokens:</label>
            <input type="number" id="maxTokens" value="2048" min="100" max="4096" step="100">
        </div>
        <div class="form-group">
            <label for="temperature">Temperature (0-1):</label>
```

```
            <input type="number" id="temperature" value="0.7" min="0" max="1" step="0.1">
        </div>
    </div>

    <button id="generateBtn" onclick="generateContent()">Generate Content</button>

    <div class="loading" id="loading">
        <div class="spinner"></div>
        <p>Generating content...</p>
    </div>

    <div class="output" id="output">
        <h3>Generated Content:</h3>
        <div class="output-text" id="outputText"></div>
        <div class="meta-info" id="metaInfo"></div>
    </div>
</div>

<script>
    // ============== CONFIGURATION ==============
    // ⚠️ REPLACE THESE WITH YOUR VALUES ⚠️
    const COGNITO_CONFIG = {
        UserPoolId: 'us-east-1_XXXXXXXXX',  // From Step 2
        ClientId: 'XXXXXXXXXXXXXXXXXXXXXXXXX'  // From Step 2
    };

    const API_ENDPOINT = 'https://YOUR-API-ID.execute-api.us-east-1.amazonaws.com/prod/generate';  // From Step 13
    // ============================================

    const poolData = {
        UserPoolId: COGNITO_CONFIG.UserPoolId,
        ClientId: COGNITO_CONFIG.ClientId
    };

    const userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
    let currentUser = null;
    let signupEmail = '';

    // Check if user is already logged in
    window.onload = function() {
        currentUser = userPool.getCurrentUser();
        if (currentUser) {
            currentUser.getSession((err, session) => {
                if (err || !session.isValid()) {
```

```
            showAuthContainer();
        } else {
            getUserAttributes();
        }
    });
  }
};

function showLogin() {
  document.getElementById('loginForm').classList.remove('hidden');
  document.getElementById('signupForm').classList.add('hidden');
  document.getElementById('verificationForm').classList.add('hidden');
  clearMessages();
}

function showSignup() {
  document.getElementById('loginForm').classList.add('hidden');
  document.getElementById('signupForm').classList.remove('hidden');
  document.getElementById('verificationForm').classList.add('hidden');
  clearMessages();
}

function showVerification(email) {
  signupEmail = email;
  document.getElementById('verifyEmail').textContent = email;
  document.getElementById('loginForm').classList.add('hidden');
  document.getElementById('signupForm').classList.add('hidden');
  document.getElementById('verificationForm').classList.remove('hidden');
  clearMessages();
}

function showAuthContainer() {
  document.getElementById('authContainer').classList.remove('hidden');
  document.getElementById('appContainer').classList.add('hidden');
}

function showAppContainer() {
  document.getElementById('authContainer').classList.add('hidden');
  document.getElementById('appContainer').classList.remove('hidden');
}

function clearMessages() {
  document.querySelectorAll('.error-message, .success-message').forEach(el => {
    el.classList.add('hidden');
```

```
        el.textContent = '';
    });
}

function showError(elementId, message) {
    const el = document.getElementById(elementId);
    el.textContent = message;
    el.classList.remove('hidden');
}

function showSuccess(elementId, message) {
    const el = document.getElementById(elementId);
    el.textContent = message;
    el.classList.remove('hidden');
}

async function signup() {
    clearMessages();

    const name = document.getElementById('signupName').value.trim();
    const email = document.getElementById('signupEmail').value.trim();
    const password = document.getElementById('signupPassword').value;

    if (!name || !email || !password) {
        showError('signupError', 'Please fill in all fields');
        return;
    }

    const attributeList = [
        new AmazonCognitoIdentity.CognitoUserAttribute({
            Name: 'email',
            Value: email
        }),
        new AmazonCognitoIdentity.CognitoUserAttribute({
            Name: 'name',
            Value: name
        })
    ];

    userPool.signUp(email, password, attributeList, null, (err, result) => {
        if (err) {
            showError('signupError', err.message);
            return;
        }
```

```
            showSuccess('signupSuccess', 'Account created! Please check your email for verification code.');
            setTimeout(() => showVerification(email), 2000);
        });
    }

    function verifyEmail() {
        clearMessages();

        const code = document.getElementById('verificationCode').value.trim();

        if (!code) {
            showError('verifyError', 'Please enter the verification code');
            return;
        }

        const userData = {
            Username: signupEmail,
            Pool: userPool
        };

        const cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);

        cognitoUser.confirmRegistration(code, true, (err, result) => {
            if (err) {
                showError('verifyError', err.message);
                return;
            }

            alert('Email verified successfully! Please sign in.');
            showLogin();
        });
    }

    function resendCode() {
        const userData = {
            Username: signupEmail,
            Pool: userPool
        };

        const cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);

        cognitoUser.resendConfirmationCode((err, result) => {
            if (err) {
```

```
            showError('verifyError', err.message);
            return;
        }
        alert('Verification code resent! Check your email.');
    });
}

function login() {
    clearMessages();

    const email = document.getElementById('loginEmail').value.trim();
    const password = document.getElementById('loginPassword').value;

    if (!email || !password) {
        showError('loginError', 'Please enter email and password');
        return;
    }

    const authenticationData = {
        Username: email,
        Password: password
    };

    const authenticationDetails = new AmazonCognitoIdentity.AuthenticationDetails(authenticationData);

    const userData = {
        Username: email,
        Pool: userPool
    };

    const cognitoUser = new AmazonCognitoIdentity.CognitoUser(userData);

    cognitoUser.authenticateUser(authenticationDetails, {
        onSuccess: (result) => {
            currentUser = cognitoUser;
            getUserAttributes();
        },
        onFailure: (err) => {
            showError('loginError', err.message);
        }
    });
}

function getUserAttributes() {
```

```
    currentUser.getSession((err, session) => {
        if (err) {
            showAuthContainer();
            return;
        }

        currentUser.getUserAttributes((err, attributes) => {
            if (err) {
                console.error(err);
                return;
            }

            const userAttrs = {};
            attributes.forEach(attr => {
                userAttrs[attr.Name] = attr.Value;
            });

            document.getElementById('userName').textContent = userAttrs.name || 'User';
            document.getElementById('userEmail').textContent = userAttrs.email || '';

            showAppContainer();
        });
    });
}

function logout() {
    if (currentUser) {
        currentUser.signOut();
        currentUser = null;
    }
    showAuthContainer();
    showLogin();
}

async function generateContent() {
    const prompt = document.getElementById('prompt').value.trim();
    const maxTokens = parseInt(document.getElementById('maxTokens').value);
    const temperature = parseFloat(document.getElementById('temperature').value);

    const outputDiv = document.getElementById('output');
    const loadingDiv = document.getElementById('loading');
    const generateBtn = document.getElementById('generateBtn');

    if (!prompt) {
```

```
            alert('Please enter a prompt');
            return;
        }

        // Get access token
        currentUser.getSession((err, session) => {
            if (err || !session.isValid()) {
                alert('Session expired. Please login again.');
                logout();
                return;
            }

            const accessToken = session.getAccessToken().getJwtToken();

            // Show loading
            loadingDiv.classList.add('show');
            outputDiv.classList.remove('show');
            generateBtn.disabled = true;

            fetch(API_ENDPOINT, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${accessToken}`
                },
                body: JSON.stringify({
                    prompt: prompt,
                    max_tokens: maxTokens,
                    temperature: temperature
                })
            })
            .then(response => response.json())
            .then(data => {
                if (data.error) {
                    throw new Error(data.error);
                }

                document.getElementById('outputText').textContent = data.generated_text;

                // Display metadata
                const metaInfo = `
Model: ${data.model_used}
User: ${data.user.name} (${data.user.email})
Input Tokens: ${data.tokens_used.input_tokens || 'N/A'}
```

```
Output Tokens: ${data.tokens_used.output_tokens || 'N/A'}
                `.trim();
                document.getElementById('metaInfo').textContent = metaInfo;

                outputDiv.classList.add('show');
            })
            .catch(error => {
                console.error('Error:', error);
                alert(`Error: ${error.message}`);
            })
            .finally(() => {
                loadingDiv.classList.remove('show');
                generateBtn.disabled = false;
            });
        });
    }

    // Allow Shift+Enter to submit
    document.addEventListener('DOMContentLoaded', function() {
        const promptField = document.getElementById('prompt');
        if (promptField) {
            promptField.addEventListener('keydown', function(e) {
                if (e.key === 'Enter' && e.shiftKey) {
                    e.preventDefault();
                    generateContent();
                }
            });
        }
    });
  </script>
</body>
</html>
```

⚠️ **IMPORTANT - YOU MUST UPDATE 3 VALUES:**

1. Line 224: `UserPoolId: 'us-east-1_XXXXXXXXX'` - Replace with your User Pool ID from Step 2
2. Line 225: `ClientId: 'XXXXXXXXXXXXXXXXXXXXXXXXXX'` - Replace with your App Client ID from Step 2
3. Line 228: `API_ENDPOINT` - Replace with your API Gateway URL from Step 13

Step 18: Upload to S3

1. Go to your S3 bucket
2. Click **Upload**
3. Click **Add files**
4. Select the `index.html` file you just created
5. Click **Upload**

PART 7: TEST YOUR APPLICATION

Step 19: Test the Complete Application

1. Go to your S3 bucket → **Properties** tab
2. Scroll to **Static website hosting**
3. Click the **Bucket website endpoint** URL

Test Signup Flow:

1. Click "Don't have an account? Sign up"
2. Enter your name, email, and password (min 8 characters)
3. Click "Create Account"
4. Check your email for verification code
5. Enter the code and click "Verify Email"

Test Login Flow:

1. Enter your email and password
2. Click "Sign In"
3. You should see the main content generator interface

Test Content Generation:

1. Enter a prompt like "Write a short poem about clouds"
2. Adjust max tokens and temperature if needed
3. Click "Generate Content"
4. Wait for the AI-generated response

PART 8: OPTIONAL - ADD CLOUDFRONT FOR HTTPS

Step 20: Create CloudFront Distribution (Optional)

1. Go to **CloudFront → Create Distribution**
2. **Origin domain**: Select your S3 bucket from dropdown
3. **Origin access**: Select **Origin access control settings (recommended)**
4. Click **Create control setting → Create**
5. **Viewer protocol policy**: **Redirect HTTP to HTTPS**
6. **Default root object**: `index.html`
7. Click **Create distribution**

Step 21: Update S3 Bucket Policy for CloudFront

CloudFront will show you a policy to add. Replace your current S3 bucket policy with the one CloudFront provides. It will look like this:

```json
{
 "Version": "2012-10-17",
 "Statement": {
  "Sid": "AllowCloudFrontServicePrincipal",
  "Effect": "Allow",
  "Principal": {
   "Service": "cloudfront.amazonaws.com"
  },
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::YOUR-BUCKET-NAME/*",
  "Condition": {
   "StringEquals": {
    "AWS:SourceArn": "arn:aws:cloudfront::YOUR-ACCOUNT:distribution/YOUR-DISTRIBUTION-ID"
   }
  }
 }
}
```

Note: Wait 5-10 minutes for CloudFront to deploy, then access your app via the CloudFront domain name (https://xxx.cloudfront.net)

TROUBLESHOOTING GUIDE

Issue: "Model not found" error

Solution:

- Verify Bedrock model access is granted in Step 1
- Check model ID in Lambda code matches enabled model
- Ensure correct AWS region

Issue: CORS errors in browser console

Solution:

- Verify CORS is enabled on API Gateway (Step 12)
- Check Lambda returns CORS headers
- Ensure OPTIONS method exists in API Gateway

Issue: "Invalid or expired token"

Solution:

- User session expired, logout and login again
- Check Cognito User Pool ID and Client ID are correct
- Verify Lambda has Cognito permissions

Issue: "User is not confirmed"

Solution:

- User must verify email first
- Check spam folder for verification code
- Use "Resend Code" button

Issue: Lambda timeout

Solution:

- Increase Lambda timeout to 30 seconds (Step 8)
- Check Bedrock model is responding

- Review CloudWatch logs

Issue: Signup email not received

Solution:

- Check spam/junk folder
- Verify email is valid
- Use Cognito console to manually confirm user

COST ESTIMATION

Monthly costs for moderate usage (10,000 requests):

Service	Usage	Cost
Bedrock	10K requests × 500 tokens avg	$3-15
Lambda	10K invocations	$0.20
API Gateway	10K requests	$0.035
Cognito	Up to 50K users	FREE
S3	1 GB storage + transfer	$0.50
CloudFront (optional)	1 GB transfer	$0.085
Total		**~$4-16/month**

Free Tier Benefits:

- Lambda: 1M requests/month free
- API Gateway: 1M requests/month free (12 months)
- Cognito: 50K MAU free forever
- S3: 5GB storage free (12 months)

SECURITY CHECKLIST

✅ Completed Automatically:

- [x] Authentication with Cognito
- [x] Token-based API authorization
- [x] Email verification required
- [x] CORS properly configured
- [x] IAM least privilege roles
- [x] HTTPS via CloudFront (if completed Step 20)

✅ Recommended Next Steps:

- [ ] Enable MFA in Cognito for sensitive apps
- [ ] Add AWS WAF to API Gateway
- [ ] Enable CloudTrail for audit logging
- [ ] Set up CloudWatch alarms for errors
- [ ] Implement rate limiting per user
- [ ] Enable S3 versioning for backup
- [ ] Use Secrets Manager for sensitive config

MONITORING & LOGS

View Lambda Logs:

1. Lambda console → Monitor tab → View CloudWatch logs
2. Look for errors or performance issues

View API Gateway Logs:

1. API Gateway → Stages → prod → Logs/Tracing
2. Enable execution logging and access logging

Monitor Bedrock Usage:

1. Bedrock console → Usage
2. Track token consumption and costs

Set Up CloudWatch Alarms:

1. CloudWatch → Alarms → Create alarm
2. Monitor Lambda errors, duration, throttles
3. Set SNS notifications for alerts

NEXT STEPS & ENHANCEMENTS

Level 1 - Easy Enhancements:

1. **Custom Domain**: Use Route53 + ACM certificate
2. **Error Handling**: Better error messages in UI
3. **Loading States**: Improved UX during generation
4. **Copy Button**: Add copy-to-clipboard for generated text

Level 2 - Medium Enhancements:

1. **DynamoDB History**: Store user's generation history
2. **Multiple Models**: Let users choose different AI models
3. **Streaming**: Implement real-time token streaming
4. **Usage Limits**: Rate limiting per user
5. **Password Reset**: Forgot password functionality

Level 3 - Advanced Enhancements:

1. **RAG System**: Add Knowledge Bases for document Q&A
2. **Image Generation**: Integrate Stable Diffusion models
3. **Conversation History**: Multi-turn conversations
4. **Admin Dashboard**: User management and analytics
5. **API Keys**: Allow programmatic access
6. **Webhooks**: Integration with other services

USEFUL AWS CONSOLE LINKS

- **Bedrock**: https://console.aws.amazon.com/bedrock/

- **Cognito**: https://console.aws.amazon.com/cognito/
- **Lambda**: https://console.aws.amazon.com/lambda/
- **API Gateway**: https://console.aws.amazon.com/apigateway/
- **S3**: https://console.aws.amazon.com/s3/
- **CloudFront**: https://console.aws.amazon.com/cloudfront/
- **IAM**: https://console.aws.amazon.com/iam/
- **CloudWatch**: https://console.aws.amazon.com/cloudwatch/

SUPPORT & RESOURCES

- AWS Bedrock Documentation
- Cognito Developer Guide
- Lambda Developer Guide
- API Gateway Documentation
- AWS Support

FINAL CHECKLIST

Before going live, verify:

- [ ] Bedrock models are enabled and accessible
- [ ] Cognito User Pool is created and configured
- [ ] Lambda function is deployed and tested
- [ ] API Gateway is deployed to 'prod' stage
- [ ] S3 bucket is public and website hosting enabled
- [ ] HTML file has correct User Pool ID, Client ID, and API URL
- [ ] Can signup, verify email, and login successfully
- [ ] Can generate content after authentication
- [ ] CloudWatch logs are working
- [ ] Costs are being monitored

🎉 **Congratulations! Your production-ready AWS Bedrock application with authentication is now complete!**