Buddy Week 1: AWS Bedrock Setup & Infrastructure Bootstrap

Days 1-7 Implementation Guide

AWS Account: 052080186586
IAM User: lopezdev
Target Region: us-east-1
Timeline: February 6-12, 2026

Day 1-2: AWS Bedrock Access & Initial Setup

Step 1: Enable Amazon Bedrock Model Access

Action: Request access to Nova 2 models in your AWS account.

Instructions:

1. **Log into AWS Console**
    - Account ID: 052080186586
    - User: lopezdev
    - Navigate to: https://console.aws.amazon.com/bedrock
2. **Request Model Access**
   Bedrock Console → Left sidebar → "Model access" → "Manage model access"
3. **Select Required Models**
    - ☑ Amazon Nova 2 Sonic (voice interface)
    - ☑ Amazon Nova 2 Lite (agentic reasoning)
    - Click "Request model access"
4. **Verify Access Status**
    - Wait 5-30 minutes for approval
    - Status should change to "Access granted" (green checkmark)
    - If denied, check AWS account type (requires standard account, not sandbox)

Verification Command:

```
aws bedrock list-foundation-models
--region us-east-1
--query 'modelSummaries[?contains(modelId, nova)].[modelId, modelName]'
--output table
```

Expected Output:

```
| ListFoundationModels |
+------------------------------------------+
| amazon.nova-2-sonic-v1:0 | Nova 2 Sonic |
| amazon.nova-2-lite-v1:0 | Nova 2 Lite |
+------------------------------------------+
```

Step 2: Configure AWS CLI Credentials

Ensure lopezdev user has programmatic access:

Configure AWS CLI with your credentials

```
aws configure --profile buddy-dev
```

Enter when prompted:

```
AWS Access Key ID: [Your Access Key]
AWS Secret Access Key: [Your Secret Key]
Default region name: us-east-1
Default output format: json
```

Test configuration

```
aws sts get-caller-identity --profile buddy-dev
```

Expected Response:

```
{
"UserId": "AIDA...",
"Account": "052080186586",
"Arn": "arn:aws:iam::052080186586:user/lopezdev"
}
```

Step 3: Create IAM Role for Lambda Execution

File: `iam-lambda-role.json`

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Principal": {
"Service": "lambda.amazonaws.com"
},
"Action": "sts:AssumeRole"
}
]
}
```

Commands:

Create Lambda execution role

```
aws iam create-role
--role-name BuddyLambdaExecutionRole
--assume-role-policy-document file://iam-lambda-role.json
--profile buddy-dev
```

Attach managed policies

```
aws iam attach-role-policy
--role-name BuddyLambdaExecutionRole
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
--profile buddy-dev
```

```
aws iam attach-role-policy
--role-name BuddyLambdaExecutionRole
--policy-arn arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess
--profile buddy-dev
```

```
aws iam attach-role-policy
--role-name BuddyLambdaExecutionRole
--policy-arn arn:aws:iam::aws:policy/AmazonSNSFullAccess
--profile buddy-dev
```

Create custom Bedrock policy:

File: `bedrock-policy.json`

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"bedrock:InvokeModel",
"bedrock:InvokeModelWithResponseStream"
],
"Resource": [
"arn:aws:bedrock:us-east-1::foundation-model/amazon.nova-sonic-v2:0",
"arn:aws:bedrock:us-east-1::foundation-model/amazon.nova-lite-v2:0"
]
}
]
}
```

```
aws iam put-role-policy
--role-name BuddyLambdaExecutionRole
--policy-name BedrockInvokePolicy
--policy-document file://bedrock-policy.json
--profile buddy-dev
```

Day 3-4: DynamoDB Schema Setup

Create Patient Profiles Table

```
aws dynamodb create-table
--table-name BuddyPatientProfiles
--attribute-definitions
AttributeName=patientId,AttributeType=S
--key-schema
AttributeName=patientId,KeyType=HASH
--billing-mode PAY_PER_REQUEST
--region us-east-1
--profile buddy-dev
```

Seed Test Data

File: test-patient-john.json

```
{
"patientId": {"S": "test-patient-001"},
"profile": {
"M": {
"name": {"S": "John Doe"},
"preferredName": {"S": "John"},
"birthdate": {"S": "1945-03-15"},
"dementiaStage": {"S": "moderate"}
}
},
"people": {
"L": [
{
"M": {
"name": {"S": "Sarah"},
"relationship": {"S": "daughter"},
"visitSchedule": {"S": "Tuesdays"},
"sharedActivities": {
"L": [{"S": "gardening"}]
}
}
}
]
},
"routines": {
"L": [
{
"M": {
"timeOfDay": {"S": "morning"},
"steps": {
"L": [
{"S": "brush teeth"},
{"S": "take medication"},
{"S": "eat breakfast"}
]
},
"context": {"S": "Toothbrush is in the blue cup by the sink"}
}
}
]
},
"medications": {
"L": [
{
"M": {
"name": {"S": "Donepezil"},
"dosage": {"S": "10mg"},
"timing": {"S": "after breakfast"},
"appearance": {"S": "small white round pill"}
}
}
]
},
"safetyProfile": {
"M": {
"caregiverPhone": {"S": "+15551234567"},
"emergencyContacts": {
"L": [{"S": "daughter"}, {"S": "neighbor"}]
},
"medicalConditions": {
"L": [{"S": "Alzheimer's"}, {"S": "hypertension"}]
},
"allergies": {
"L": [{"S": "penicillin"}]
},
"knownTriggers": {
"L": [{"S": "nighttime confusion"}, {"S": "crowds"}]
}
}
},
"conversationHistory": {
"L": []
}
}
```

Insert test patient:

```
aws dynamodb put-item
--table-name BuddyPatientProfiles
--item file://test-patient-john.json
--profile buddy-dev
```

Verify insertion:

```
aws dynamodb get-item
--table-name BuddyPatientProfiles
```

--key '{"patientId": {"S": "test-patient-001"}}'
--profile buddy-dev

Day 5-6: Lambda Function Skeleton

Create Lambda Deployment Package

Directory structure:
buddy-lambda/
├── lambda_function.py
├── requirements.txt
└── utils/
├── **init**.py
├── dynamodb_helper.py
└── bedrock_helper.py

File: `lambda_function.py`

import json
import os
import boto3
from datetime import datetime
from utils.dynamodb_helper import get_patient_profile, log_conversation
from utils.bedrock_helper import invoke_nova_lite

Initialize AWS clients

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

def lambda_handler(event, context):
"""
Main handler for Buddy Alexa Skill
"""
print(f"Received event: {json.dumps(event)}")

```
# Extract patient ID from session (will come from Alexa later)
patient_id = event.get('session', {}).get('user', {}).get('userId', 'test-patient-001')

# Parse user utterance
utterance = event.get('request', {}).get('intent', {}).get('slots', {}).get('query', {}).get

if not utterance:
    return build_response("I didn't understand. Can you say that again?")

# Retrieve patient profile from DynamoDB
```

```
    patient_profile = get_patient_profile(patient_id)

    if not patient_profile:
        return build_response("I'm sorry, I couldn't find your profile. Please contact 

    # Route intent (simplified for Week 1)
    if "what do i do" in utterance.lower() or "routine" in utterance.lower():
        response_text = handle_routine_query(patient_profile)
    elif "who is" in utterance.lower():
        response_text = handle_people_query(utterance, patient_profile)
    elif "medication" in utterance.lower() or "take" in utterance.lower():
        response_text = handle_medication_query(patient_profile)
    else:
        # Fallback to Nova Lite agent
        response_text = invoke_nova_lite(utterance, patient_profile)

    # Log conversation
    log_conversation(patient_id, utterance, response_text, escalation_level=0)

    return build_response(response_text)
```

def handle_routine_query(patient_profile):
"""
Handle 'What do I do next?' queries
"""
current_hour = datetime.now().hour

```
    # Determine time of day
    if 5 <= current_hour < 12:
        time_period = "morning"
    elif 12 <= current_hour < 17:
        time_period = "afternoon"
    else:
        time_period = "evening"

    # Find matching routine
    routines = patient_profile.get('routines', [])
    matching_routine = next((r for r in routines if r.get('timeOfDay') == time_period
    if not matching_routine:
        return f"Let's relax this {time_period}. Would you like to chat?"

    steps = matching_routine.get('steps', [])
    context = matching_routine.get('context', '')

    if steps:
        return f"Let's {steps[0]}. {context}"
    else:
        return "I'm here if you need anything."
```

def handle_people_query(utterance, patient_profile):
"""
Handle 'Who is X?' queries
"""
# Extract name from utterance (simple regex for Week 1)
import re
match = re.search(r"who is (\w+)", utterance.lower())

```
if not match:
    return "Who would you like to know about?"

name_query = match.group(1).capitalize()

# Search people array
people = patient_profile.get('people', [])
person = next((p for p in people if p.get('name', '').lower() == name_query.lower

if not person:
    return f"I don't have information about {name_query}. Would you like to asl

relationship = person.get('relationship', 'someone special')
visit_schedule = person.get('visitSchedule', '')
activities = person.get('sharedActivities', [])

response = f"{name_query} is your {relationship}."
if visit_schedule:
    response += f" {name_query} visits on {visit_schedule}."
if activities:
    response += f" You both love {activities[0]}."

return response
```

def handle_medication_query(patient_profile):
"""
Handle medication queries
"""
medications = patient_profile.get('medications', [])

```
    if not medications:
        return "I don't have your medication schedule. Please check with your careg

    # For Week 1, return first medication with timing
    med = medications[0]
    name = med.get('name', 'your medication')
    timing = med.get('timing', 'as prescribed')
    appearance = med.get('appearance', '')

    response = f"You take {name}, {timing}."
    if appearance:
        response += f" It's {appearance}."
    response += " Please confirm with your caregiver if you're unsure."

    return response
```

def build_response(speech_text):
"""
Build Alexa-compatible response
"""
return {
'version': '1.0',
'response': {
'outputSpeech': {
'type': 'PlainText',
'text': speech_text
},
'shouldEndSession': False
}
}

File: utils/dynamodb_helper.py

```
import boto3
from datetime import datetime

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
table = dynamodb.Table('BuddyPatientProfiles')

def get_patient_profile(patient_id):
"""
Retrieve patient profile from DynamoDB
"""
try:
response = table.get_item(Key={'patientId': patient_id})
return response.get('Item')
except Exception as e:
print(f"Error fetching patient profile: {e}")
return None

def log_conversation(patient_id, utterance, response, escalation_level=0):
"""
Append conversation to history
"""
try:
conversation_entry = {
'timestamp': datetime.utcnow().isoformat(),
'userUtterance': utterance,
'assistantResponse': response,
'escalationLevel': escalation_level,
'repeatCount': 1 # TODO: Calculate actual repeat count
}
```

```
        table.update_item(
            Key={'patientId': patient_id},
            UpdateExpression='SET conversationHistory = list_append(if_not_exists(co
            ExpressionAttributeValues={
                ':entry': [conversation_entry],
                ':empty_list': []
            }
        )
        print(f"Logged conversation for patient {patient_id}")
    except Exception as e:
        print(f"Error logging conversation: {e}")
```

File: utils/bedrock_helper.py

```
import boto3
import json
```

bedrock = boto3.client('bedrock-runtime', region_name='us-east-1')

def invoke_nova_lite(utterance, patient_profile):
"""
Invoke Nova Lite for generic queries (Week 1 placeholder)
"""
# Simplified invocation - will expand with tool definitions in Week 2
patient_name = patient_profile.get('profile', {}).get('preferredName', 'there')

```
prompt = f"""You are Buddy, a compassionate assistant for people with dementi
```

Patient name: {patient_name}

Patient said: "{utterance}"

Respond in 1-2 short, calming sentences (10-15 words each). Be warm and reassuring."""

```
try:
    response = bedrock.invoke_model(
        modelId='amazon.nova-lite-v2:0',
        contentType='application/json',
        accept='application/json',
        body=json.dumps({
            'messages': [
                {
                    'role': 'user',
                    'content': [{'text': prompt}]
                }
            ],
            'inferenceConfig': {
                'max_new_tokens': 100,
                'temperature': 0.7
            }
        })
    )

    response_body = json.loads(response['body'].read())
    assistant_message = response_body['output']['message']['content'][0]['text']
    return assistant_message

except Exception as e:
    print(f"Error invoking Nova Lite: {e}")
    return "I'm here for you. Let's take this one step at a time."
```

File: `requirements.txt`

boto3==1.34.28

Deploy Lambda Function

cd buddy-lambda

Install dependencies

pip install -r requirements.txt -t .

Create deployment package

zip -r buddy-lambda.zip .

Deploy to AWS Lambda

aws lambda create-function
--function-name BuddyAlexaSkillHandler
--runtime python3.11
--role arn:aws:iam::052080186586:role/BuddyLambdaExecutionRole
--handler lambda_function.lambda_handler
--zip-file fileb://buddy-lambda.zip
--timeout 30
--memory-size 512
--region us-east-1
--profile buddy-dev

Note ARN from output:
arn:aws:lambda:us-east-1:052080186586:function:BuddyAlexaSkillHandler

---

Day 7: End-to-End Testing

Test Lambda Locally

File: `test-event.json`

{
"session": {
"user": {
"userId": "test-patient-001"
}
},
"request": {

```
"intent": {
"slots": {
"query": {
"value": "What do I do this morning?"
}
}
}
}
}
```

Invoke test:

```
aws lambda invoke
--function-name BuddyAlexaSkillHandler
--payload file://test-event.json
--region us-east-1
--profile buddy-dev
response.json
```

```
cat response.json
```

Expected response:

```
{
"version": "1.0",
"response": {
"outputSpeech": {
"type": "PlainText",
"text": "Let's brush teeth. Toothbrush is in the blue cup by the sink"
},
"shouldEndSession": false
}
}
```

Test Scenarios

Scenario 1: Routine Query

```
{
"request": {
"intent": {
"slots": {
"query": {"value": "What do I do next?"}
}
}
}
}
```

Scenario 2: People Query

```
{
"request": {
"intent": {
"slots": {
"query": {"value": "Who is Sarah?"}
}
}
}
}
```

Scenario 3: Medication Query

```
{
"request": {
"intent": {
"slots": {
"query": {"value": "What medication do I take?"}
}
}
}
}
```

Week 1 Checklist

Day 1-2: AWS Setup ✅

- [ ] Bedrock model access granted (Nova 2 Sonic + Lite)
- [ ] AWS CLI configured with lopezdev credentials
- [ ] IAM role created (BuddyLambdaExecutionRole)
- [ ] Bedrock invoke permissions attached

Day 3-4: DynamoDB ✅

- [ ] BuddyPatientProfiles table created
- [ ] Test patient "John Doe" inserted
- [ ] Data retrieval verified

Day 5-6: Lambda Function ✅

- [ ] Lambda function deployed
- [ ] DynamoDB helper functions working
- [ ] Bedrock helper connects to Nova Lite
- [ ] Basic intent routing functional

Day 7: Testing ✅

- [ ] Routine query returns correct morning steps
- [ ] People query returns Sarah's information
- [ ] Medication query returns Donepezil details
- [ ] Conversation logged to DynamoDB

Troubleshooting

Issue: Bedrock Access Denied

Error:
An error occurred (AccessDeniedException) when calling the InvokeModel operation

Solution:

1. **First-time invocation auto-enables models**—no manual activation needed post-2026 AWS update
2. Check IAM role has `bedrock:InvokeModel` permission in policy
3. Confirm model ID is correct: `amazon.nova-lite-v2:0` (not v1:0)
4. Verify region is `us-east-1` (some models have regional restrictions)
5. For Anthropic models (if you switch later), submit use case details on first invocation

Issue: Lambda Timeout

Error:
Task timed out after 3.00 seconds

Solution:

```
aws lambda update-function-configuration
--function-name BuddyAlexaSkillHandler
--timeout 30
--profile buddy-dev
```

Issue: DynamoDB Item Not Found

Error:
'Item' key not found in response

Solution:

Verify table exists

```
aws dynamodb describe-table
--table-name BuddyPatientProfiles
--profile buddy-dev
```

Re-insert test patient

```
aws dynamodb put-item
--table-name BuddyPatientProfiles
--item file://test-patient-john.json
--profile buddy-dev
```

Cost Tracking (Week 1)

Actual spend estimate:

- Lambda invocations: $0.00 (Free Tier: 1M requests)
- DynamoDB storage: $0.00 (Free Tier: 25GB)
- Nova Lite calls: ~$0.50 (50 test queries @ $0.01 each)
- **Total Week 1:** ~$0.50

Remaining budget: $99.50 from $100 hackathon promo credits

Next Steps: Week 2 Preview

Days 8-14 will focus on:

1. **Nova Lite Agentic Workflow** - Define tools for DynamoDB queries, implement tool orchestration
2. **Safety Classification Logic** - Keyword rules + LLM risk scoring for 3-level escalation
3. **Caregiver Alert System** - SNS integration for SMS notifications with conversation summaries

Preparation for Week 2:

- Keep testing Lambda with various queries
- Review Nova Lite tool definition syntax
- Draft safety keyword lists (Level 1 vs Level 2 triggers)

You've completed Week 1 infrastructure setup! Lambda is querying DynamoDB and invoking Nova Lite. Ready to add agentic capabilities next week! ▯