

準備好探索更多功能了嗎？歡迎註冊或登入，開始使用 Notion 建立內容。

註冊或登入

# 黑客松



Date

空



Status

空



影片網址

空

## ▼ ICAM

### ▼ 影片



OneDrive File

## ▼ ICAM 540



AWS\_ICAM\_Quick\_Start.md chunyu1021/AI\_Hackathon\_2025  
main

### 0. 系統更新 & 基本工具

( 沒有 build-essential、git、cmake 會連 make 都跑不了 )

```
sudo apt update && sudo apt upgrade -y # 工具鏈 + CMake + Git + pkg-config + m4
sudo apt install -y build-essential cmake git pkg-config m4
```

### 1. 安裝 KVS-SDK 依賴 (GStreamer + 基本 Lib)

官方 README 指定的最小組合如下：

```
sudo apt install -y \ libssl-dev libcurl4-openssl-dev liblog4cplus-dev \
libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev \ gstreamer1.0-plugins-base-apps gstreamer1.0-plugins-good \
gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly \ gstreamer1.0-tools
```

若之後你還要 JNI，再另外裝：

```
sudo apt install openjdk-17-jdk
```

並在 CMake 加 `-DBUILD_JNI=TRUE`

### 2. 下載原始碼並建立 build 目錄

```
git clone https://github.com/aws-labs/amazon-kinesis-video-streams-producer-sdk-cpp.git
mkdir -p amazon-kinesis-video-streams-producer-sdk-cpp/build
cd amazon-kinesis-video-streams-producer-sdk-cpp/build
```

### 3. CMake 設定 ( 開啟 GStreamer 外掛 )

```
cmake -DBUILD_GSTREAMER_PLUGIN=TRUE ..
```

如需關閉內建依賴（自己系統有新版 OpenSSL/cURL），可加

- `DBUILD_DEPENDENCIES=OFF`

## 4. 編譯

```
make # 完成後 build/ 會出現： # libgstkvssink.so ← GStreamer 外掛 # kvs_gstreamer_sample ← 範例推流程式
```

## 5. 設定執行期環境變數

```
cd .. # 回到 repo 根目錄 export GST_PLUGIN_PATH=$PWD/build export LD_LIBRARY_PATH=$PWD/open-source/local/lib
```

之後若搬到別處，記得把兩個路徑改成新位置。

## 6. 準備 AWS 認證

```
export AWS_ACCESS_KEY_ID=<YourAccessKey> export AWS_SECRET_ACCESS_KEY=<YourSecretKey> export AWS_DEFAULT_REGION=<ap-northeast-1 等>
```

( 或 `aws configure` + IAM Role 均可 )

## 7. 實際推流指令

將 `<YourStreamName>` 改成你在 Kinesis Video Streams 主控台建立的名稱。

```
gst-launch-1.0 -e v4l2src do-timestamp=TRUE device=/dev/video10 ! \ vide
oconvert ! video/x-raw,format=I420,width=1920,height=1080,framerate=30/1
! \ x264enc bframes=0 key-int-max=30 bitrate=500 tune=zerolatency ! \ h2
64parse ! video/x-h264,stream-format=avc,alignment=au,profile=baseline !
\ kvssink stream-name="<YourStreamName>" storage-size=512 fragment-durat
ion=2000
```

## 8. ( 可選 ) 驗證與排錯

檢查項目	指令 / 現象
kvssink 已被 GStreamer 辨識	<code>gst-inspect-1.0 kvssink</code>
AWS 認證 OK	<code>aws sts get-caller-identity</code>
推流成功但看不到畫面	KVS 主控台點「HLS playback」→ 等待 10-20 秒緩衝
<code>/dev/video10</code> 不存在	確定 ICAM Web Tool 「Playing」； 或 <code>v4l2-ctl --list-devices</code>

## ( 一鍵腳本 ) 全部合併

```
#!/bin/bash sudo apt update && sudo apt install -y build-essential cmake
git pkg-config m4 \ libssl-dev libcurl4-openssl-dev liblog4cplus-dev \ l
ibgstreamer1.0-dev libgstreamer-plugins-base1.0-dev \ gstreamer1.0-plugi
ns-base-apps gstreamer1.0-plugins-good \ gstreamer1.0-plugins-bad gstrea
mer1.0-plugins-ugly gstreamer1.0-tools git clone https://github.com/aws/
amazon-kinesis-video-streams-producer-sdk-cpp.git cd amazon-kinesis-
video-streams-producer-sdk-cpp mkdir build && cd build cmake -DBUILD_GST
REAMER_PLUGIN=TRUE .. make -j$(nproc) cd .. export GST_PLUGIN_PATH=$PWD/
build export LD_LIBRARY_PATH=$PWD/open-source/local/lib echo "SDK build
done. Set AWS env vars then run gst-launch to push stream."
```

把這段存成 `install_kvs_sdk.sh`，給新 Ubuntu `bash install_kvs_sdk.sh`，整套環境就就緒。

### ▼ 登入 ICAM 540

完成！你現在已能在任何乾淨 Ubuntu 主機上，跑 ICAM-540 → Kinesis Video Streams 的完整流程。

```
ssh icam-540@192.168.0.100
```

## ▼ 執行.py

### ① 確定你有 python3 與 pip

```
python3 --version python3 -m pip --version # 如果 pip 沒裝，看下一步
```

### ② 若沒有 pip，先在 Ubuntu 裝它

```
sudo apt update sudo apt install -y python3-pip # 帶進 python3-setuptools
```

### ③ ( 可選 ) 進虛擬環境，避免污染系統

```
python3 -m venv venv source venv/bin/activate # 之後提示會變 (venv) $
```

### ④ 安裝 OpenCV：用 headless 版省掉 GUI 依賴

```
python3 -m pip install --upgrade pip # 先把 pip 升級 python3 -m pip  
install opencv-python # 或：opencv-python-headless
```

## ▼ 流程說明

### ▼ Enviroment

#### ICAM-540

部署於被照護者房間或公共區域的智慧相機，負責即時擷取現場的影像與音訊。

#### Video Stream

ICAM-540 將原始視訊串流 ( H.264 、 RTSP 或透過 v4l2loopback ) 推送至後端處理管線。

#### Pre-processing / Anomaly Filter

對影像幀與音訊訊號進行初步清洗與格式轉換 ( 去噪、色彩與取樣格式標準化 ) 。

過濾掉常見的無效幀 ( 空白、遮擋 ) 或背景雜訊，只保留疑似異常的片段 ( 如跌倒、非正常動作、異常聲響 ) 交由 Agent 後續處理。

### ► 詳細說明

#### Sensors

- **TOF / IR / RGB 攝影機**：提供深度感測、熱像與可見光三種模態的資料，增強夜間或遮擋狀況下的感知能力。
- **音訊擷取**：麥克風收錄咳嗽、喘息、呼救聲等聲波，補足視覺感知的盲點。

#### Feedback

- **Success/Fail**：記錄後續執行動作 ( 如發出警示、送出通知 ) 是否成功到達，以及護理人員是否已接收。
- **Event Log**：將每次偵測到的事件與系統回應結果寫入日誌，以供後續系統自我評估與持續優化。

## AI Agent

## ▼ Perception

### 1. 視覺：人臉 / 跌倒 / 行為偵測

- 服務：
  - SageMaker JumpStart—Object Detection (COCO SSD)
  - SageMaker JumpStart—Anomaly Detection (如有提供)

### 建置步驟

#### 1. 部署模型 Endpoint

- 在 SageMaker Console → JumpStart → Vision 模型 → 選擇「Object Detection (COCO SSD)」→ Deploy Real-time Endpoint → 取名 `jumpstart-vision-od`。
- 若 JumpStart 有「Anomaly Detection」範本，也可同樣部署第二支 Endpoint (例如 `jumpstart-vision-anom`)。

#### 2. 影像前處理

- 截幀：用 GStreamer 或 FFmpeg 將 H.264 串流解成固定速率 (如每秒 1–5 幀) 的 JPEG/PNG。
- 尺寸 / 色彩規格化：轉成模型輸入大小 (如 640×480 RGB)，可用 Pillow 或 OpenCV：

```
python 複製編輯 from PIL import Image im = Image.open('frame.jpg').convert('RGB').resize((640,480)) im.save('frame_norm.png')
```

#### 3. 呼叫 Endpoint

```
python 複製編輯 import boto3, json sm = boto3.client('sagemaker-runtime') resp = sm.invoke_endpoint( EndpointName='jumpstart-vision-od', ContentType='image/png', Body=open('frame_norm.png','rb').read() ) preds = json.loads(resp['Body'].read())['predictions'] # preds 內含每個 [label] (e.g. person) 與 [score]
```

#### 4. 事件標記

- 若 `pred.label == 'person'` 或 `pred.label == 'fall'`，且 `pred.score ≥ 0.6` → 標為「偵測到人 / 跌倒事件」
- 若 JumpStart Anomaly Endpoint 回傳高於某閾值的異常分數 → 標為「行為異常」



## 2. 音訊：咳嗽 / 語調分析

- Amazon Transcribe：將音訊轉成文字（支援即時串流與批次轉寫）。
- Amazon Bedrock LLM（如 Claude 3 或 Titan Text）：對轉寫結果做關鍵詞或語意判斷，回傳是否含有「咳嗽、喘息、呼救」等異常事件。

## 建置步驟

### □ 建立 Transcribe 流程

#### 1. 即時串流 (Real-time) 或 批次 (Batch) 轉寫

- 即時：使用 [Transcribe Streaming SDK](#)
- 批次：將錄好的音訊檔（WAV/MP3）上傳至 S3，然後透過 `StartTranscriptionJob` API 觸發

#### 2. 範例：啟動批次轉寫

```
python 複製編輯 import boto3 transcribe = boto3.client('transcribe')
job_name = 'audio-transcription-job' s3_uri = 's3://your-bucket/input/audio.wav' output_uri = 's3://your-bucket/output/' transcribe.start_transcription_job( TranscriptionJobName=job_name, Media={'MediaFileUri': s3_uri}, MediaFormat='wav', LanguageCode='zh-TW', OutputBucketName='your-bucket' ) # 等待完成後，結果會寫入 output_uri 下的 JSON
```

## 2. 取得與解析轉寫結果

- 轉寫完成後，Transcribe 會在 S3 產生一個 JSON 檔，結構類似：

```
jsonc 複製編輯 { "results": { "transcripts": [{ "transcript": "他剛才咳嗽了三聲，然後大口喘氣" }], "items": [ ... ] } }
```

- 讀入 `transcript` 欄位，即可取得純文字內容。

## 3. 生理訊號：HR / BP / SpO2 數值輸入

- 服務：
  - AWS IoT Core（裝置佈署、MQTT 傳送）
  - Amazon Timestream（時序資料庫存取）

## 建置步驟

## 1. IoT 連線

- 在 AWS IoT Core 中為每台生理感測裝置 ( BLE 、 Zigbee 、 LoRa ) 設定 Thing 。
- 設定 MQTT Topic ，如 `sensors/<userId>/physio` ，且綁定權限 Policy 。

## 2. 裝置端上傳

- 每秒或每分鐘讀取一次心率 / 血壓 / 血氧，透過 TLS MQTT 發佈 JSON：

```
json 複製編輯 { "timestamp": 1682400000000, "HR": 78, "BP": 120/80, "SpO2": 97 }
```

## 3. Timestream 寫入

- IoT Core Rule → 觸發 AWS Lambda → 將資料寫入 Timestream：

```
python 複製編輯 import boto3 ts = boto3.client('timestream-write') ts.write_records( DatabaseName='PhysioDB', TableName='UserMetrics', Records=[{ 'Dimensions':[{ 'Name': 'UserId', 'Value':userId}], 'MeasureName': 'HR', 'MeasureValue':str(hr), 'MeasureValueType': 'DOUBLE', 'Time':str(timestamp) }, ... ] )
```

## 4. 事件標記

- Lambda 可同時檢查閾值 ( 如  $HR > 100$  、  $SpO_2 < 90$  ) → 若異常 → 標為「生理異常事件」

---

### 總結：

- 視覺：JumpStart Vision Endpoint → 呼叫 → 閾值判定 → 人 / 跌倒 / 異常標記
- 音訊：JumpStart Audio (YAMNet) → 呼叫 → 關鍵 label 閾值判定 → 咳嗽 / 喘息標記
  - Transcribe：音訊→文字
  - Bedrock LLM：文字→異常判斷
  - 業務邏輯：`anomaly` 欄位驅動「可疑/略過」流程
- 生理：IoT Core MQTT → Lambda → Timestream → 閾值判定 → 生理異常標記

## ▼ Memory & Knowledge Base

### 1. 個人歷史事件 – Amazon DynamoDB

#### 服務特性

- NoSQL、高可用、低延遲
- Auto Scaling：自動調整讀寫容量
- TTL：自動過期刪除舊事件

#### 建置步驟

##### 1. 建立 DynamoDB 表格

- TableName：UserEventHistory
- Partition Key：UserId (String)
- Sort Key：EventTime (Number, Unix ms)
- TTL 屬性：設定 ExpiryTime 欄位 ( Unix 秒 )，自動刪除 90 天以上紀錄。

##### 2. Lambda 寫入

Agent 每偵測到一次事件 ( 跌倒、可疑聲音、異常生理訊號 ) 就呼叫 Lambda，執行：

```
python 複製編輯 import boto3, time
ddb = boto3.resource('dynamodb')
tbl = ddb.Table('UserEventHistory')
def record_event(user_id, event_type, metadata):
    now_ms = int(time.time()*1000) # TTL 90 天後
    expiry = int(time.time()) + 90*24*3600
    tbl.put_item(Item={ 'UserId': user_id,
                        'EventTime': now_ms,
                        'EventType': event_type, # e.g. "FALL", "COUGH"
                        'Metadata': metadata, # e.g. {"confidence":0.8}
                        'ExpiryTime': expiry })
```

##### 3. DynamoDB 查詢

Decision 階段需要「某人過去 X 小時內跌倒過嗎？」可用 Query：

```
python 複製編輯 from boto3.dynamodb.conditions import Key
cutoff = now_ms - X*3600*1000
resp = tbl.query( KeyConditionExpression=Key('UserId').eq(user_id) & Key('EventTime').gt(cutoff),
                  FilterExpression="EventType = :et",
                  ExpressionAttributeValues={":et":"FALL"} )
recent_falls = resp['Items']
```

## 2. 向量化病例检索 – Amazon OpenSearch Service

### 服務特性

- 全文 + KNN 向量搜尋 ( 支援 OpenSearch Serverless 或 Provisioned )
- 高可用、多 AZ
- 與 AWS 身分、IAM、CloudWatch 完整整合

### 建置步驟

#### 1. 建立 OpenSearch Serverless Collection

- Console → OpenSearch Serverless → Create collection → 啟用 **kNN vector search**
- 定義 Mapping :

```
json 複製編輯 { "properties": { "case_id": { "type": "keyword" },  
  "vector": { "type": "knn_vector", "dimension": 1536 }, "text_snip  
p": { "type": "text" } } }
```

#### 2. Embedding 生成

- 使用 **Amazon Bedrock Embeddings API** ( 或 SageMaker JumpStart embeddings 模型 ) 把每份歷史病例摘要 / SOP 段落轉成 1536 維向量 :

```
python 複製編輯 bedrock = boto3.client('bedrock-runtime') resp =  
bedrock.invoke_model( modelId='anthropic.claude-embeddings', cont  
entType='application/json', body=json.dumps({"input": text_snip})  
) embedding = json.loads(resp['body'].read())['embedding']
```

#### 3. 索引至 OpenSearch

```
python 複製編輯 from opensearchpy import OpenSearch os_client = OpenS  
earch( hosts=[{'host': 'xxx', 'port': 443}], http_auth=('user', 'pas  
s'), use_ssl=True, verify_certs=True ) os_client.index( index='case-v  
ectors', id=case_id, body={'vector': embedding, 'text_snip': text_sni  
p} )
```

#### 4. 相似案例檢索

Decision 時，將當前事件摘要做同樣 embedding，然後呼叫 kNN search：

```
python 複製編輯 query_embedding = ... # 上同 resp = os_client.search(
index='case-vectors', knn={'vector': {'vector': query_embedding, 'k':
3}} ) similar_cases = [hit['_source']['text_snip'] for hit in resp['h
its']]['hits']]
```

### 3. Domain SOP 資料庫 – Amazon S3 + Amazon Bedrock Retrieval

#### 服務特性

- **S3**：海量存放、版本控制、加密
- **Amazon Bedrock Retrieval (RAG)**：於 Bedrock 端對 S3 文檔做自動 CHUNK、Embedding、Index，並提供檢索 API

#### 建置步驟

## 1. SOP 文件存放

- 在 S3 建立 `s3://my-sop-bucket/`
- 上傳所有 .md/.txt 格式的 SOP 文檔：

```
arduino 複製編輯 s3://my-sop-bucket/ fall-response.md hypertension-alert.md medication-noncompliance.md
```

### ▼ SOP

```
--- title: "跌倒應急處置流程" id: "001" version: "1.0"
last_updated: "2025-04-20" --- # 跌倒應急處置流程 ## 目的 簡要說明
此流程的目的—在使用者跌倒後，能迅速啟動正確的處理程序。 ## 適用範圍
- 被照護者老人公寓 - 夜間值班護理人員 ## 前置條件 1. 監控系統已正常
運作 2. 護理站距離≤5分鐘步程 ## 作業步驟 1. **辨識並確認跌倒** - 收
到系統告警或目視確認。 2. **評估現場安全** - 確保周圍無危險物或滑倒風
險。 3. **緊急呼救** - 使用護理站對講系統呼叫支援。 4. **初步檢查**
- 檢查意識、呼吸、出血情況。 5. **通知家屬 / 醫師** - 如有必要，立即
轉送醫療機構。 ## 注意事項 - 全程保持與被照護者溝通 - 避免拉扯關節
## 參考文件 - `002_medication-noncompliance.md` - ISO 9001:2015 品
質管理手冊
```

## 2. Bedrock Retrieval 索引

- Console → Amazon Bedrock → Knowledge Store → Create store → 指定 SOP S3 bucket
- Bedrock 會自動將文檔分段、產生 embedding、並建立可檢索引

### 3. RAG 檢索呼叫

Decision 階段，用 Bedrock Retrieval API 對 SOP Knowledge Store 查詢：

```
python 複製編輯 resp = bedrock.invoke_model( modelId='amazon-bedrock-retrieval', contentType='application/json', accept='application/json', body=json.dumps({ "query": "使用者跌倒後立即要做什麼步驟?", "top_k": 2, "knowledgeStoreId": "my-sop-store" }) ) top_sops = json.loads(resp['body'].read())['retrieved_chunks']
```

## 什麼是 Knowledge Store ?

在 Bedrock 中，「Knowledge Store」就是一個託管的文件檢索庫。你指定要索引的 S3 bucket 路徑，Knowledge Store 會：

1. **拉取文件**：掃描並下載你指定的所有文字檔（支援 `.md`，`.txt`，`.pdf`，`.html` 等格式）。
2. **文件切段 (Chunking)**：依照設定的最大字數或段落分隔規則，把長文件拆成多個「文檔段 (chunks)」。例如每 500 字或遇到二級標題就切一次。
3. **Embedding 產生**：對每個 chunk 自動呼叫 Bedrock Embeddings 模型（如 Amazon Embeddings）產生向量表示。
4. **索引建立**：將所有 chunk 的向量與原始文字一併儲存到一個託管的向量索引中，支援高速 KNN 檢索。
5. **Metadata 儲存**：對每個 chunk，其來源檔案路徑、文件名稱、chunk 序號、YAML metadata（若有）都會一併儲存。

一旦索引完成，你就可以直接對 Knowledge Store 發搜尋請求（RAG），Bedrock 會在後端跑 kNN，幫你找出「最相關的 SOP 片段」，然後把它們回傳給 LLM 用在 prompt 裡。

## 建立 Knowledge Store 的詳細步驟

### 1. 準備 SOP 文件到 S3

- 在 S3 上建立 bucket（例如 `my-sop-bucket`），並上傳你的 `.md` / `.txt` / `.pdf` 文件到子目錄 `sop/`。

### 2. 開啟 Bedrock Console

- 登入 AWS Console → 服務選單搜尋 **Bedrock** → 進入 **Amazon Bedrock** 管理介面。

### 3. 進入 Knowledge Store

- 左側選單點 Knowledge stores → 按下 Create knowledge store。

### 4. 配置資料來源 (Data source)

- Data source type：選 `S3`。
- S3 path：輸入 `s3://my-sop-bucket/sop/`。
- (可選) 設定 IAM role：讓 Bedrock 能夠讀取該 bucket，若尚未有，會引導你建立一個具有 `s3:GetObject` 權限的 role。

### 5. 設定切段規則 (Chunking rule)

- Chunk size：指定每個 chunk 最多多少字元 (預設 500–1000 字元)。
- Chunk overlap：若希望上下文銜接，可設定重疊字元數 (例如 50 字元)，增加檢索品質。
- 段落分隔符：預設會以二級以上標題 (`##`、`###`) 作為自然切點。

### 6. 選擇 Embeddings 模型

- Embedding model：可選用 Bedrock 自帶的 `amazon-embedding-multi-5x1` (或其他你在帳號中可用的模型)。

### 7. 建立與執行

- 確認設定後按 **Create**。
- Bedrock 會進入「索引中 (Indexing)」狀態，並顯示目前進度。這過程可能視文件量長短而定，通常幾分鐘內完成。

### 8. 監控與管理

- 索引完成後 Knowledge Store 狀態變成 **Active**。
- 你可以在 Console 中看到「總共分了多少 chunks」、每個檔案對應多少 chunk，以及 embedding 成功率等指標。

---

## 使用索引做檢索 (RAG)

當 Knowledge Store 處於 Active，Decision 階段就能呼叫 Bedrock Retrieval API：



## ▼ Planning / Decision ( 規劃與決策 )

### 1. RAG Retrieval ( 檢索增強生成 )

結合「向量化檢索」與「檢索式生成 (Retrieval-Augmented Generation)」，先將問題與相關文件段落融合，再交給 LLM。

#### 4. 輸入再 Prompt

- 把 fall-response 前兩段，"chunk" 到 LLM 的 system prompt，再做最終決策或指令生成，確保建議符合 SOP。
- 加入相關上下文：
- query：你的自然語言查詢
- 個人歷史：從 DynamoDB 查到最近一次跌倒時間、頻率。
- top\_k：要回傳最相關的前 K 段文字
- 相似案例：從 OpenSearch 找到過去類似跌倒案例處置摘要。
- knowledgeStoreId：知識庫的唯一識別碼
- SOP 段落：由 Bedrock Retrieval 拿到「跌倒應急處置流程」文件中最相關傳後，關聯三段，把這幾段 chunk 直接插入 LLM 的 system prompt，確保模型回覆時「有標準作業流程當參考」，兼顧準確與可審計。

#### 2. 檢索步驟

```
text 複製編輯 Query: "使用者跌倒後首要處置步驟?" |— DynamoDB: 查: user-123, 最近 24 小時內無跌倒 |— OpenSearch: kNN 檢索 → 取 3 節相似「跌倒案例」說明 |— Bedrock Retrieval: 取 2 段「001_fall-response.md」SOP
```

#### 3. 組成 RAG Prompt

```
markdown 複製編輯 # System: 你是長照專家，目前收到一位老人跌倒事件。以下資訊供你參考： 1. 歷史：此人過去 48 小時無跌倒紀錄。 2. 相似案例： - 病例 A: ... - 病例 B: ... 3. SOP: - 步驟 1: 確認安全 - 步驟 2: 呼救 # User: 事件：使用者於 14:32 跌倒，請根據上方資料給出最優處置建議。
```

## 2. LLM 推理 ( Amazon Bedrock Claude 3 / Titan Text )

利用 Retrieval 結果為上下文，讓大型語言模型深度推理、生成具體、合規、可操作的決策或建議。

#### 1. 選擇模型

- Claude 3：強在長篇、穩定性高、細節敘述。
- Titan Text：回應速度較快、輕量且成本較低。

## 2. 呼叫 Bedrock Run

```
json 複製編輯 { "modelId": "anthropic.claude-3", "prompt": "<前述 RAG Prompt>", "maxTokens": 256, "temperature": 0.0 }
```

- **temperature=0**：生成 deterministic 回答，杜絕隨機性。
- **maxTokens**：限制回答長度，保證回傳重點清晰。

## 3. 解析回應

模型回傳一段文字，包含：「立即確保現場安全 → 立即呼救 → 檢查意識並提供初步急救 → 通知家屬...」，你可將其結構化成 JSON，以便下一步政策引擎讀取。

# 3. Policy Engine ( 策略引擎：AWS Step Functions )

將 LLM 生成的建議，編排成可執行的工作流程 ( workflow )，並撰寫成有狀態管理、錯誤重試、並行分支的 Step Functions State Machine。

## 1. 設計 State Machine

- **State 1: Alert Notification**
  - 功能：呼叫 SNS/Pinpoint 推送通知給護理人員與家屬。
- **State 2: Emergency Call**
  - 功能：若模型建議「撥打 119」，透過 API 觸發第三方通訊服務。
- **State 3: Record Event**
  - 功能：把最終決策結果寫回 DynamoDB History，作為後續學習資料。
- **State 4: Monitoring Check**
  - 功能：等待並檢查護理人員回覆 ( 可設定 Heartbeat 機制或人工標註觸發 )，若超時則進行 Escalation 分支通知更高層級。

## 2. 錯誤重試與補償

- 每個 Task 加上 **Retry** 與 **Catch** 配置，例如通知失敗可重試 3 次，最終失敗則寫入 Dead Letter。

## 3. 並行與條件分支

- 當需要同時執行「聯絡家屬」與「通知護理站」時，用 **Parallel** State。
- 根據 LLM 回傳 JSON 中的 **requiresMedicalEvacuation: true**，加分支執行「呼叫救護車」。

## ▼ 4. 啟動與監控 Action

- Lambda 或 API Gateway 觸發該 State Machine 執行，並在 CloudWatch Logs / X-Ray 監控執行狀態與效能。

### 1. 在 EC2 的後端新增“陪伴聊天”服務

• 可在 Step Functions Console 看到可視化流程圖，追蹤每一步的執行結果。假設你原本在 EC2 上跑一個 Flask + Gunicorn，監聽 3000 端口，現在加一個新路由

`/api/chat/soothing`，流程如下：

#### 1. 透過 Bedrock 呼叫 LLM

- System Prompt 設為「你是個溫暖的聊天夥伴，專門陪長照人士，避免他們自殘，安撫他們情緒。」
- User Prompt 帶入從前端傳來的「今天心情如何？」或偵測到的行為訊號

#### 2. 用 Polly 合成 LLM 的回應

- 把 Bedrock 回傳的文字用 Polly 轉成 MP3

#### 3. 回傳給前端

- 讓前端像播放警示一樣，播放安撫語音

### 範例程式 ( `app/main.py` )

```
import os, json, base64 from flask import Flask, request, jsonify import boto3 app = Flask(__name__) bedrock = boto3.client('bedrock-runtime', region_name='us-west-2') polly = boto3.client('polly', region_name='us-west-2') SYSTEM_PROMPT = """ You are a compassionate companion for elderly care residents, focused on soothing their emotions and preventing self-harm. Speak gently and empathetically. """ @app.route('/api/chat/soothing', methods=['POST']) def soothing_chat(): """ 前端傳入 JSON: { "userId": "...", "input": "我好難過..." } 回傳 JSON: { "audioBase64": "...", "text": "..."} """ data = request.get_json() user_input = data.get('input', '') # 1. 呼叫 Bedrock LLM prompt = SYSTEM_PROMPT + "\nUser: " + user_input + "\nCompanion:" resp = bedrock.invoke_model(modelId='anthropic.claude-3', # 或 titan-text contentType='application/json', accept='application/json', body=json.dumps({ "prompt": prompt, "max_tokens": 128, "temperature": 0.7 })) result = json.loads(resp['body'].read()) reply = result.get('completions', [{}])[0].get('data', {}).get('text', "抱歉，我在這裡陪著你。") # 2. 用 Polly 合成 polly_resp = polly.synthesize_speech(Text=reply, VoiceId='Zhiyu', OutputFormat='mp3') mp3 = polly_resp['AudioStream'].read() b64_audio = base64.b64encode(mp3).decode('utf-8') return jsonify({"text": reply, "audioBase64": b64_audio})
```

### 2. 前端呼叫並播放

在你的前端 ( Playground 或自訂 UI ) 加入按鈕或自動觸發：

```
async function playSoothing(inputText) { const res = await fetch('/api/chat/soothing', { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify({ input: inputText }) }); const { audioBase64, text } = await res.json(); // 顯示文字回應 (可選) document.getElementById('chatReply').innerText = text; // 播放語音 const bytes = Uint8Array.from(atob(audioBase64), c=>c.charCodeAt(0)); const blob = new Blob([bytes], { type: 'audio/mp3' }); const url = URL.createObjectURL(blob); const audio = new Audio(url); audio.play(); } // 範例：使用者按“聊聊”按鈕時 document.getElementById('sootheBtn').addEventListener('click', ()=>{ const userMsg = document.getElementById('userInput').value; playSoothing(userMsg); });
```

### 3. 部署步驟回顧

1. 更新後端程式：把上面程式碼加入 `app/main.py`。
2. 重新 build & restart：

```
cd ~/voice-agent docker-compose build backend docker-compose up -d backend
```

3. 測試：打開前端 UI，輸入類似「我很孤單」的文字，按下「聊聊」按鈕，應該會聽到溫柔的安撫語音並看到回應文字。

這樣，你的 EC2 上不但能 偵測警示，也能 主動陪伴長照人士，在他們情緒低落或有自殘念頭時，提供溫暖的聊天關懷。

## 1. 在 AWS Console 建立 HealthLake Datastore

1. 登入 AWS Console → 搜尋 HealthLake → 點 Create datastore
2. 設定：
  - Datastore name：例如 `LongTermCareEvents`
  - FHIR version：選擇 `R4`
  - Permissions：自動建立一組 IAM role，或指定已有的 role，該 role 需有 `healthlake:UploadResources` 權限。
3. 點 Create，等待狀態變成 Active。

## 2. 建立 IAM Role ( 如需要 )

若你選擇自行指定 role，請確保它具有以下 Policy：

```
json 複製編輯 { "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "healthlake:UploadResources", "healthlake:StartFHIRImportJob", "healthlake:DescribeFHIRImportJob" ], "Resource": "*" } ] }
```

並把這個 role 指定給 HealthLake datastore。

## 3. 撰寫 Lambda 或 EC2 上的 Python 程式 ( boto3 )

在你的後端程式 ( 例如 `record_event.py` ) 中，呼叫 HealthLake 的 `upload_resources` API，將事件封裝為 FHIR `Observation` 資源：

```
python 複製編輯 import boto3 import json import time # 1. 初始化 client h1 = boto3.client('healthlake', region_name='us-west-2') DATASTORE_ID = 'LongTermCareEvents' # Create 時顯示的 ID def record_fall_event(patient_id: str, confidence: float, timestamp_ms: int): # 2. 構造 FHIR Observation obs = { "resourceType": "Observation", "status": "final", "category": [{ "coding": [{ "system": "http://terminology.hl7.org/CodeSystem/observation-category", "code": "survey" }] }], "code": { "coding": [{ "system": "http://loinc.org", "code": "92712-5", "display": "Fall event" }], "text": "偵測到跌倒事件" }, "subject": { "reference": f"Patient/{patient_id}" }, "effectiveDateTime": time.strftime('%Y-%m-%dT%H:%M:%SZ', time.gmtime(timestamp_ms/1000)), "valueQuantity": { "value": confidence, "unit": "score", "system": "http://unitsofmeasure.org", "code": "{score}" } } # 3. 上傳資源 response = h1.upload_resources( datastoreId=DATASTORE_ID, payload=json.dumps([obs]).encode('utf-8') ) print("Upload response:", response) # 範例呼叫 if __name__ == "__main__": record_fall_event(patient_id="12345", confidence=0.85, timestamp_ms=int(time.time()*1000))
```

- `upload_resources` 可一次上傳多筆資源，payload 必須是 UTF-8 bytes 的 JSON list。
- 回傳值中含有每個資源的上傳結果，若有失敗可重試或寫入 Dead Letter。

## 4. 使用 AWS CLI

若你偏好用 CLI，自動化腳本也能這麼做：

## 1. 先將 Observation JSON 存成檔案 `obs.json` :

```
json 複製編輯 [{ "resourceType": "Observation", "status": "final", "category": [{ "coding": [{ "system": "http://terminology.hl7.org/CodeSystem/observation-category", "code": "survey" }] }], "code": { "coding": [{ "system": "http://loinc.org", "code": "92712-5", "display": "Fall event" }], "text": "偵測到跌倒事件" }, "subject": { "reference": "Patient/12345" }, "effectiveDateTime": "2025-04-24T14:32:00Z", "valueQuantity": { "value": 0.85, "unit": "score", "system": "http://unitsofmeasure.org", "code": "{score}" } } ]
```

## 2. 上傳 :

```
bash 複製編輯 aws healthlake upload-resources \ --datastore-id LongTermCareEvents \ --payload fileb://obs.json \ --region us-west-2
```

## 5. 驗證與查詢

- 上傳後，進入 **HealthLake Console** → 你的 Datastore → 點 **FHIR Search**

