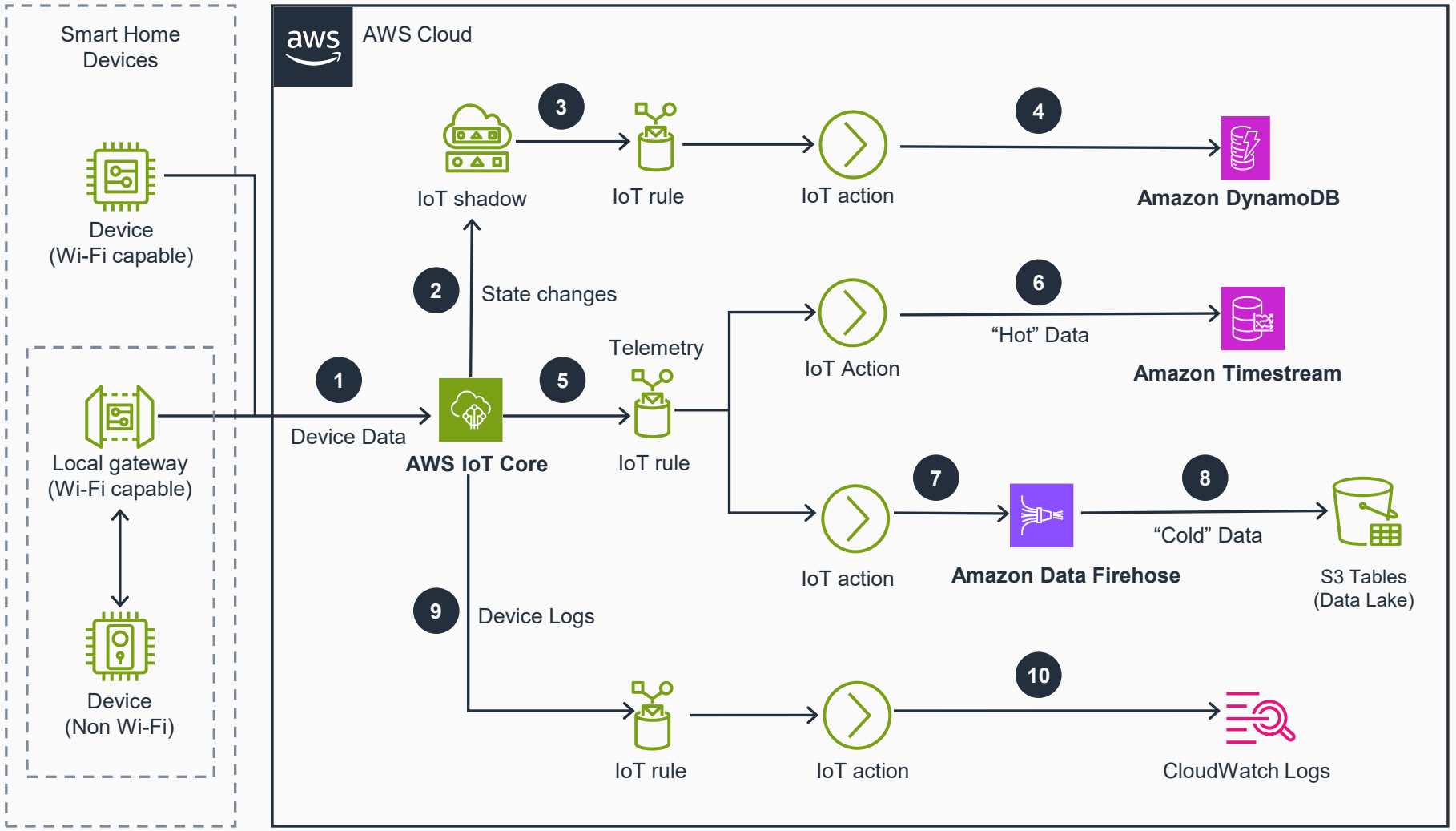
Guidance for Building Smart Home Solutions on AWS IoT

1- Data Ingestion

Smart Home devices generate various types of data (telemetry, alerts, command responses) to be consumed by different categories of users. This diagram illustrates how to build robust data ingestion pipelines with AWS IoT Core as message broker.



1. Smart Home devices publish data to **AWS IoT Core** using MQTT protocol (directly or through a local Gateway). Data can be of 3 kinds: State changes, Telemetry or Device logs.
2. State changes are typically reported through Shadow service (which stores latest desired and reported states).
3. [Optional] Reported state changes are captured by Rules Engine through a dedicated shadow topic filter.
4. [Optional] The state changes captured are routed to **DynamoDB** to record state changes history.
5. Telemetry data is captured on regular MQTT topics by Rules Engine.
6. The first rule action saves telemetry data to : **Amazon Timestream** to enable near-real-time monitoring use-cases**.** TTL (Time-To-Live) value set on Timestream tables determines how long data is stored before being discarded.
7. Second rule action sends telemetry data to **Amazon Data Firehose** for buffering before delivery to Data Lake.
8. After either a fixed time interval has elapsed or the buffer is full, buffered data is delivered to an S3 Table (managed Apache Iceberg table) for long-term storage.
9. Logs published by device on a dedicated topic are processed by Rules Engine.
10. By leveraging the **CloudWatch** Rule action, logs can be streamed in real time to a designated CloudWatch Log Group, enabling immediate visibility and operational access for CloudOps teams.

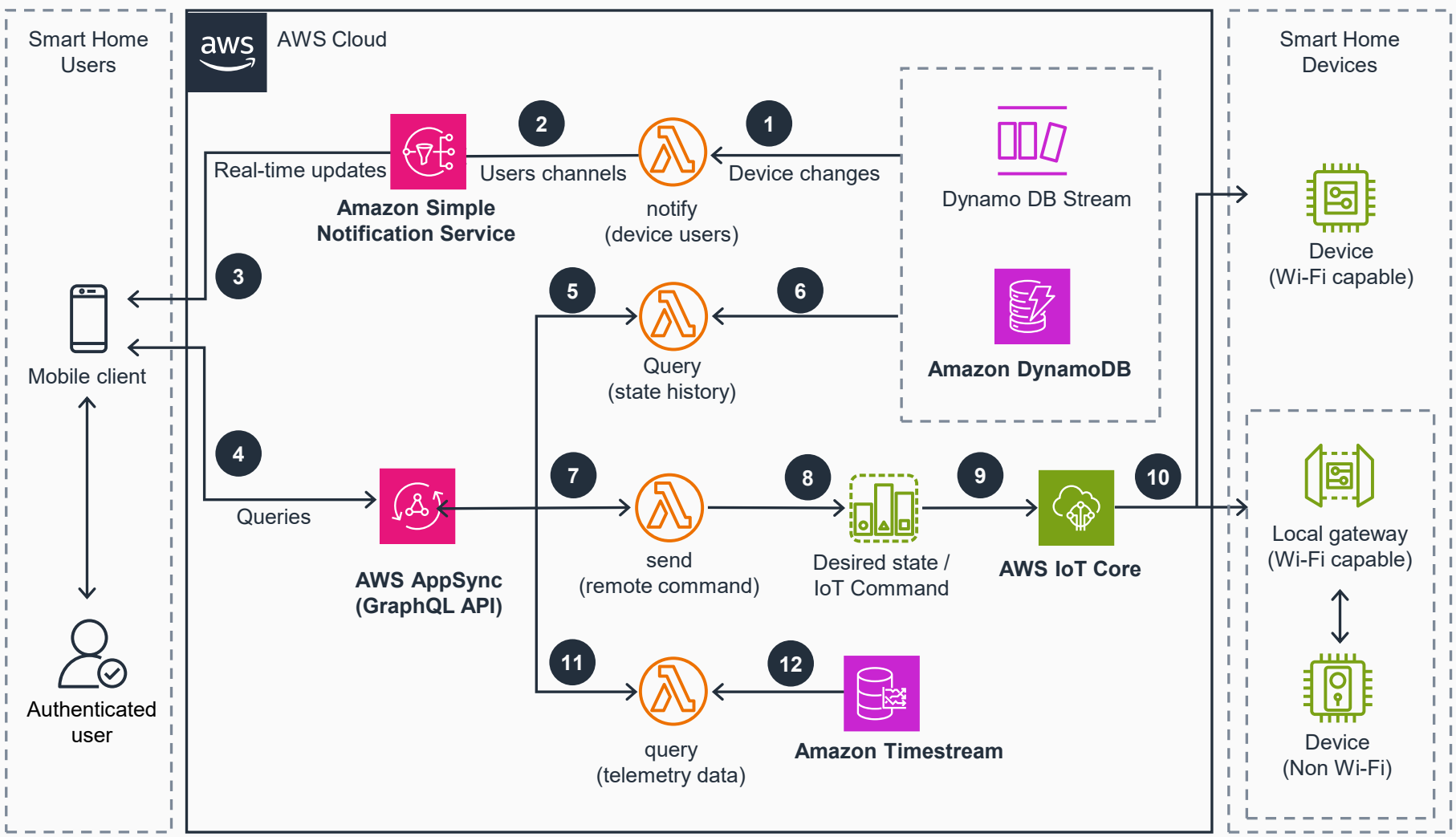
Reviewed for technical accuracy November 13, 2025

AWS Reference Architecture

Guidance for Building Smart Home Solutions on AWS IoT

2- Remote Command & Control

Smart Homes allow users to control and monitor their devices even when outside of the home premises. This diagram illustrates how to build these capabilities at scale by bundling AWS IoT suite with other AWS managed services.



1. When a new device state change is recorded in **DynamoDB**, DynamoDB stream triggers a Lambda function to notify all device users.
2. The notification Lambda function retrieves the list of authorized device users from the device table (not represented) and publishes to each user channel on **Amazon Simple Notification Service** (**Amazon SNS**).
3. Connected users that subscribed to their respective Amazon SNS (through Android or IoS) receive the update in real-time.
4. User can also initiate queries through a GraphQL endpoint managed by AWS AppSync.
5. Device State history queries are resolved by a dedicated **Lambda function**.
6. The state history Lambda retrieves the device history from Dynamo DB**.**
7. Remote commands are processed by a dedicated lambda function.
8. Lambda function leverages either Device shadow updates or IoT Commands to deliver user command.
9. **Device Shadow / IoT Commands** services use AWS IoT Core reserved topics to communicate with end-devices.
10. AWS IoT Core sends MQTT messages to the device or its gateway, based on the connectivity model.
11. Telemetry data queries (e.g., visualization) are resolved by a dedicated lambda.
12. Telemetry lambda function retrieves ‘hot data’ directly from **Amazon Timestream** (enabling near-real-time use cases).

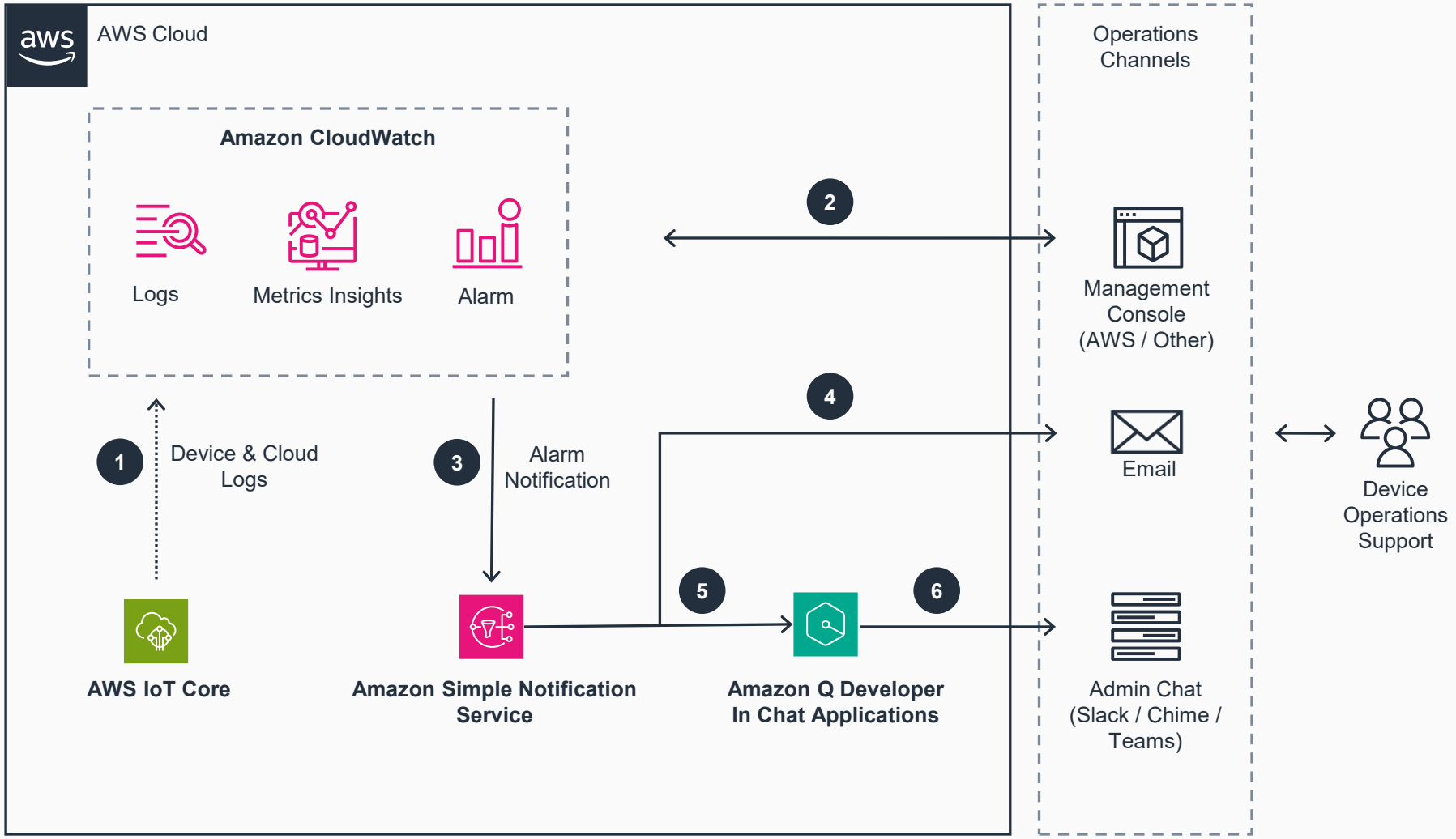
Reviewed for technical accuracy November 13, 2025

AWS Reference Architecture

Guidance for Building Smart Home Solutions on AWS IoT

3- Fleet Monitoring

A successful Smart Home solution requires continuous monitoring of devices operations. This diagram illustrates how to enforce operational excellence at scale for large smart home devices fleets.



1. **AWS IoT Core** sends both device-side and cloud-side logs to **Amazon CloudWatch**. Cloud-side logs are sent to AWSIoTLogsV2 Log group,
   Device-side logs are published by device on custom-defined MQTT topic, and forwarded to a custom-defined CloudWatch Log group by AWS IoT Core Rules Engine

2. Device operations support team interacts with CloudWatch service (through Console, Command-line, …).
   This includes enabling Anomaly detection on previous Log Groups. The anomaly model detect common maintenance event patterns in Log streams (like the presence of ERROR / WARNING / CRITICAL tags).
   Upon enabling, an “AnomalyCount” metric is created. An alarm can then be defined on the metric (a threshold set to a given number of anomalies detected).

3. When the alarms triggers, CloudWatch sends a notification to a pre-configured **Amazon Simple Notification Service (SNS)** topic to alert device operations team.

4. SNS service can push direct e-mail notification to the operations team’s distribution list e-mail.

5. Amazon SNS may also deliver the message to **Amazon Q Developer in Chat Applications (prev. AWS ChatBot),** if the same SNS topic has been set on Q in Chat Applications.

6. **Amazon Q Developer in Chat Applications** publishes the notification message to the pre-configured Chat application group channel (**Slack**, **Amazon Chime** and **Microsoft Teams** are currently supported).

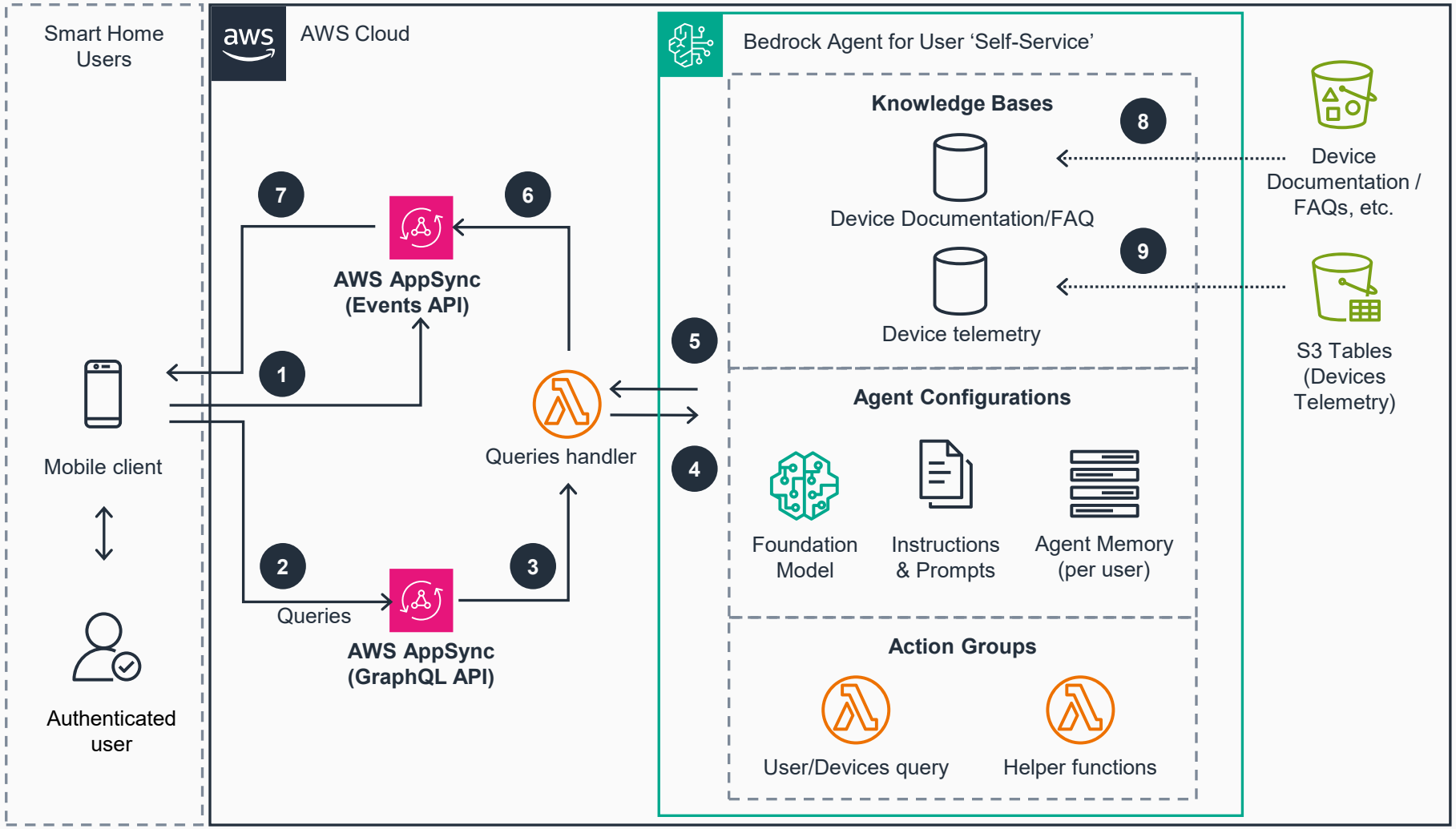
Reviewed for technical accuracy November 13, 2025

AWS Reference Architecture

Guidance for Building Smart Home Solutions on AWS IoT

4- Self-Service Diagnosis

Beyond the usual Command & Control features, Generative AI unlocks new type of use cases for Smart Home users. This diagram illustrates how AI Agents can leverage Device documentation and past activity to enable ‘self-service’ customer support



1. Smart Home user initiates a WebSocket connection from mobile client to **AppSync Events** endpoint; then subscribes to its self-service agent dedicated channel to receive future agent responses.
2. User sends its query in natural language to an AWS AppSync GraphQL endpoint.
3. The query resolves to a Lambda function "‘Queries handler”.
4. Queries handler invokes the **Amazon Bedrock** agent and initiates a session (with user Id as session Id). This allows Bedrock service to retrieve the following items and add them to the GenAI Model context:
   - The past conversations between the user and the agent (Agent Memory feature)
   - The list of devices the user has access to
5. The agent foundation model:
   - Handles the raw request by following the Agent instructions and prompts,
   - Augments the answer with Knowledge Bases (built using both Device Documentation and Device Telemetry data in S3 Tables)
   - Uses functions in Action Groups eventually to fulfil intermediate tasks for customer.
   - Generates a response to Queries handler request.
6. Queries handler publishes response to Self-Service agent dedicated channel.
7. Mobile client receives response from handler in real-time.
8. ‘Device Documentation’ knowledge base can be synced every time a new device model is released. Syncing is done through a vector store service (e.g. Amazon OpenSearch)
9. ‘Device Telemetry’ knowledge base can be refreshed periodically, to ensure Agent is grounded on most recent data. Syncing is done through a query engine (like Amazon Redshift) parsing S3 Tables

Reviewed for technical accuracy November 13, 2025

AWS Reference Architecture