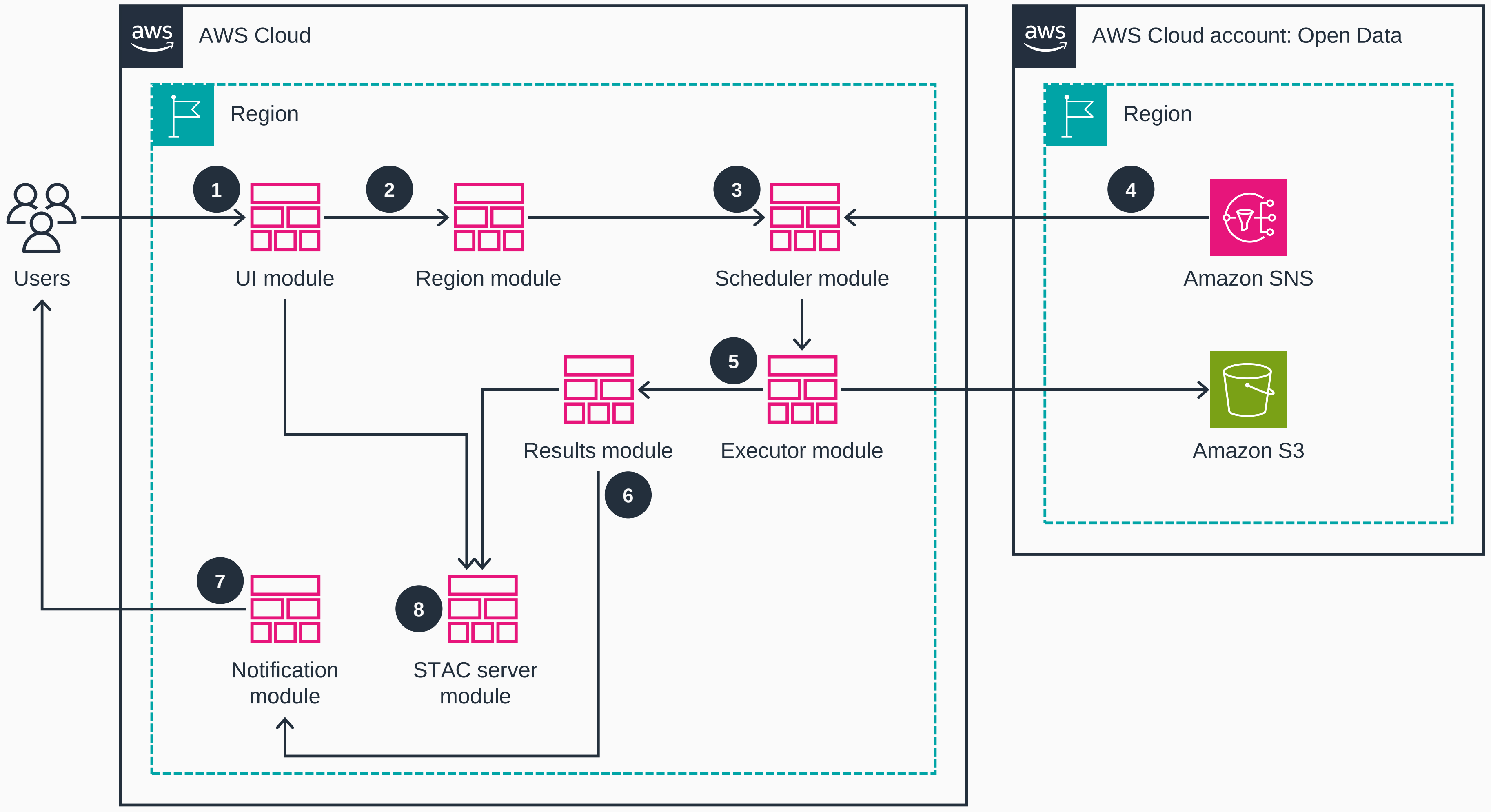
Guidance for Automated Geospatial Insights Engine on AWS

This architecture diagram shows how to automate geospatial analysis to improve supply and demand forecasting, automate risk management, and improve customer outcomes. This slide provides an overview of the architecture, showing the key modules and their interactions. Slides 2-7 detail the primary modules.



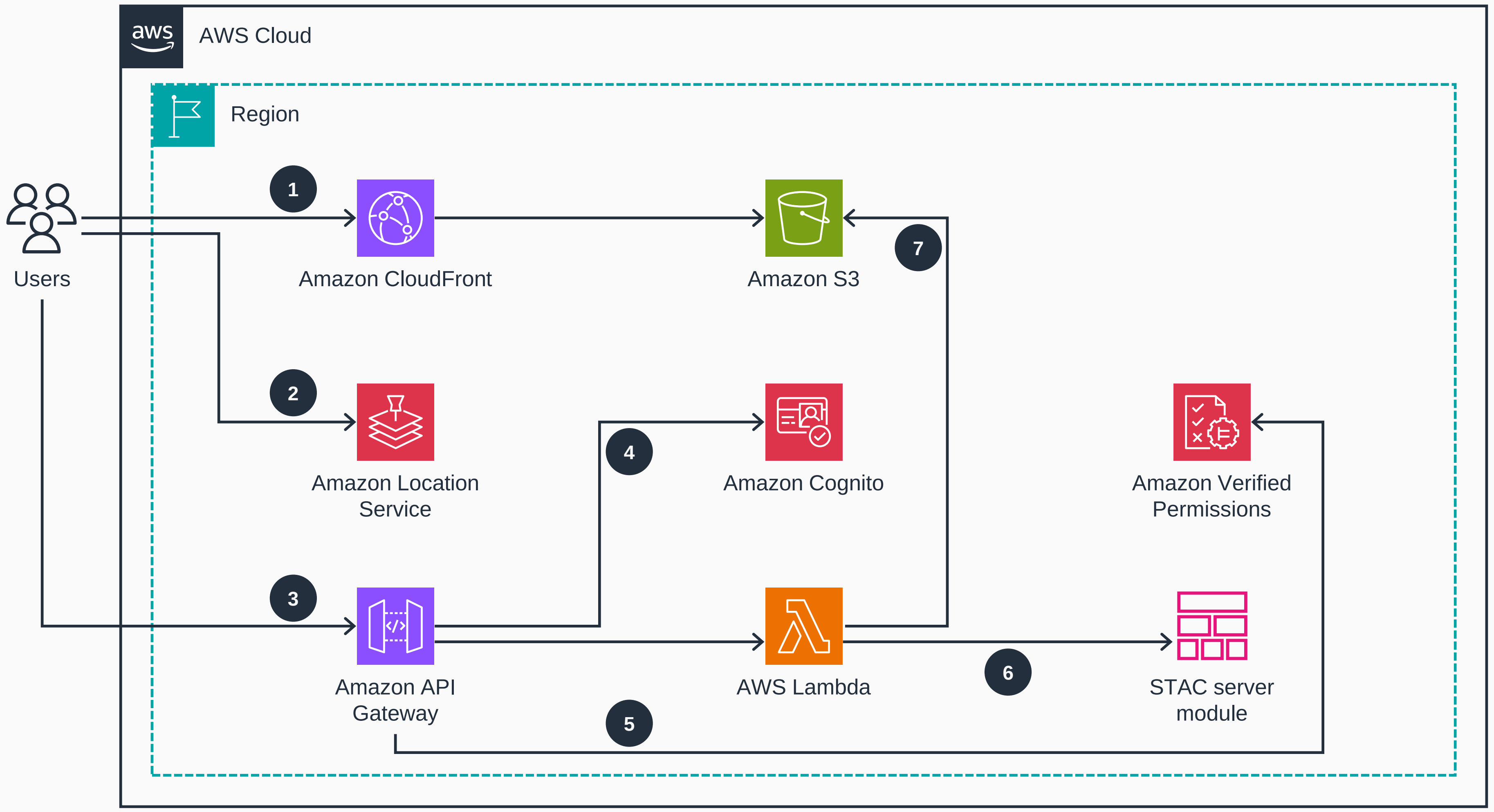
1. The *UI module* is responsible for transforming analysis results into displayable visual assets. It uses a geospatial mapping service to obtain base map data, and it renders interactive maps on the frontend using a map-rendering library.
2. The *region module* manages the hierarchical structures of groups, regions, polygons, and states.
3. The *scheduler module* manages the scheduling of engine processing tasks based on each region's processing configuration.
4. The *scheduler module* also subscribes to new satellite images using an **Amazon Simple Notification Service (Amazon SNS)** topic hosted in **Open Data on AWS**.
5. The *executor module* manages the execution of the engine analysis, invoked by the *schedule module* or by new sentinel images in **Amazon Simple Storage Service (Amazon S3)** that match the created regions.
6. The *results module* acts as an intermediary between the *region* and *executor modules*, transforming and publishing farm data to a SpatioTemporal Asset Catalog (STAC) server and mapping execution results to their corresponding farms.
7. The *notification module* manages your subscriptions to notifications generated by other modules.
8. Note: The *STAC server module* is an **AWS Cloud Development Kit (AWS CDK)** port of stac-server, an execution of the STAC API specification for searching and serving metadata for geospatial data, including but not limited to satellite imagery.

Reviewed for technical accuracy October 18, 2024

AWS Reference Architecture

UI Module

This module is responsible for providing a UI, managing authentication and authorization, and querying the STAC server for analysis results. It then converts the results into renderable assets, uses Amazon Location Service for base map data, and renders interactive maps on the frontend by using a map-rendering library like MapLibre GL JS.

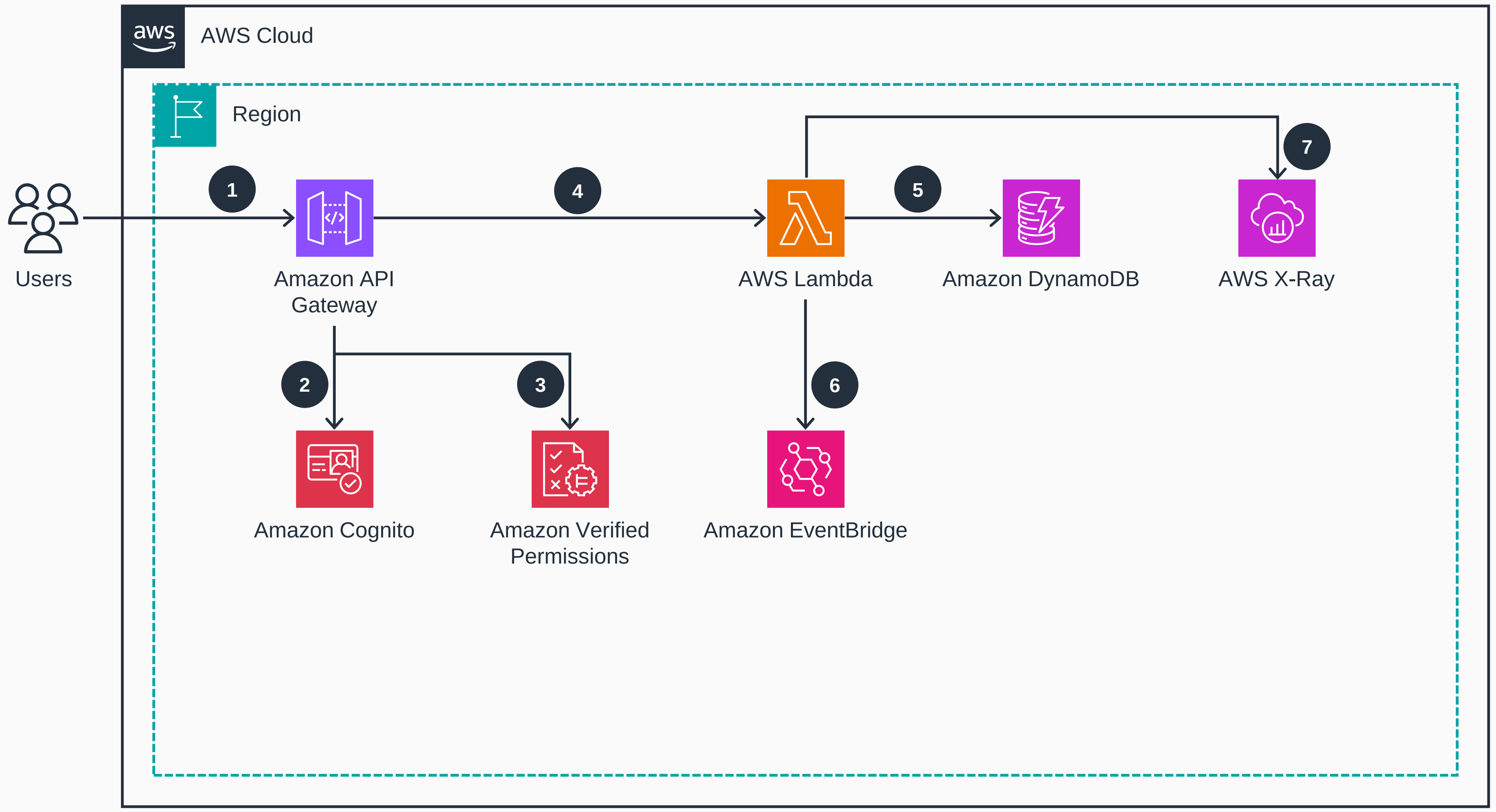


1. Load the sample React application from the **S3** bucket, using **Amazon CloudFront** to maintain low latency and high performance.
2. The React application renders maps using **Amazon Location Service**.
3. The React application overlays the base map with the processed satellite images by querying the tiler API on **Amazon API Gateway**.
4. **Amazon Cognito** delivers a token for your authentication.
5. **Amazon Verified Permissions** uses the role encoded in your token to perform fine-grained authorization.
6. Once you have been authenticated, the **AWS Lambda** tiler-API function queries the STAC server's metadata of the polygon execution result (based on the current view port of the map).
7. The **Lambda** tiler-API function generates signed URLs for all image assets specified in the STAC items and returns them to the React application for rendering.

AWS Reference Architecture

Region Module

This module manages the hierarchical structures of groups, regions, polygons, and states. It publishes events when changes occur (such as a farm being created, updated, or removed). This enables other components, like the *scheduler module*, to react accordingly.



1 Create grower, farm, and field resources using the region API through **API Gateway**. During the process of setting up a new farm, you'll be able to define the schedule and prioritization for the geospatial data processing tasks associated with it.

2 **Amazon Cognito** delivers a token for your authentication.

3 **Verified Permissions** uses the role encoded in your token to perform fine-grained authorization. Your role is stored in custom:role, a custom attribute of **Amazon Cognito**.

4 Once you have been authenticated, **API Gateway** forwards the request to **Lambda**.

5 The grower, region, and farm data is stored in an **Amazon DynamoDB** table.

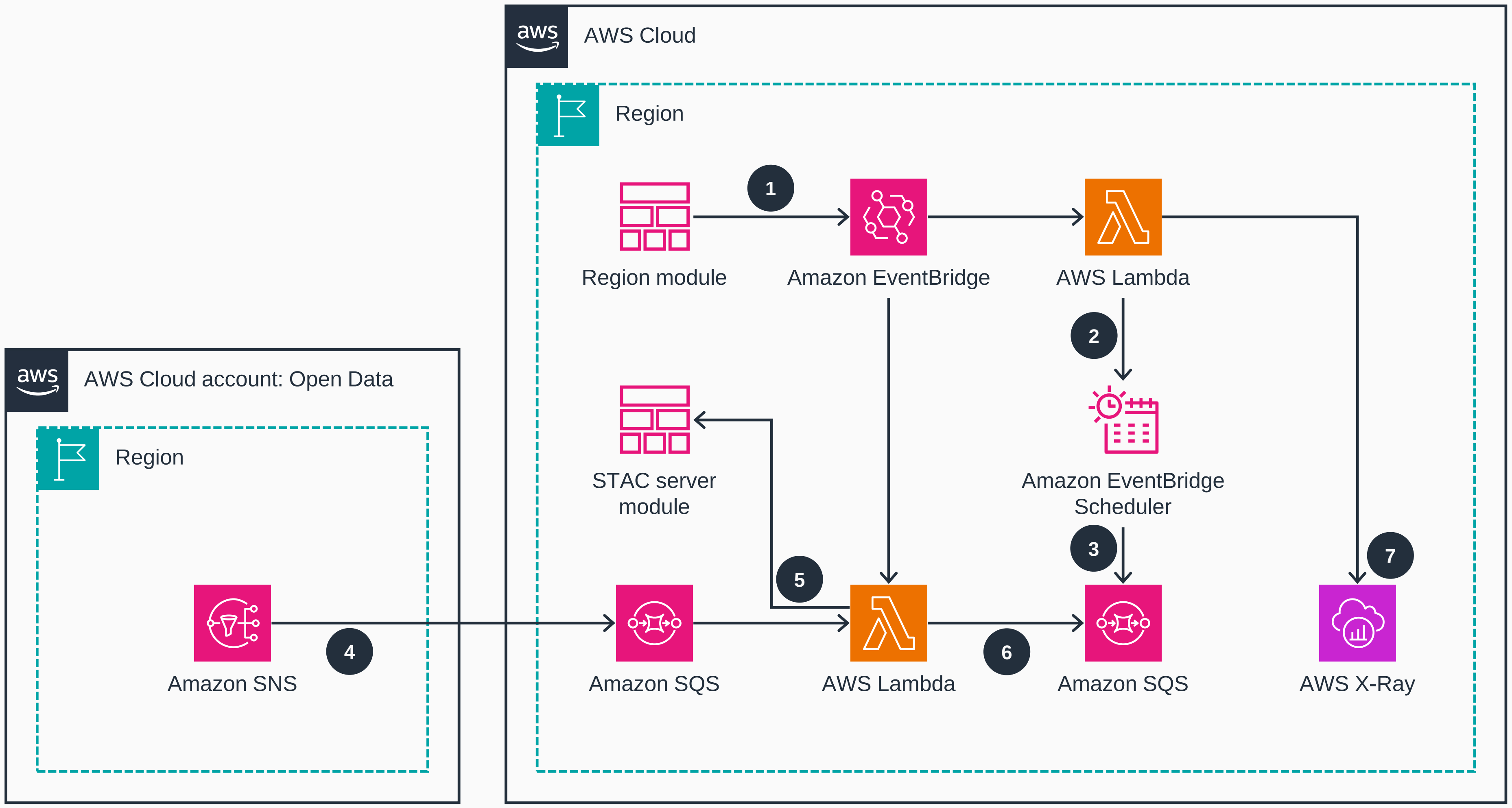
6 A region's resource changes emit events to an **Amazon EventBridge** event bus. Events can be tracked to update other components of the framework.

7 **Lambda** records a segment with details about invoking and running the function and sends it to **AWS X-Ray**.

AWS Reference Architecture

Scheduler Module

This module manages the scheduling of the engine's processing tasks, based on each region's processing configuration. This module subscribes to an Amazon SNS new-satellite-image topic hosted in Open Data on AWS.



1 The *scheduler module* subscribes to farm events published by the *region module*.

2 If a farm is configured with a recurrent schedule, it creates a processing schedule for that farm using an **EventBridge** scheduler.

3 When a schedule is invoked, the **EventBridge** scheduler puts messages into the job queue for the farm to be processed.

4 The *scheduler module* also subscribes to new-satellite-image notifications through an **Amazon SNS** topic hosted in **Open Data on AWS**. **Amazon Simple Queue Service (Amazon SQS)** queues messages.

5 When a new satellite image becomes available in the **Open Data on AWS** bucket, **Lambda** queries the *STAC server module* to check if the image includes the geographic region where the farms are located.

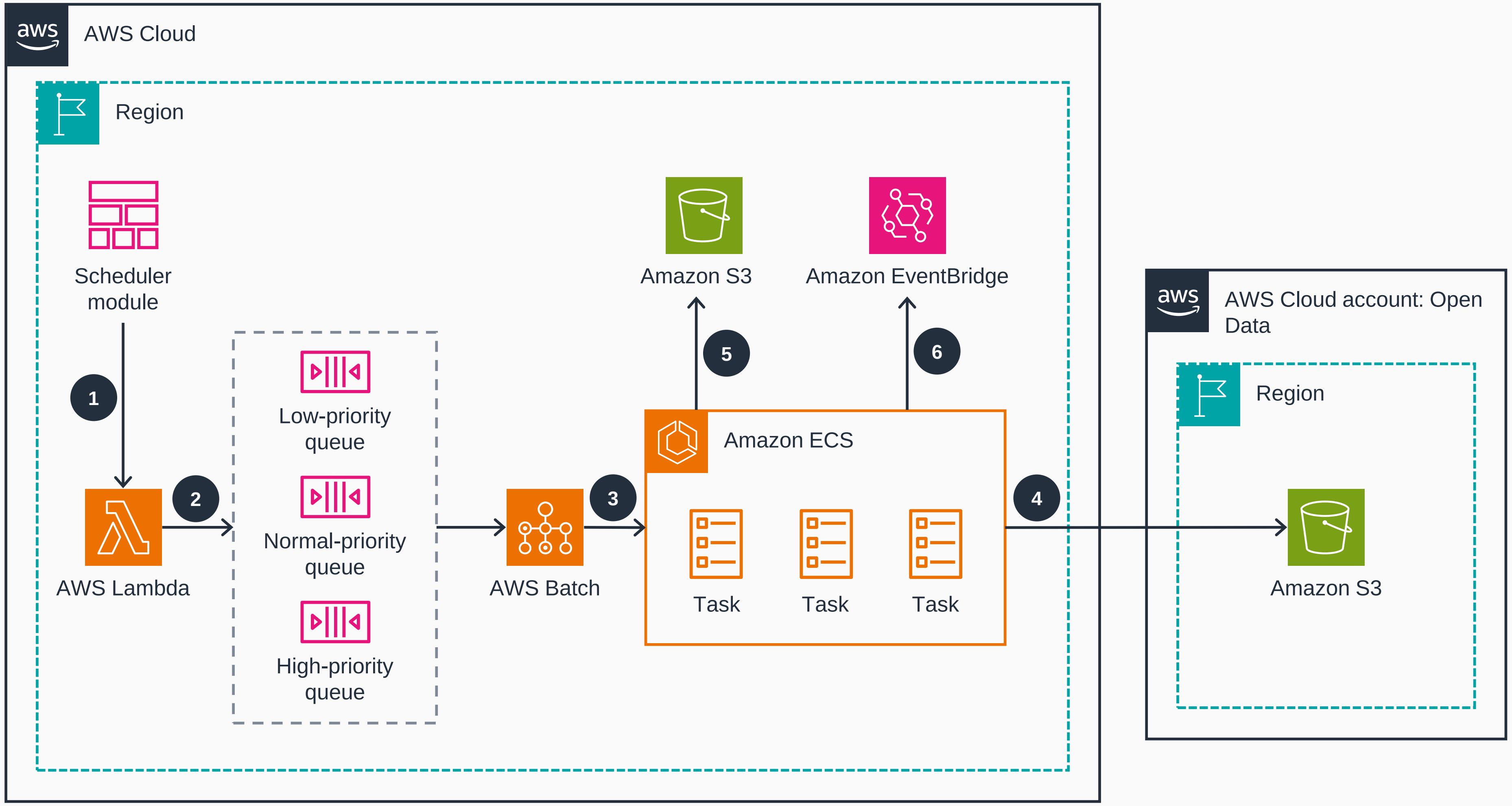
6 In the event of spatial intersection, **Lambda** puts a processing-task message in the job queue.

7 **Lambda** records a segment with details about invoking and running the function and sends it to **X-Ray**.

AWS Reference Architecture

Executor Module

This module manages the execution of the farm analysis invoked by the *schedule module* or by new sentinel images that match created regions.

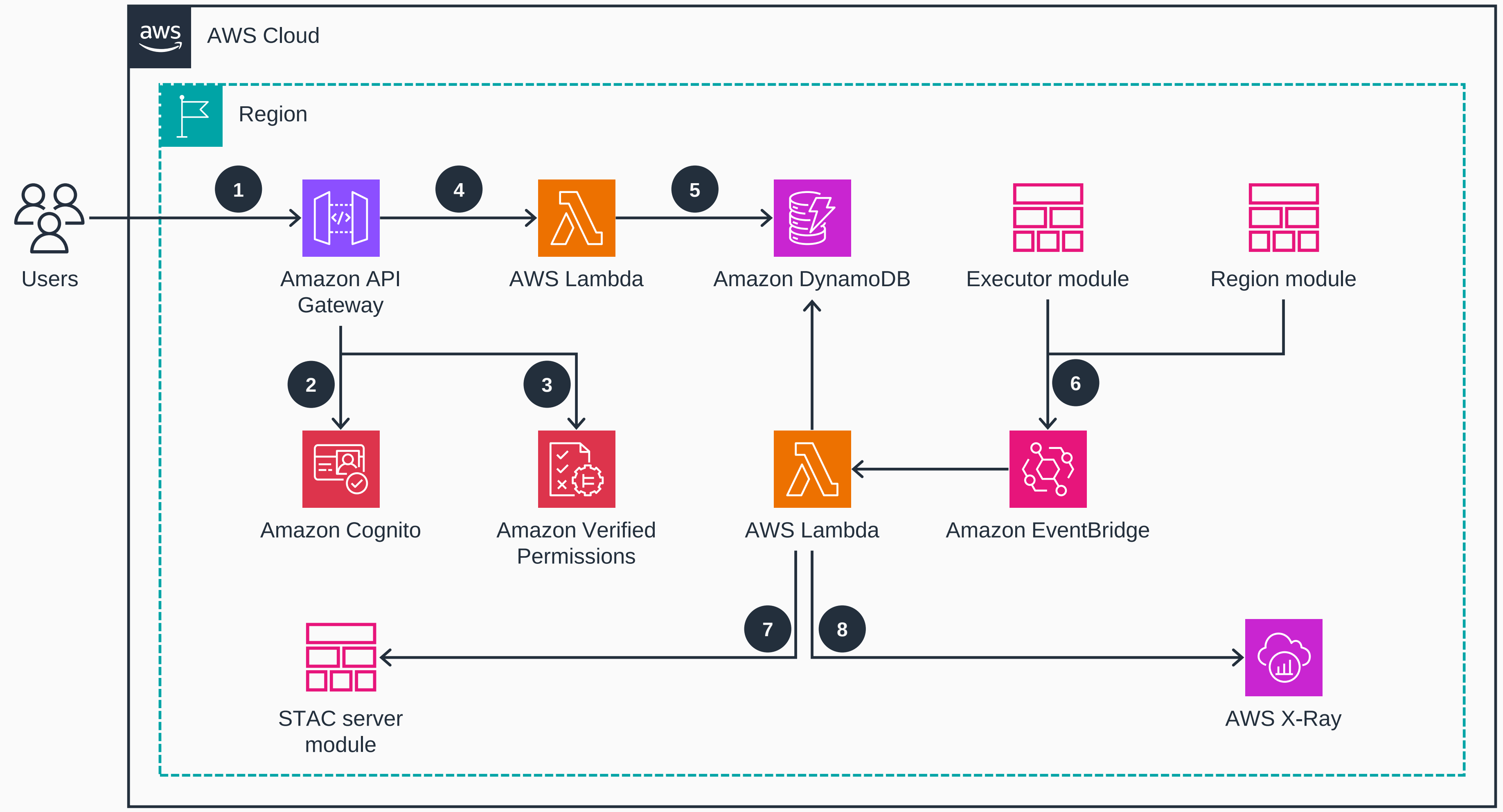


1. The *executor module* subscribes to the job queue of the *scheduler module*, which queues messages when a farm is scheduled for processing.
2. The queue invokes the *executor module* using **Lambda**, which then places the job in the right-priority **AWS Batch** queue based on the farm configuration.
3. **AWS Batch** will pick up a job from these queues and launch tasks using **AWS Fargate** for **Amazon Elastic Container Service (Amazon ECS)** to process the fields associated with the farm in parallel.
4. The **Amazon ECS** tasks retrieve the satellite images from the **S3** bucket hosted in **Open Data on AWS**.
5. These tasks then process the images and store the results in an **S3** bucket.
6. Once the execution finishes, the **Amazon ECS** tasks publish the results' metadata to an **EventBridge** event bus.

AWS Reference Architecture

Results Module

This module acts as an intermediary between the *region module* and the *executor module*. It transforms farm data into STAC items and publishes them to the STAC server, then maps the execution results to their corresponding regions. This enables efficient geospatial data management, retrieval, and access to execution results for a specific region.



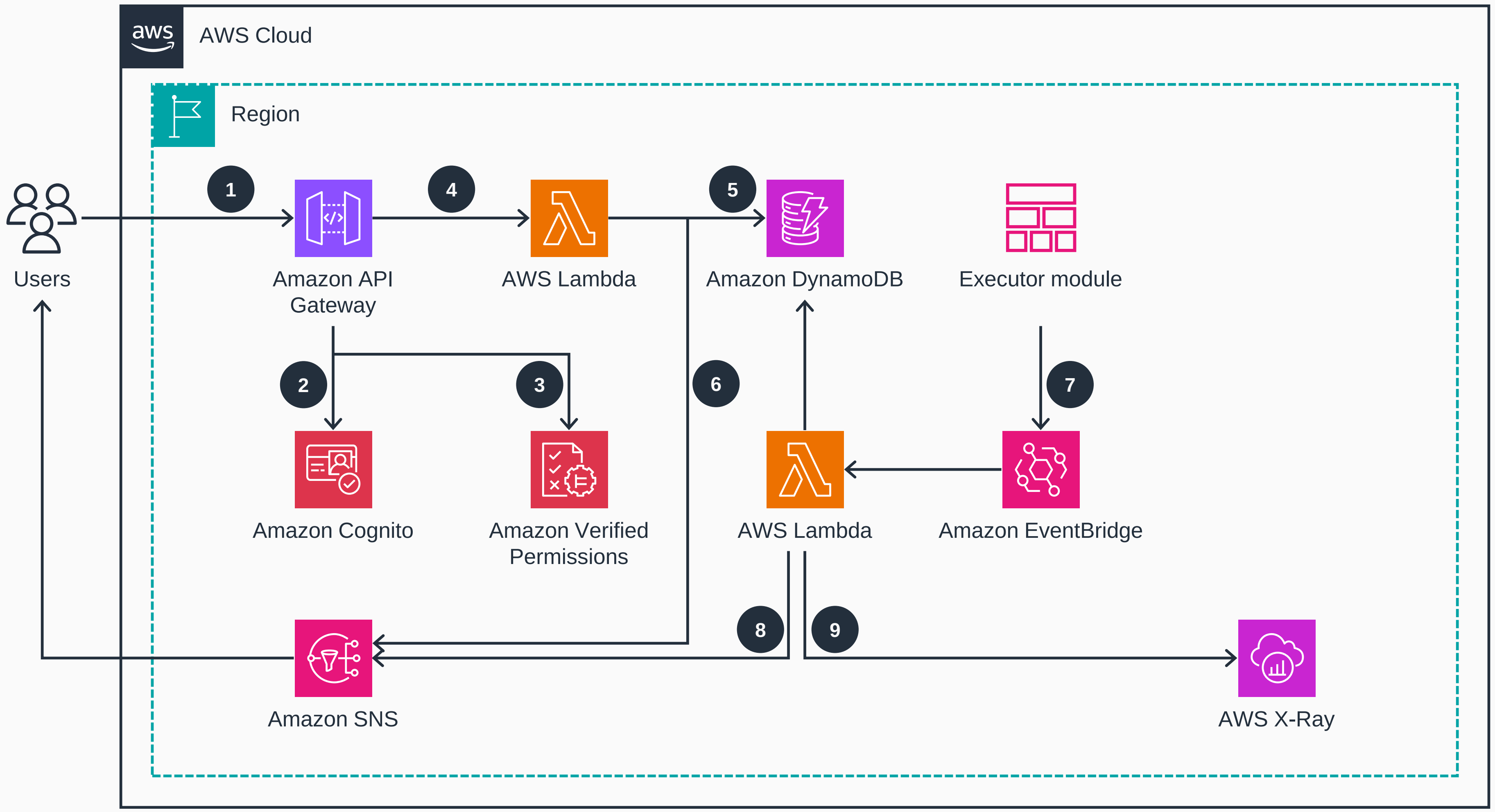
1. Interact with the *results module* through **API Gateway**.
2. **Amazon Cognito** delivers a token for your authentication.
3. **Verified Permissions** uses the role encoded in your token to perform fine-grained authorization.
4. Once you have been authenticated, **API Gateway** forwards the request to **Lambda**.
5. The data containing the execution result details—such as the unique ID, status, last updated time, and execution specifics—is stored in a **DynamoDB** table.
6. The *results module* subscribes to grower- and farm-change events from the *region module* and farm-analysis-result events from the *executor module*.
7. The *results module* transforms these events into a STAC item and publishes it to the **Amazon SNS** *STAC server module* topic.
8. **Lambda** records a segment with details about invoking and running the function and sends it to **X-Ray**.

Reviewed for technical accuracy October 18, 2024

AWS Reference Architecture

Notification Module

This module manages your subscriptions to notifications generated by other modules.



1. Create a subscription for farm processing events through **API Gateway**.
2. **Amazon Cognito** delivers a token for your authentication.
3. **Verified Permissions** uses the role encoded in your token to perform fine-grained authorization.
4. Once you have been authenticated, **API Gateway** forwards the request to **Lambda**.
5. Your list of subscription data is stored in a **DynamoDB** table.
6. **Lambda** subscribes you to an **Amazon SNS** farm topic. (It will create the topic if it does not already exist.)
7. **Lambda** subscribes you to an **EventBridge** event bus for farm processing events published by the *execution module*.
8. **Lambda** transforms the events into notification messages and publishes them to you (or other subscribed users) through **Amazon SNS**.
9. **Lambda** records a segment with details about invoking and running the function and sends it to **X-Ray**.

AWS Reference Architecture