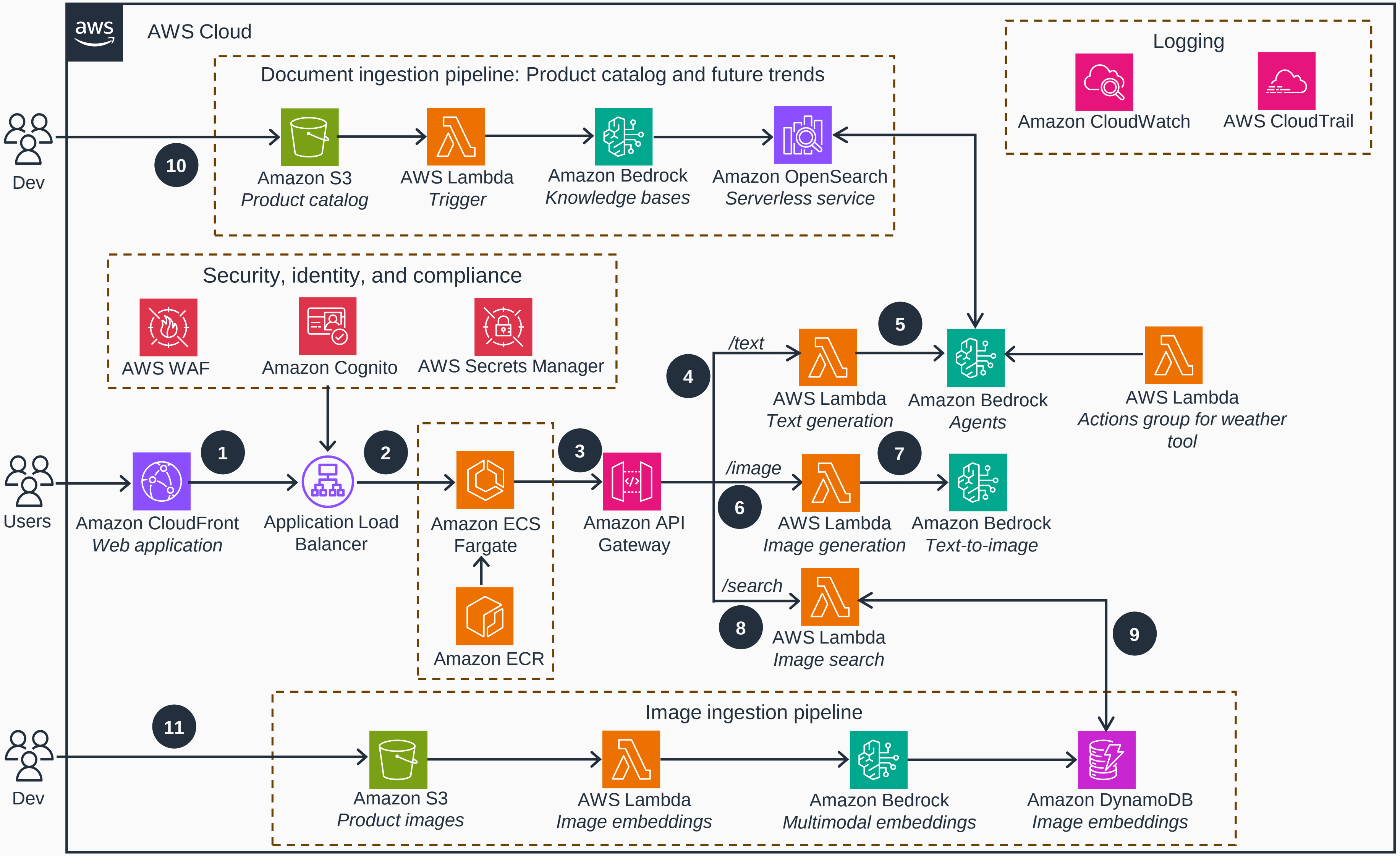
Guidance for a Virtual Personal Stylist on AWS

This architecture diagram shows how retailers can build a multi-modal application that offers personalized styling and outfit recommendations to customers. There are 11 steps, with 1-8 shown here; steps 9-11 are shown on the next slide.



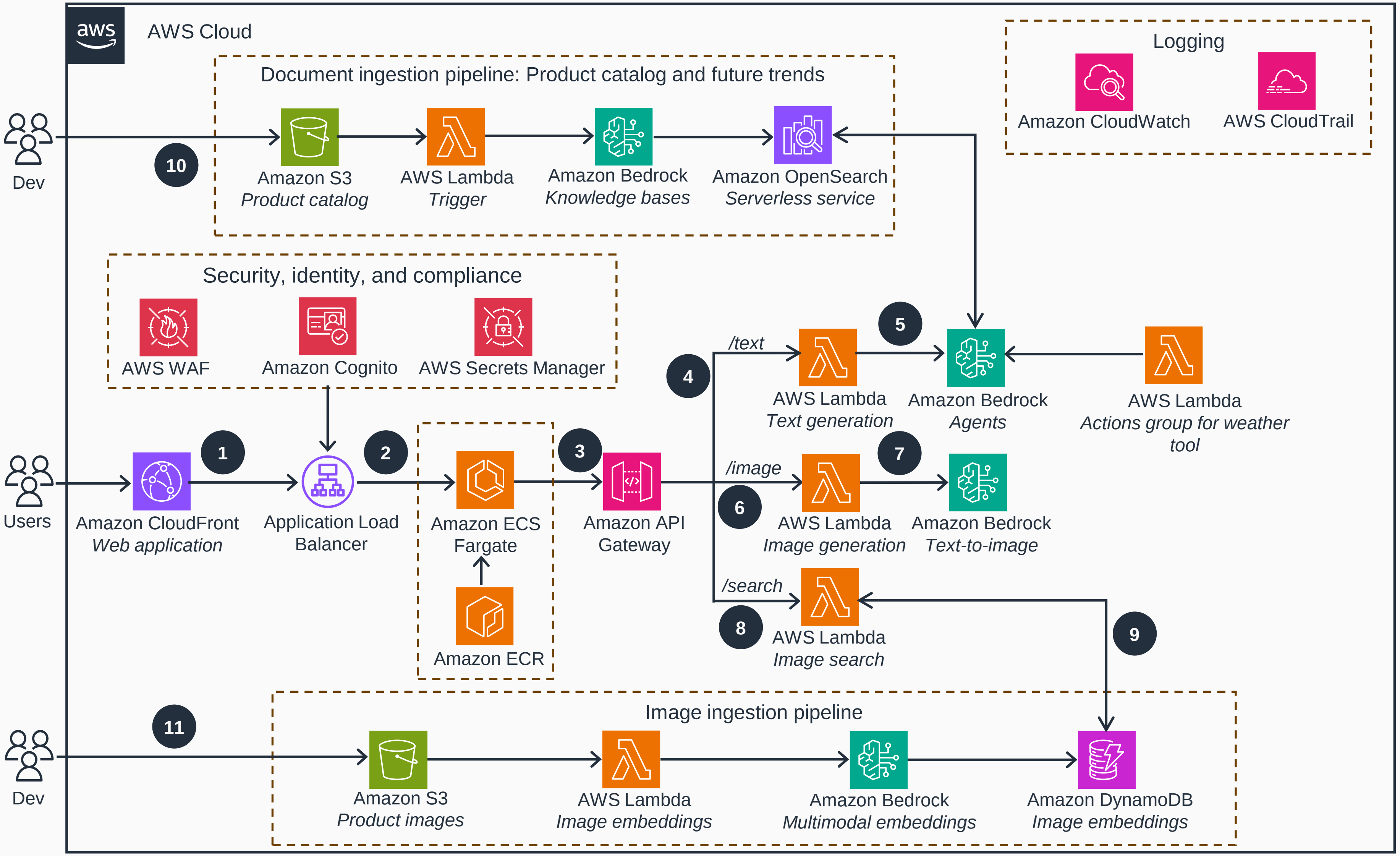
1. The application is hosted on **AWS Fargate for Amazon Elastic Container Service** (Amazon ECS), served through **Amazon CloudFront**, and coupled with **AWS WAF**.
2. The user authenticates to the application through Amazon Cognito user pools. The application retrieves an API key, URL, and **Amazon Cognito** user pool ID from **AWS Secrets Manager**.
3. The user query is sent to **Amazon API Gateway** using an API URL and key stored in **Secrets Manager**.
4. An **AWS Lambda** *Text generation* function, routed through /text, is invoked to call **Agents for Amazon Bedrock**.
5. **Agents for Bedrock** use the user's location to determine weather conditions and responds with styling recommendations. It uses data stored in **Amazon Simple Storage Service** (Amazon S3). This data is indexed in Amazon OpenSearch Serverless using Knowledge Bases for Amazon Bedrock.
6. When the user initiates a request to generate an image, the /image routing triggers a **Lambda** *Image generation* function.
7. The **Lambda** Image generation function queries the Amazon Bedrock Image generation model, which sends a Base64-encoded image back to the **Lambda** function. The image is subsequently fed back to **API Gateway**.
8. When the user initiates the product search feature, the **API Gateway** invokes the /search endpoint, which searches for image embeddings stored in **Amazon DynamoDB**. Images stored in **Amazon S3** trigger a **Lambda** function to create multi-modal image embeddings and insert them into the **DynamoDB** database.

Reviewed for technical accuracy July 31, 2024

AWS Reference Architecture

Guidance for a Virtual Personal Stylist on AWS

Steps 9-11



9 The **Lambda** function responsible for product image search creates query embeddings and uses cosine similarity to search for similar images from the **DynamoDB** database. The top three ranked images are then returned as a response from **API Gateway** and displayed to the end user within the browser application.

10 Product catalog ingestion pipeline: When the user uploads new product catalog files to an **Amazon S3** bucket, a **Lambda** event is triggered that invokes a document ingestion job on the knowledge bases for **Amazon Bedrock**. The knowledge bases for **Amazon Bedrock** then create a vector index within the vector database, specifically OpenSearch Serverless. As the data is stored and indexed in the vector database, it becomes searchable.

11 Image ingestion pipeline: When the user uploads images from their fashion catalog to an **Amazon S3** bucket, a **Lambda** event is triggered. This event invokes the Amazon Titan Multimodal Embeddings G1 model, available on Amazon Bedrock, to create image embeddings for the incoming images. The generated image embeddings are then indexed and stored in **DynamoDB**.

Note: In production-ready applications, **DynamoDB** may be replaced with **OpenSearch Serverless** for the purpose of storing image embeddings.

Reviewed for technical accuracy July 31, 2024

AWS Reference Architecture