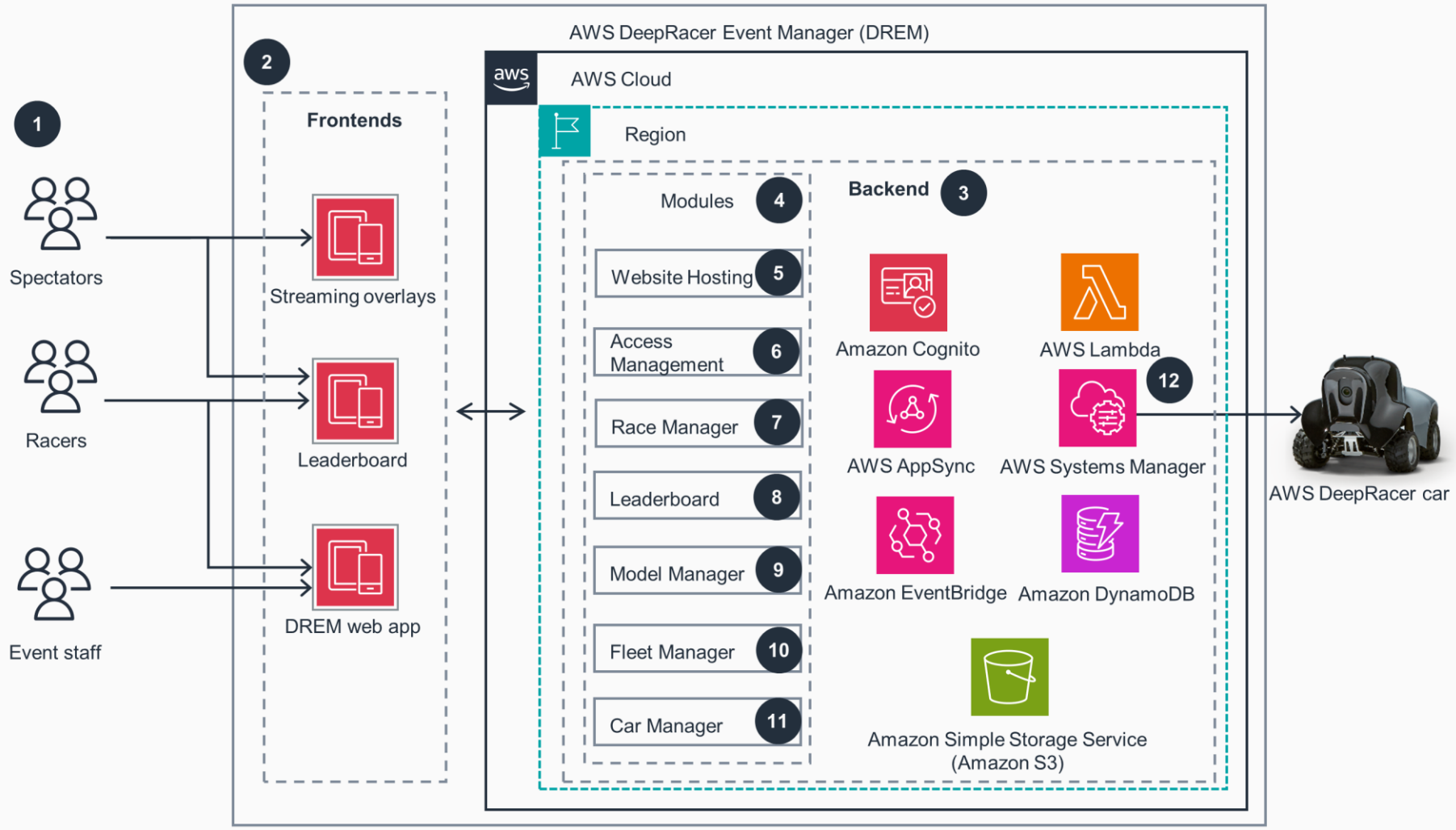
Guidance for AWS DeepRacer Event Management

This architecture diagram shows how you can automate the multiple operations that are entailed for AWS DeepRacer events. These events include website hosting, access management, race management, model management, fleet management, and car management, all captured here in subsequent slides. AWS DeepRacer is an Autonomous 1/18th scale race car, driven by ML, and designed to test reinforcement learning (RL) models when on a physical racing track.



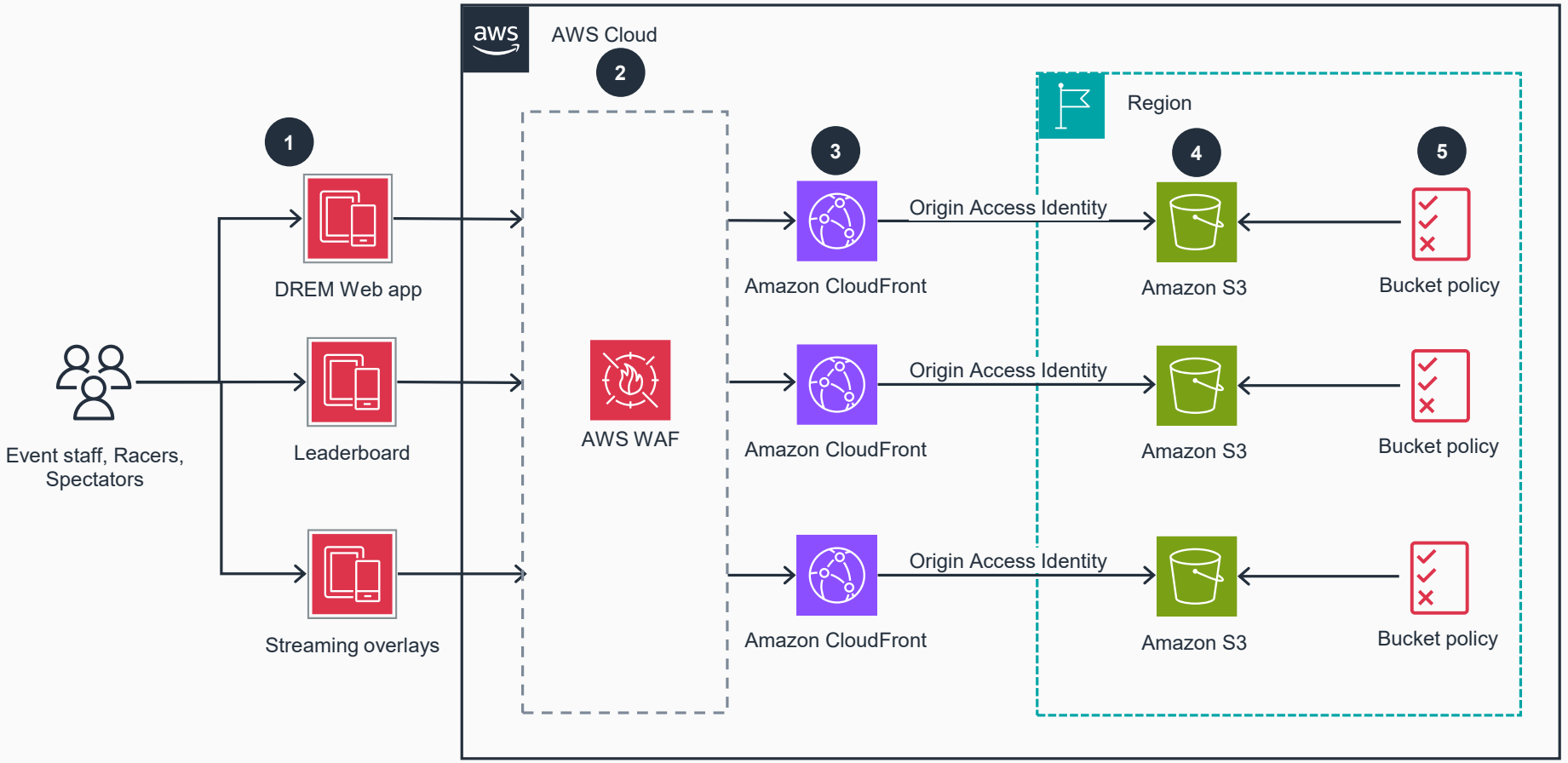
1. Event staff, racers, and spectators access the web applications hosted in AWS with the AWS DeepRacer Event Manager (DREM) through a web browser.
2. DREM consists of three different frontends built using React.
3. The backend is shared by all frontends and built using the serverless components: **AWS Lambda**, **AWS AppSync**, **Amazon DynamoDB**, and **Amazon EventBridge**.
4. The DREM backend is composed of a suite of modules, each focusing on a specific set of features. The modules own their own API definitions and AWS infrastructure.
5. Website Hosting: Hosting for the set of web-based applications that comprise up DREM.
6. Access Management: Allows registered users of DREM to be added and removed from groups, which allows them to access more privileged functions.
7. Race Manager: Records and manages data for races and events.
8. Leaderboard: Manages, displays, and publishes the race leaderboard used at DeepRacer events.
9. Model Manager: Facilitates racers uploading models to DREM and event staff pushing those models to cars ready for racing on the track.
10. Fleet Manager: Designates collections of **AWS DeepRacer** cars to simplify device management.
11. Car Manager: Activates new cars, adding them to DREM, and then manages them.
12. Each **AWS DeepRacer** car is connected to DREM through **AWS Systems Manager**.

Reviewed for technical accuracy October 25, 2023

AWS Reference Architecture

Website Hosting

DREM consists of a number of web-based applications, hosted using Amazon S3 and Amazon CloudFront, which are accessed by event staff, racers, and spectators.

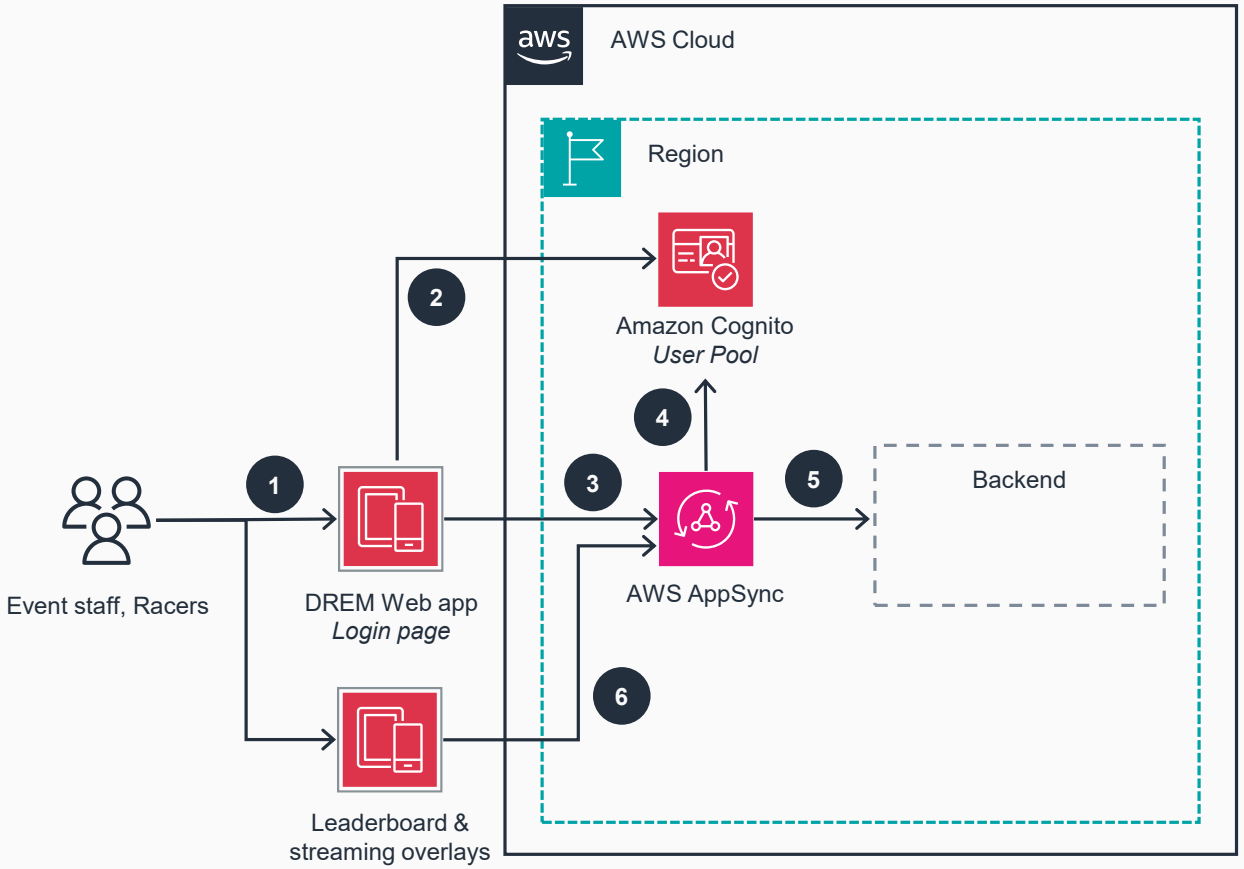


1. DREM consist of three different web apps hosted in AWS, including DREM itself, the leaderboard, and the event streaming overlays.
2. The frontends are protected by **AWS WAF** which filters malicious web traffic, helping protect the application against common exploits and bots.
3. Each app has its own **Amazon CloudFront** distribution. **CloudFront** is a global content delivery network.
4. Each app is backed by its own private **Amazon Simple Storage Service** (Amazon S3) bucket. The connection between **Amazon S3** and **CloudFront** is protected by an origin access identity (OAI).
5. Each **Amazon S3** bucket is protected by a bucket policy so the website only can be reached through the **CloudFront** distribution network.

AWS Reference Architecture

Access Management

User accounts are created and stored in Amazon Cognito. Access to model management and event functionality requires an account. Leaderboards and streaming overlays are publicly accessible.



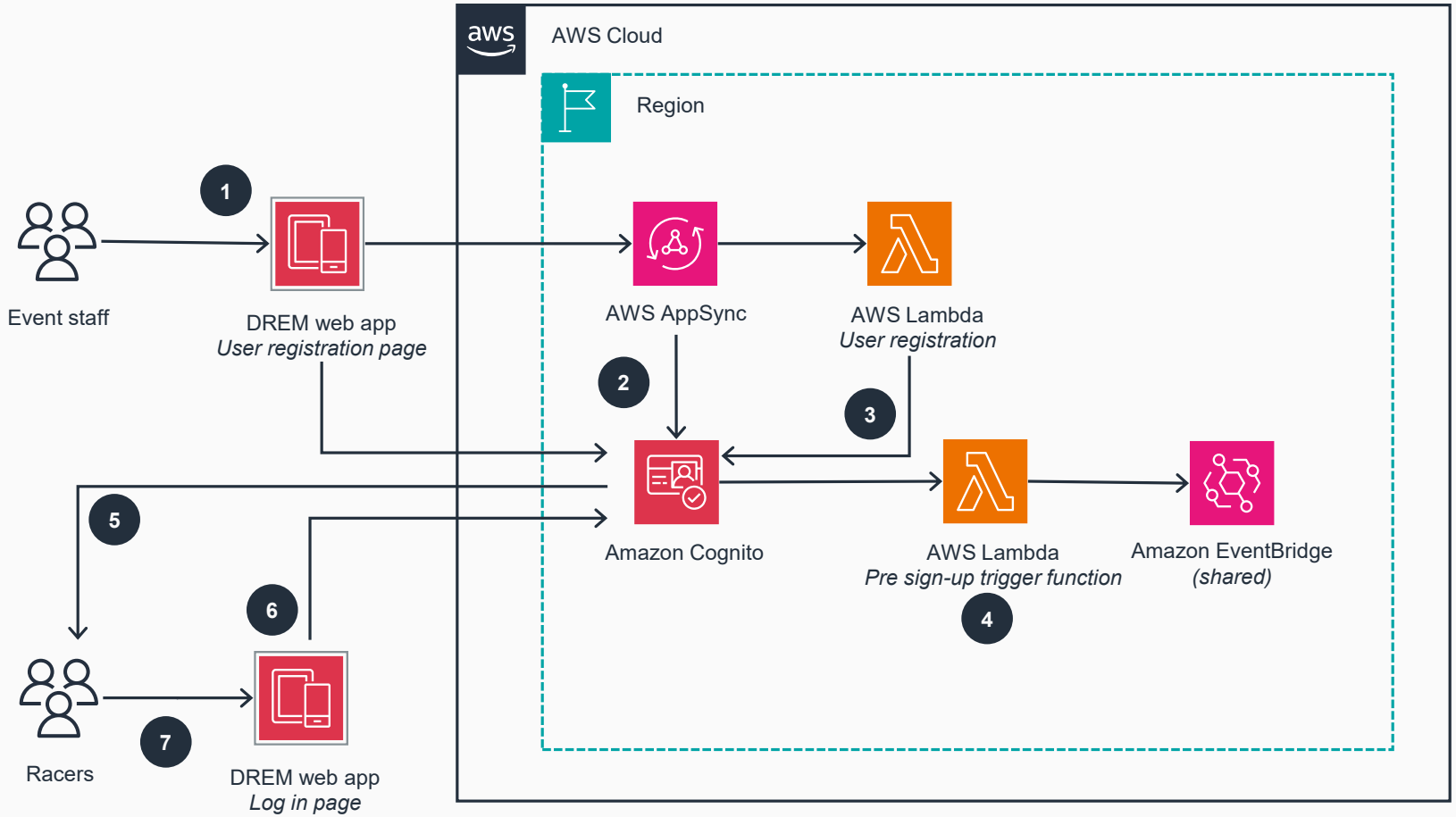
1. Users access the DREM web app URL through a browser. If the user has no active login, the user is redirected to the login page.
2. The provided user credentials are verified with **Amazon Cognito** and JSON Web Tokens (JWTs) are returned. The token data includes which **Cognito** user group the logged-in user belongs to.
3. When the DREM web app accesses the **AWS AppSync** API, the **Cognito** JWT is added to each request.
4. **Cognito** verifies the JWTs.
5. Access to the various **AWS AppSync** API endpoints is restricted by **Cognito** groups.
6. The leaderboard and streaming overlays, both open to the public, have read-only access to leaderboard information. The **AWS AppSync** access for these websites is protected using API keys.

Reviewed for technical accuracy October 25, 2023

Access Management

User Registration

DREM has two racer registration flows: Racers sign up independently or event staff members facilitate racer sign-up.



1. At larger events, walk-up participants who don't bring their own models, choose a default pre-trained model. An event staff member logs into the DREM app and browses to the user registration page to facilitate racer sign up.
2. Information collected through the registration form for each racer is submitted to the **AWS AppSync** API.
3. The **AWS AppSync** API verifies that the provided JSON Web Token (JWT) is valid and has access to invoke the AddUser API method.
4. The user registration **Lambda** function creates the user in the **Cognito** user pool if the user does not already exist.
5. Before the user is created, **Cognito** invokes a pre-signup **Lambda** trigger function which emits an **EventBridge** event notifying other modules that a new user has been created.
6. **Cognito** sends an email to the newly created user with information on how to access the DREM web app.
7. After following the instructions in the email, the newly created user can log into the DREM web app and upload or delete their own **AWS DeepRacer** models.

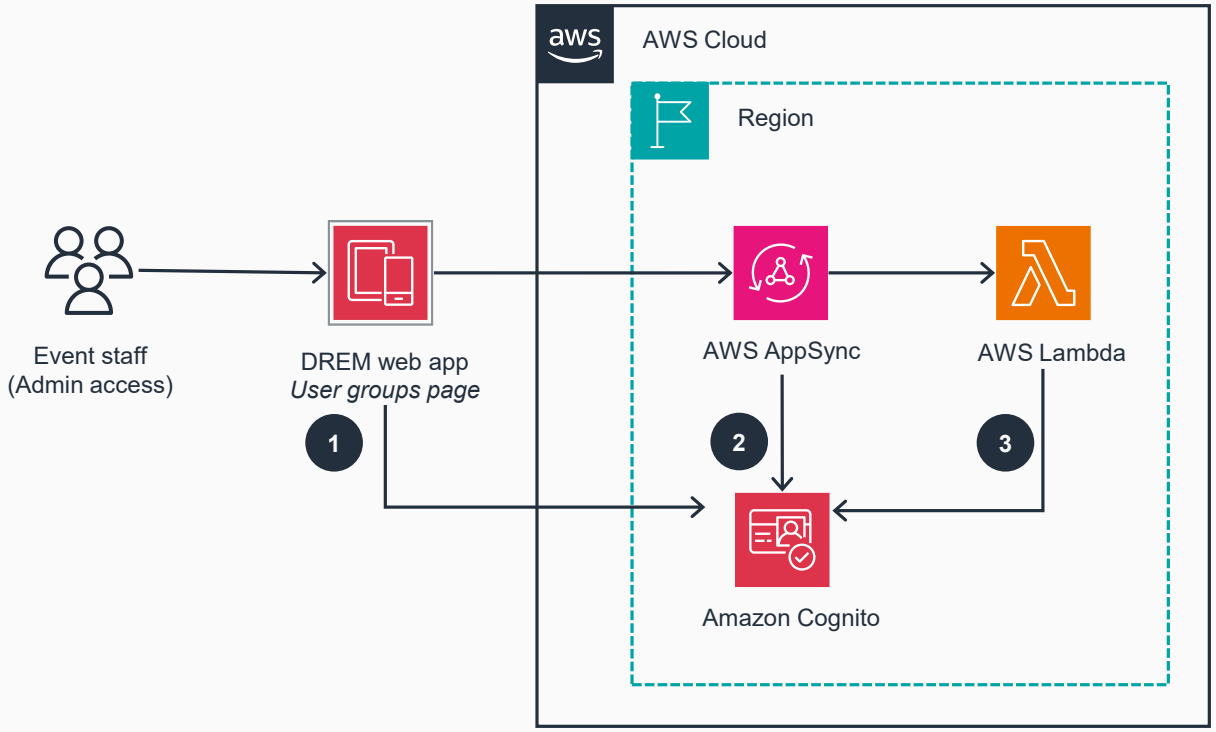
Note: A racer can also create their own user by browsing to the DREM web app and selecting signup. A user is then created in **Cognito** and the process follows Steps 5 and 6.

Reviewed for technical accuracy October 25, 2023

Access Management

Privileged Access

For an event staff member to gain privileged access to DREM, their users need to belong to certain Amazon Cognito groups. DREM comes with a default admin user which is created during deployment. This admin user can then be used to grant privileged access to other members of the event staff.



1. An event staff member with admin access can log into the DREM web app and browse to the groups page. There they can add or remove users from groups to change a user's privileges. The update request is sent to the **AWS AppSync** API.

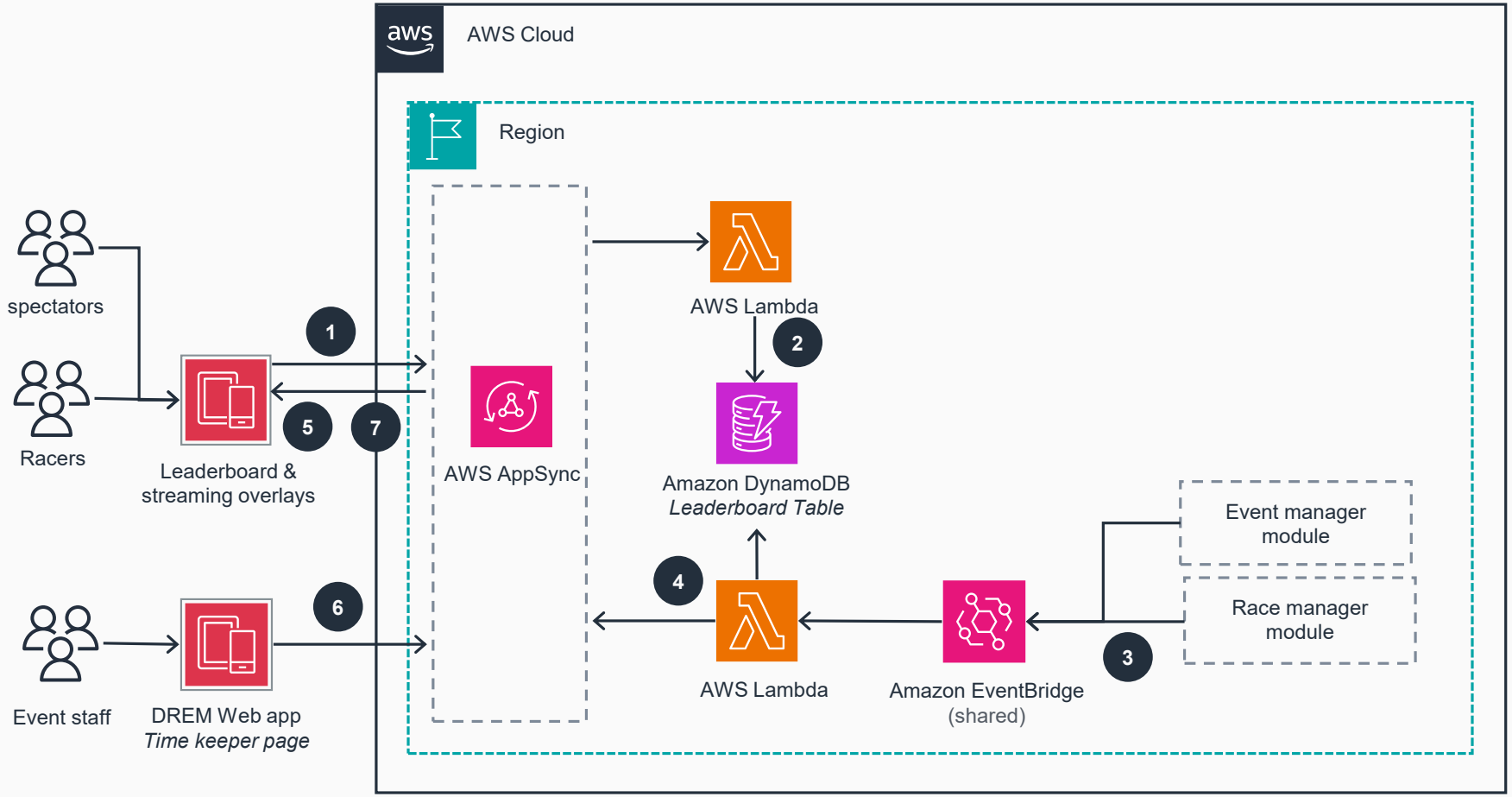
2. The **AWS AppSync** API verifies that the provided JSON Web Token (JWT) is valid and has access to invoke the *addUserToGroup*/*deleteUserFromGroup* API methods.

3. The **Lambda** resolver function calls **Cognito** and either adds the user to the group or removes them.

Race Manager & Leaderboard

Leaderboard

Leaderboards allow racers to see how they rank against other competitors and are accessible to anyone with the link for the event.



1. When the leaderboard or streaming overlays are loaded, the **AWS AppSync** API is invoked to get all current leaderboard entries for the event.
2. The **Lambda** function performs a query in **DynamoDB** for the relevant leaderboard items.
3. After each race, an event staff member submits the race to the backend, invoking an **Amazon EventBridge** event from the race manager.
4. The leaderboard entry and stats are calculated for the user by a **Lambda** function. The new leaderboard entry is stored in **DynamoDB** and a GraphQL mutation to the **AWS AppSync** API is sent.
5. The mutation invokes a subscription on the **AWS AppSync** API, pushing the newly added leaderboard entry to the connected leaderboards and streaming overlays.
6. During a race, the time keeper page in the DREM web app regularly submits GraphQL mutations to inform other parts of the system of the current status of the race.
7. The mutations invoke a subscription which the leaderboard and streaming overlay is subscribing to, thus pushing the race status info to the connected clients.

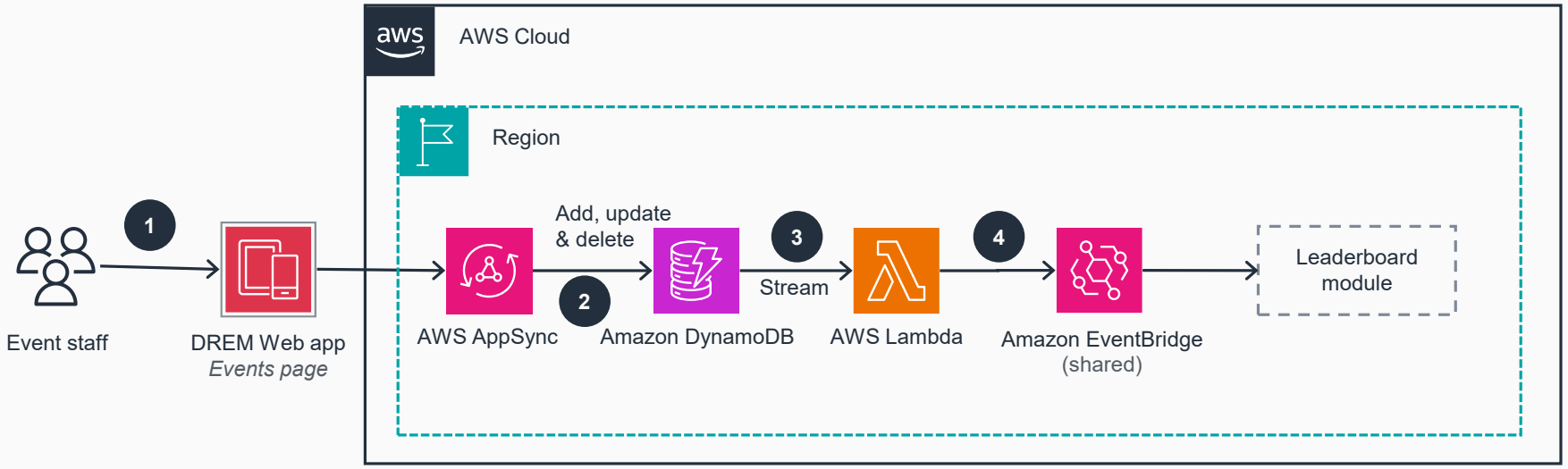
Reviewed for technical accuracy October 25, 2023

AWS Reference Architecture

Race Manager & Leaderboard

Event Manager

Events (races) are stored in Amazon DynamoDB and are managed by DREM users with operator or admin group membership.

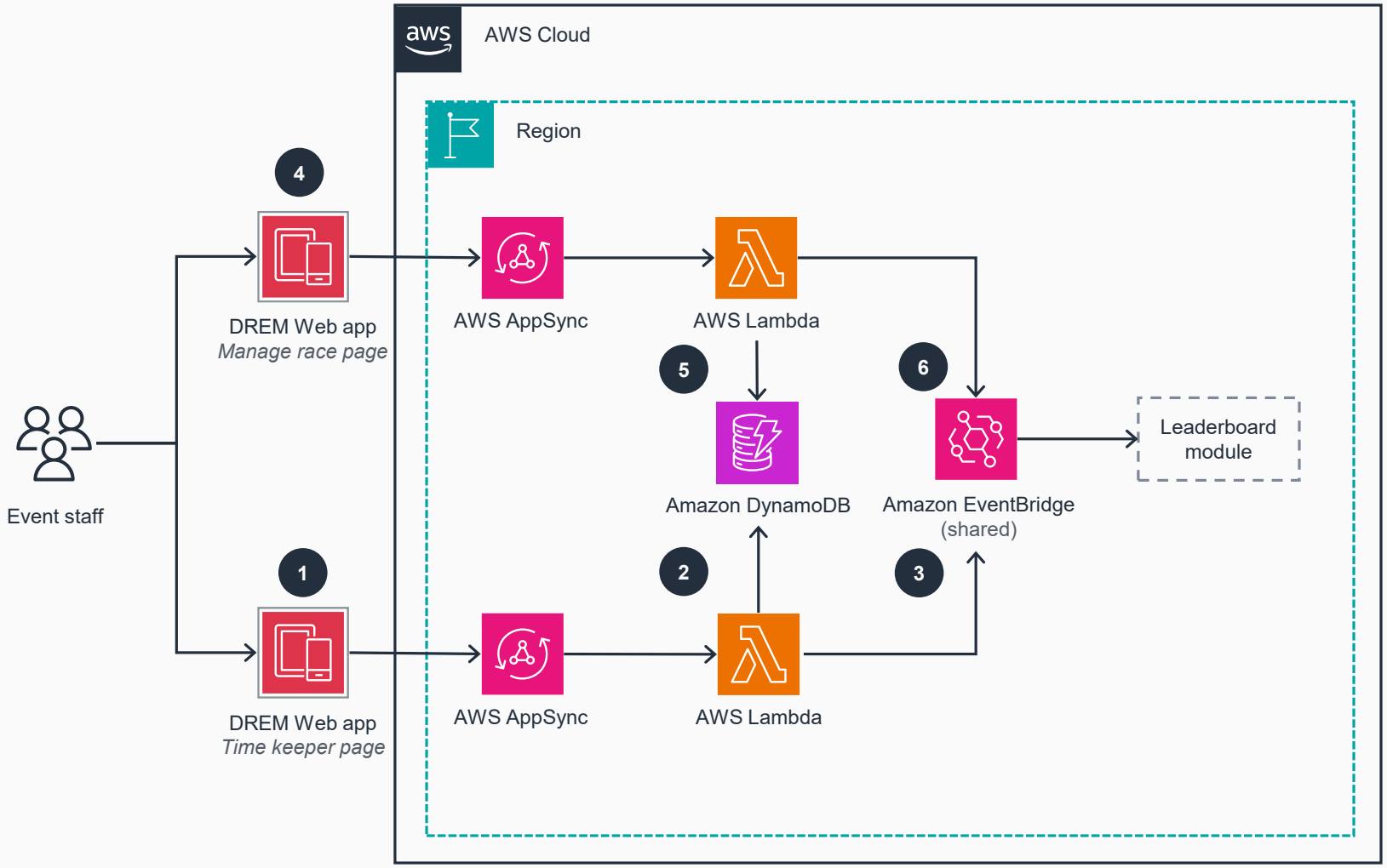


1. A member of the event staff logs into the DREM app and browses to the events page. On the events page, they can create a new event, update, or delete existing events.
2. The request is sent to the **AWS Appsync** API and the item is created, updated, or deleted in **DynamoDB.**
3. When the item is added, updated, or deleted in **DynamoDB,** the change is sent on the **DynamoDB** stream which invokes an **Lambda** function.
4. The **Lambda** function generates an event and emits it to **EventBridge** to notify the leaderboard module of the change.

Race Manager & Leaderboard

Race Manager

Lap times are recorded for events using the race manager. Times can be recorded either manually or automatically by using a Raspberry Pi and timing strips for improved timing accuracy.



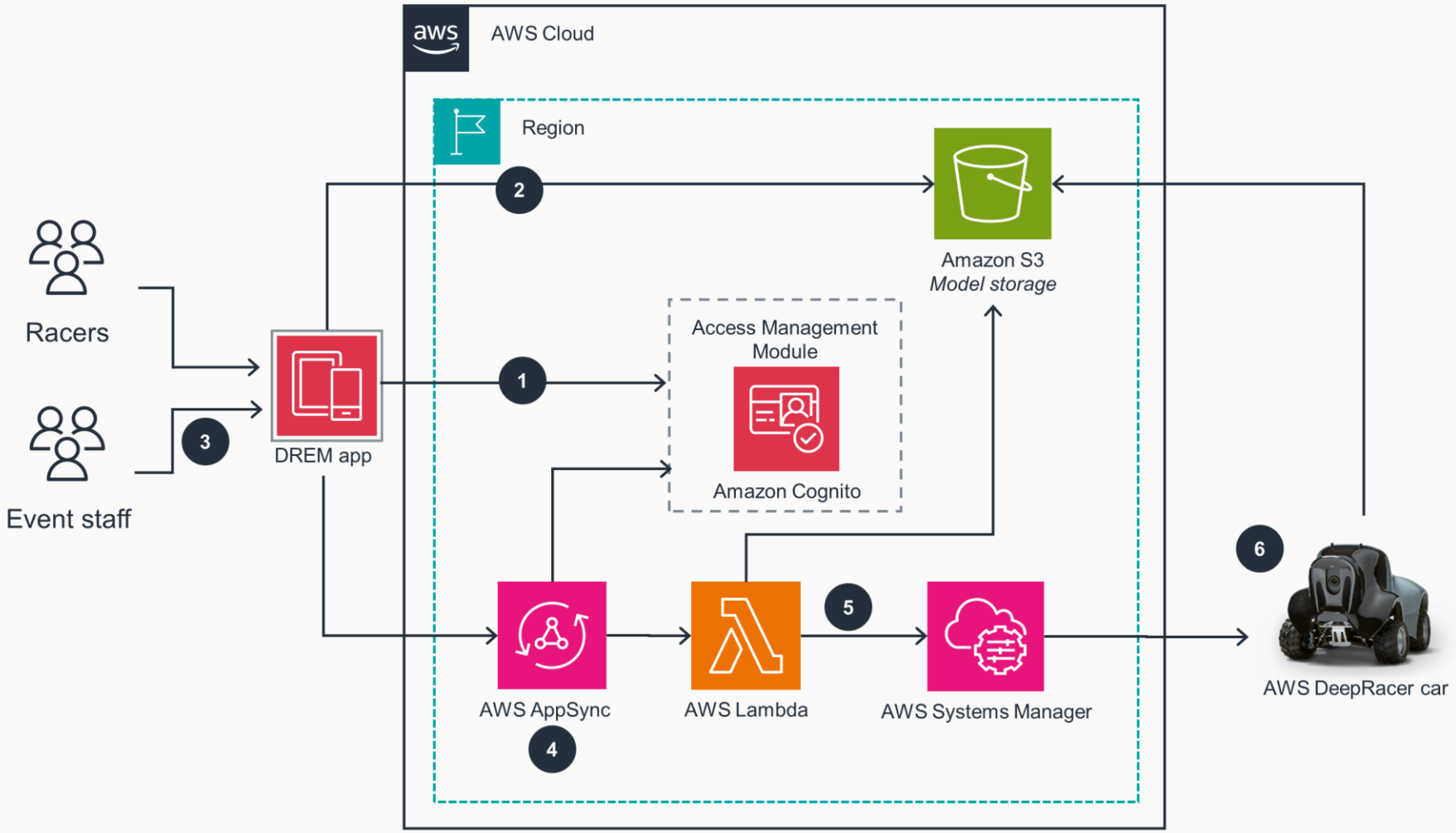
1. Timekeeping: A member of the event staff logs into the DREM app and browses to the time keeper page to select the race for which they are recording the lap times. On the time keeper page, users record the lap times and submit them to **AWS AppSync** when the race is over.
2. The request is sent to the **AWS AppSync** API and a race summary is calculated and stored in **DynamoDB** through a **Lambda** function.
3. The **Lambda** function generates a new race summary event and emits it to **EventBridge** to notify the leaderboard module of the change.
4. Race management: A member of the event staff logs into the DREM app and browses to the manage race page. On the manage race page, they can modify and delete existing races and individual lap times for a racer. Modified races are sent to **AWS AppSync.**
5. The request is sent to the **AWS Appsync** API and the item(s) are updated or deleted in **DynamoDB** through a **Lambda** function.
6. The **Lambda** function generates a new race summary event and emits it to **EventBridge** to notify the leaderboard module of the change.

Reviewed for technical accuracy October 25, 2023

AWS Reference Architecture

Model Manager

Getting models from the racers to the car is the main function of DREM. Users upload their models into an Amazon S3 bucket for storage. Event staff members can see all of the uploaded models and select specific models to send to particular cars in order to facilitate an efficient racing queue.



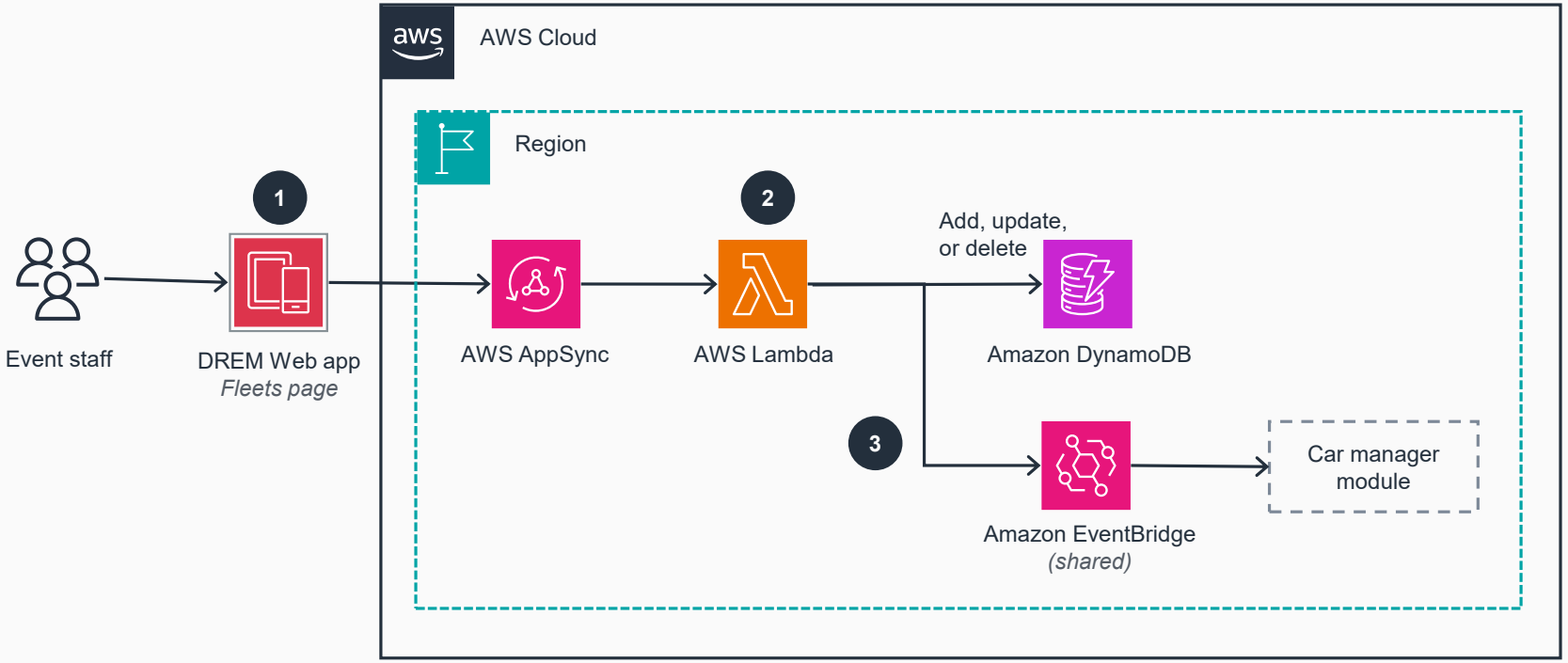
1. Racers and staff authenticate with **Cognito.**
2. Each racer uploads the pre-trained **AWS DeepRacer** models they want to test or race on the physical **AWS DeepRacer** car to **Amazon S3** through the DREM App. Racers can also delete their models from **Amazon S3**.
3. An event staff member selects specific uploaded **AWS DeepRacer** model(s) to send to a particular connected **AWS DeepRacer** car. Multiple cars are connected to enable cars to be prepared to race while others are racing, and 'swap outs' take place in the event of a crash.
4. The upload request is sent to the **AWS DeepRacer** car through an **AWS Appsync** API which validates authorization with **Cognito.**
5. The API invokes a **Lambda** function which generates a short lived pre-signed URL for the model, and then uses **Systems Manager** to send a **Systems Manager** Run Command to the car.
6. The run command processes a download script on the car, which uses the **Amazon S3** pre-signed URLs to download the **AWS DeepRacer** model onto the car that is ready for use by the racer.

Reviewed for technical accuracy October 25, 2023

AWS Reference Architecture

Fleet Manager

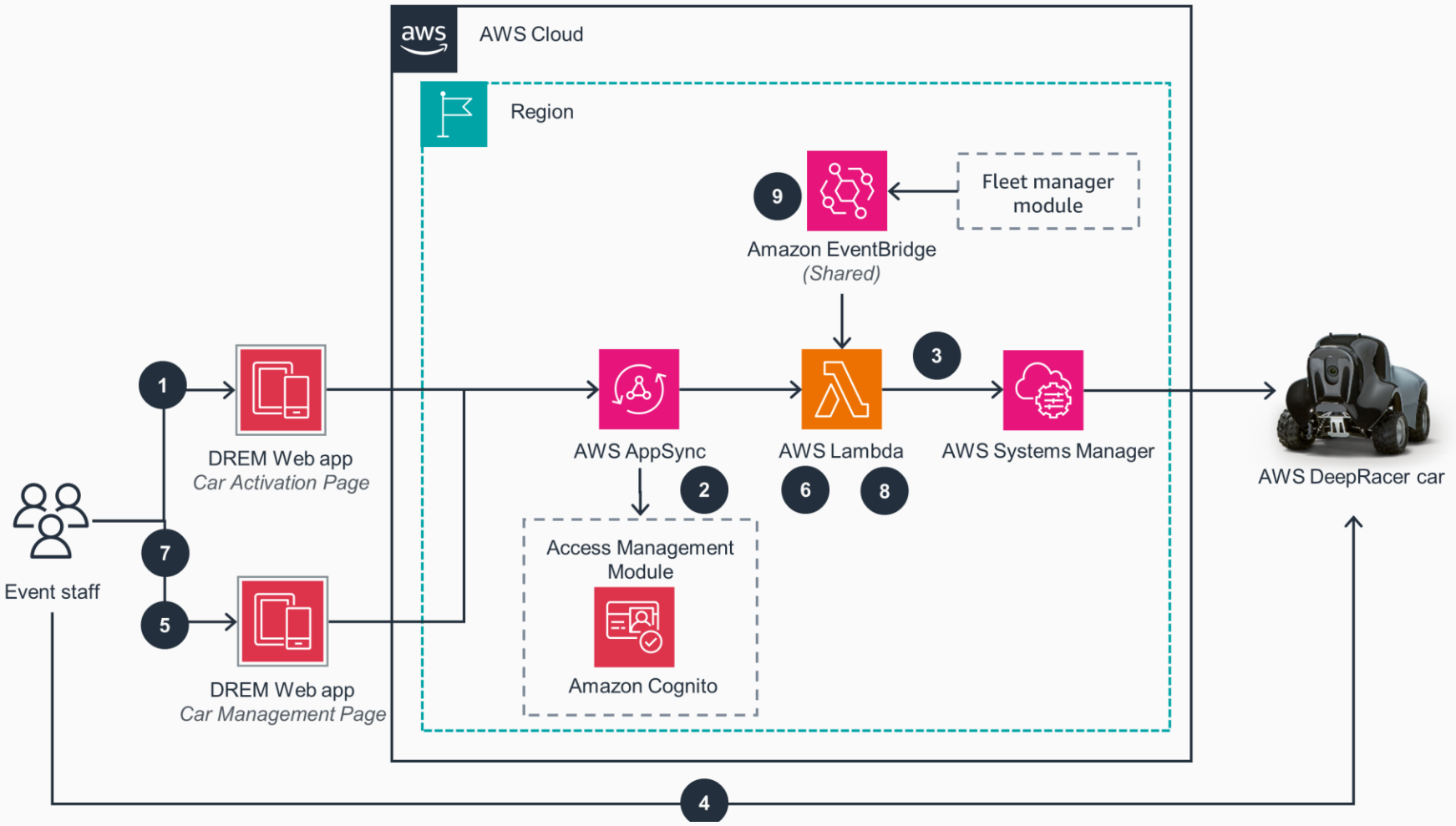
The fleet feature allows event organizers to designate a collection of cars in the DREM application. One use case is tracking cars associated with specific event hardware kits.



1. The event staff can create, update, or delete a car fleet. The request is submitted to **AWS AppSync.**
2. The request is sent through **AWS AppSync** using a **Lambda** resolver, and each fleet is stored as a **DynamoDB** item.
3. The **Lambda** resolver also emits an **EventBridge** event to notify the car manager module of the change.

Car Manager

Event staff members use the car manager feature to add cars to DREM via car activation. It's also used to remotely restart the AWS DeepRacer service on a car.



1. Event staff complete and submit a new car activation request in the DREM Web app and submit the request to **AWS AppSync.**
2. The activation request is sent to **AWS AppSync** which validates the requestor's authorization with **Cognito,** and then invokes a **Lambda** function.
3. The **Lambda** function creates a new AWS Systems Manager Hybrid Activation and returns a command line interface (CLI) prompt with the activation information passed back to event staff.
4. The **Lambda** function creates a new **Systems Manager** Hybrid Activation and returns a CLI command with the activation information passed back to event staff.
5. An event staff member submits commands to remotely administer the **AWS DeepRacer** car, such as delete all models, restart Robot Operating System (ROS), or update tail light color. The command requests are submitted to **AWS AppSync.**
6. Commands are sent through **AWS AppSync**, **Lambda**, **Systems Manager,** and sent to **AWS DeepRacer** cars as **Systems Manager** Run Commands.
7. Event staff members generate preformatted labels to print and affix to the cars, which aids in car identification. The request to create the labels is sent to **AWS AppSync.**
8. The label command is sent through **AWS AppSync** and **Lambda**, which generates the label based on data retrieved from **Systems Manager.**
9. Car tagging updates, such as updating which fleet a car belongs to, arrive from the fleet manager module through **EventBridge.** Next, they are processed through **Lambda**, which updates the tags in **Systems Manager.**

Reviewed for technical accuracy October 25, 2023

AWS Reference Architecture