Implementation Guide

DeepRacer on AWS

DeepRacer on AWS: Implementation Guide

Table of Contents

Overview

Publication date: January 2026.

DeepRacer on AWS is an educational ecosystem built around a fully autonomous 1/18th scale race car driven by reinforcement learning. With DeepRacer on AWS, you can train reinforcement learning models and evaluate their performance in the DeepRacer simulator. After training your models, you can compete in virtual races against other participants or download your trained models for deployment to your AWS DeepRacer vehicle, enabling autonomous driving in physical environments.

The solution leverages Amazon SageMaker AI for reinforcement learning model training and offers you the following capabilities.

- Deploy the necessary infrastructure including a console, simulation application used in the reinforcement learning training jobs.
- Provides option to customize reward function for the reinforcement learning model to train.
- Enables setting up and managing community races where custom models can be imported into the solution.
- Provides option to download the trained models which can be utilized in custom autonomous vehicles.

This implementation guide describes architectural considerations and configuration steps for deploying DeepRacer on AWS in the AWS Cloud. It includes links to an AWS CloudFormation template that launches and configures the AWS services required to deploy this solution using AWS best practices for security and availability.

The intended audience for implementing the DeepRacer on AWS solution in their environment includes solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals.

Use this navigation table to quickly find answers to these questions:

If you want to . . .	Read . . .
Know the cost for running this solution.	Cost

If you want to . . .	Read . . .
Understand the security considerations for this solution.	Security
Know how to plan for quotas for this solution.	Quotas
Know which AWS Regions support this solution.	Supported AWS Regions
View or download the AWS CloudFormation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution.	CloudFormation template
Access the source code.	GitHub repository

Note

For a general reference of AWS terms, see the AWS Glossary.

What is DeepRacer?

AWS DeepRacer is a fully autonomous 1/18th scale race car driven by reinforcement learning. It consists of the following components:

1. **DeepRacer on AWS**: A customer-deployable machine learning solution for training and evaluating reinforcement learning models in a three-dimensional simulated autonomous-driving environment.
2. **AWS DeepRacer vehicle**: A 1/18th scale RC car capable of running inference on a trained AWS DeepRacer model for autonomous driving.

DeepRacer on AWS

DeepRacer on AWS is a customer-deployable platform for learning and applying reinforcement learning concepts. You can use the graphical user interface to train a reinforcement learning model and to evaluate the model performance in the AWS DeepRacer simulator. From the console, you can also download a trained model for deployment to your AWS DeepRacer vehicle for autonomous driving in a physical environment.

The DeepRacer on AWS console supports the following features:

1. Create a training job to train a reinforcement learning model with a specified reward function, optimization algorithm, environment, and hyperparameters.
2. Choose a simulated track to train and evaluate a model using SageMaker AI.
3. Clone a trained model to improve training by tuning hyperparameters to optimize your model's performance.
4. Download a trained model for deployment to your AWS DeepRacer vehicle so it can drive in a physical environment.
5. Submit your model to a virtual race and have its performance ranked against other models in a virtual leaderboard.

When you use DeepRacer on AWS, you are charged based on your usage to train or evaluate and store models.

For details about pricing see the Cost section.

AWS DeepRacer vehicle

The AWS DeepRacer vehicle is a Wi-Fi enabled, physical vehicle that can drive itself on a physical track by using a reinforcement learning model.

1. You can manually control the vehicle or deploy a model for the vehicle to drive autonomously.
2. The autonomous mode runs inference on the vehicle's compute module. Inference uses images that are captured from the camera that is mounted on the front.
3. A Wi-Fi connection allows the vehicle to download software. The connection also allows the user to access the device console to operate the vehicle by using a computer or mobile device.

Topics

Explore reinforcement learning

Reinforcement learning, especially deep reinforcement learning, has proven effective in solving a wide array of autonomous decision-making problems. It has applications in financial trading, data center cooling, fleet logistics, and autonomous racing, to name a few.

Reinforcement learning has the potential to solve real-world problems. However, it has a steep learning curve because of its extensive technological scope and depth. Real-world experimentation requires that you construct a physical agent, such as an autonomous racing car. It also requires that you secure a physical environment, such as a driving track or public road. The environment can be costly, hazardous, and time-consuming. These requirements go beyond merely understanding reinforcement learning.

To help reduce the learning curve, DeepRacer on AWS simplifies the process in three ways:

1. Offering step-by-step guidance when training and evaluating reinforcement learning models. The guidance includes pre-defined environments, states, and actions, and customizable reward functions.
2. Providing a simulator to emulate interactions between a virtual agent and a virtual environment.
3. Using an AWS DeepRacer vehicle as a physical agent. Use the vehicle to evaluate a trained model in a physical environment. This closely resembles a real-world use case.

If you are a seasoned machine learning practitioner, you will find DeepRacer on AWS a welcome opportunity to build reinforcement learning models for autonomous racing in both virtual and physical environments. To summarize, use DeepRacer on AWS to create reinforcement learning models for autonomous racing with the following steps:

1. Train a custom reinforcement learning model for autonomous racing.
2. Use the simulator to evaluate a model and test autonomous racing in a virtual environment.
3. Deploy a trained model to an AWS DeepRacer to test autonomous racing in a physical environment.

Concepts and terminology

DeepRacer on AWS

DeepRacer on AWS is a machine learning solution for exploring reinforcement learning that is focused on autonomous racing. The DeepRacer on AWS console supports the following features:

1. Train a reinforcement learning model in the cloud.
2. Evaluate a trained model in a virtual simulator.
3. Submit a trained model to a virtual race and, if qualified, have its performance posted to the event's leaderboard.
4. Clone a trained model to continue training for improved performance.
5. Download the trained model artifacts for uploading to an AWS DeepRacer vehicle.
6. Place the vehicle on a physical track for autonomous driving and evaluate the model for real-world performance.
7. Remove unnecessary charges by deleting models that you don't need.

AWS DeepRacer

"AWS DeepRacer" can refer to three different vehicles:

1. The virtual race car can take the form of the original AWS DeepRacer device, the Evo device, or various digital rewards that can be earned by participating in community races. You can also customize the virtual car by changing its color.
2. The original AWS DeepRacer device is a physical 1/18th-scale model car. It has a mounted camera and an onboard compute module. The compute module runs inference in order to drive

itself along a track. The compute module and the vehicle chassis are powered by dedicated batteries known as the compute battery and the drive battery, respectively.

3. The AWS DeepRacer Evo device is the original device with an optional sensor kit. The kit includes an additional camera and LIDAR (light detection and ranging), which allow the car to detect objects behind and lateral to itself. The kit also includes a new shell.

Reinforcement learning

Reinforcement learning is a machine learning method that is focused on autonomous decision-making by an agent in order to achieve specified goals through interactions with an environment. In reinforcement learning, learning is achieved through trial and error and training does not require labeled input. Training relies on the reward hypothesis, which posits that all goals can be achieved by maximizing a future reward after action sequences. In reinforcement learning, designing the reward function is important. Better-crafted reward functions result in better decisions by the agent.

For autonomous racing, the agent is a vehicle. The environment includes traveling routes and traffic conditions. The goal is for the vehicle to reach its destination quickly without accidents. Rewards are scores used to encourage safe and speedy travel to the destination. The scores penalize dangerous and wasteful driving.

To encourage learning during training, the learning agent must be allowed to sometimes pursue actions that might not result in rewards. This is referred to as the exploration and exploitation trade-off. It helps reduce or remove the likelihood that the agent might be misguided into false destinations.

For a more formal definition, see reinforcement learning on Wikipedia.

Reinforcement learning model

A reinforcement learning model is an environment in which an agent acts that establishes three things: The states that the agent has, the actions that the agent can take, and the rewards that are received by taking action. The strategy with which the agent decides its action is referred to as a policy. The policy takes the environment state as input and outputs the action to take. In reinforcement learning, the policy is often represented by a deep neural network. We refer to this as the reinforcement learning model. Each training job generates one model. A model can be generated even if the training job is stopped early. A model is immutable, which means it cannot be modified and overwritten after it's created.

Simulator

The simulator is a virtual environment for visualizing training and evaluating models.

AWS DeepRacer vehicle

See AWS DeepRacer above.

AWS DeepRacer car

This type of DeepRacer vehicle is a 1/18th-scale model car.

Leaderboard

A leaderboard is a ranked list of vehicle performances in a race. The race can be a virtual event, carried out in the simulated environment, or a physical event, carried out in a real-world environment. The performance metric depends on the race type. It can be the fastest lap time, total time, or average lap time submitted by users who have evaluated their trained models on a track identical or similar to the given track of the race.

If a vehicle completes three laps consecutively, then it qualifies to be ranked on a leaderboard. The average lap time for the first three consecutive laps is submitted to the leaderboard.

Machine learning frameworks

Machine learning frameworks are the software libraries used to build machine learning algorithms. Supported frameworks include TensorFlow.

Policy network

A policy network is a neural network that is trained. The policy network takes video images as input and predicts the next action for the agent. Depending on the algorithm, it may also evaluate the value of the current state of the agent.

Optimization algorithm

An optimization algorithm is the algorithm used to train a model. For supervised training, the algorithm is optimized by minimizing a loss function with a particular strategy to update weights. For reinforcement learning, the algorithm is optimized by maximizing the expected future rewards with a particular reward function.

Neural network

A neural network (also known as an artificial neural network) is a collection of connected units or nodes that are used to build an information model based on biological systems. Each node is called an artificial neuron and mimics a biological neuron in that it receives an input (stimulus), becomes activated if the input signal is strong enough (activation), and produces an output predicated upon the input and activation. It's widely used in machine learning because an artificial neural network can serve as a general-purpose approximation to any function. Teaching machines to learn becomes finding the optimal function approximation for the given input and output. In deep reinforcement learning, the neural network represents the policy and is often referred to as the policy network. Training the policy network amounts to iterating through steps that involve generating experiences based on the current policy, followed by optimizing the policy network with the newly generated experiences. The process continues until certain performance metrics meet required criteria.

Hyperparameters

Hyperparameters are algorithm-dependent variables that control the performance of neural network training. An example hyperparameter is the learning rate that controls how many new experiences are counted in learning at each step. A larger learning rate results in a faster training but may make the trained model lower quality. Hyperparameters are empirical and require systematic tuning for each training.

AWS DeepRacer track

A track is a path or course on which an AWS DeepRacer vehicle drives. The track can exist in either a simulated environment or a real-world, physical environment. You use a simulated environment for training a model on a virtual track. The DeepRacer on AWS console makes virtual tracks available. You use a real-world environment for running an AWS DeepRacer vehicle on a physical track.

The AWS DeepRacer League provides physical tracks for event participants to compete. You must create your own physical track if you want to run your AWS DeepRacer vehicle in any other situation. This is documented in the *Build a race track* section in this guide.

Reward function

A reward function is an algorithm within a learning model that tells the agent whether the action performed resulted in:

1. A good outcome that should be reinforced.
2. A neutral outcome.
3. A bad outcome that should be discouraged.

The reward function is a key part of reinforcement learning. It determines the behavior that the agent learns by incentivizing specific actions over others. The user provides the reward function by using Python. This reward function is used by an optimizing algorithm to train the reinforcement learning model.

Experience episode

An experience episode is a period in which the agent collects experiences as training data from the environment by running from a given starting point to completing the track or going off the track. Different episodes can have different lengths. This is also referred to as an episode or experience-generating episode.

Experience iteration

Experience iteration (also known as experience-generating iteration) is a set of consecutive experiences between each policy iteration that performs updates of the policy network weights. At the end of each experience iteration, the collected episodes are added to an experience replay or buffer. The size can be set in one of the hyperparameters for training. The neural network is updated by using random samples of the experiences.

Policy iteration

Policy iteration (also known as policy-updating iteration) is any number of passes through the randomly sampled training data to update the policy neural network weights during gradient ascent. A single pass through the training data to update the weights is also known as an epoch.

Training job

A training job is a workload that trains a reinforcement learning model and creates trained model artifacts on which to run inference. Each training job has two sub-processes:

1. Start the agent to follow the current policy. The agent explores the environment in a number of episodes and creates training data. This data generation is an iterative process itself.
2. Apply the new training data to compute new policy gradients. Update the network weights and continue training. Repeat Step 1 until a stop condition is met.

Each training job produces a trained model and outputs the model artifacts to a specified data store.

Evaluation job

An evaluation job is a workload that tests the performance of a model. Performance is measured by given metrics after the training job is done. The standard AWS DeepRacer performance metric is the driving time that an agent takes to complete a lap on a track. Another metric is the percentage of the lap completed.

How DeepRacer works

The AWS DeepRacer vehicle is a 1/18th scale vehicle that can autonomously drive along a track by itself or race against another vehicle. The vehicle can be equipped with various sensors that include a front-facing camera, stereo cameras, radar, or LiDAR. The sensors collect data about the environment the vehicle operates in. Different sensors provide the view at different scales.

DeepRacer on AWS uses reinforcement learning to enable autonomous driving for the AWS DeepRacer vehicle. To achieve this, you train and evaluate a reinforcement learning model in a virtual environment with a simulated track. After the training, you upload the trained model artifacts to your AWS DeepRacer vehicle. You can then set the vehicle for autonomous driving in a physical environment with a real track.

Training a reinforcement learning model can be challenging, especially if you're new to the field. DeepRacer on AWS simplifies the process by integrating required components together and providing easy-to-follow wizard-like task templates. However, it's helpful to have a good understanding of the basics of reinforcement learning training implemented in DeepRacer on AWS.

Topics

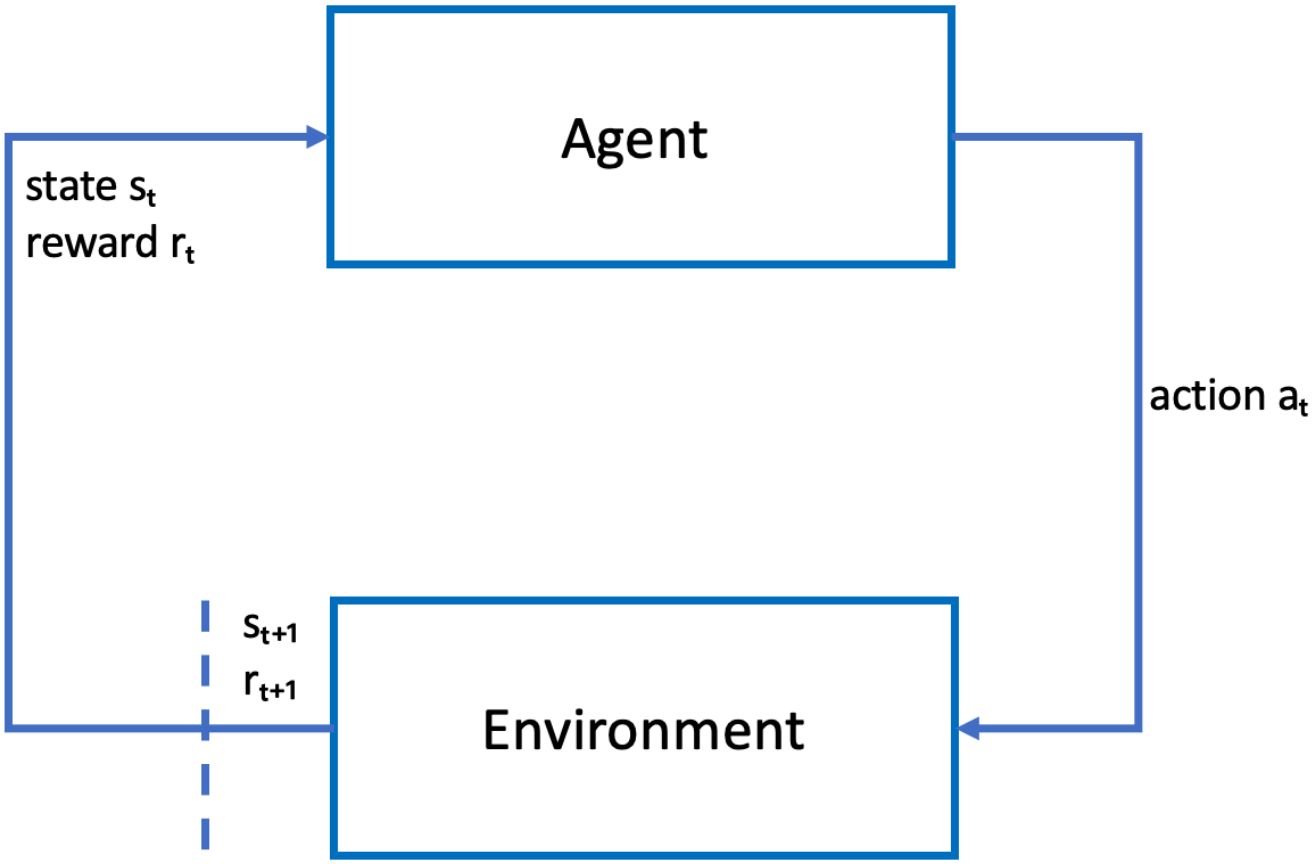
Reinforcement learning overview

In reinforcement learning, an agent, such as a physical or virtual AWS DeepRacer vehicle, with an objective to achieve an intended goal interacts with an environment to maximize the agent's total reward. The agent takes an action, guided by a strategy referred to as a policy, at a given environment state and reaches a new state. There is an immediate reward associated with any action. The reward is a measure of the desirability of the action. This immediate reward is considered to be returned by the environment.

The goal of the reinforcement learning in DeepRacer is to learn the optimal policy in a given environment. Learning is an iterative process of trials and errors. The agent takes a random initial

action to arrive at a new state. Then the agent iterates the step from the new state to the next one. Over time, the agent discovers actions that lead to the maximum long-term rewards. The interaction of the agent from an initial state to a terminal state is called an episode.

The following sketch illustrates this learning process:



The agent embodies a neural network that represents a function to approximate the agent's policy. The image from the vehicle's front camera is the environment state and the agent action is defined by the agent's speed and steering angles.

The agent receives positive rewards if it stays on-track to finish the race and negative rewards for going off-track. An episode starts with the agent somewhere on the race track and finishes when the agent either goes off-track or completes a lap.

> **Note**
>
> Strictly speaking, the environment state refers to everything relevant to the problem. For example, the vehicle's position on the track as well as the shape of the track. The

image fed through the camera mounted on the vehicle's front does not capture the entire environment state. For this reason, the environment is deemed partially observed and the input to the agent is referred to as observation rather than state. For simplicity, we use state and observation interchangeably throughout this documentation.

Training the agent in a simulated environment has the following advantages:

- The simulation can estimate how much progress the agent has made and identify when it goes off the track to compute a reward.
- The simulation relieves the trainer from tedious chores to reset the vehicle each time it goes off the track, as is done in a physical environment.
- The simulation can speed up training.
- The simulation provides better controls of the environment conditions, for example selecting different tracks, backgrounds, and vehicle conditions.

The alternative to reinforcement learning is supervised learning, also referred to as imitation learning. Here, a known dataset (of [image, action] tuples) collected from a given environment is used to train the agent. Models that are trained through imitation learning can be applied to autonomous driving. They work well only when the images from the camera look similar to the images in the training dataset.

For robust driving, the training dataset must be comprehensive. In contrast, reinforcement learning does not require such extensive labeling efforts and can be trained entirely in simulation. Because reinforcement learning starts with random actions, the agent learns a variety of environment and track conditions. This makes the trained model robust.

Action space and reward function

Action space

In reinforcement learning, the set of all valid actions, or choices, available to an agent as it interacts with an environment is called an action space. In the DeepRacer on AWS console, you can train agents in either a discrete or continuous action space.

Discrete action space

A discrete action space represents all of an agent's possible actions for each state in a finite set. For DeepRacer, this means that for every incrementally different environmental situation, the agent's neural network selects a speed and direction for the car based on input from its camera(s) and (optional) LiDAR sensor. The choice is limited to a grouping of predefined steering angle and throttle value combinations.

An AWS DeepRacer vehicle in a discrete action space approaching a turn can choose to accelerate or brake and turn left, right, or go straight. These actions are defined as a combination of steering angle and speed creating a menu of options, 0-9, for the agent. For example, 0 could represent -30 degrees and 0.4 m/s, 1 could represent -30 degrees and 0.8 m/s, 2 could represent -15 degrees and 0.4 m/s, 3 could represent -15 degrees and 0.8 m/s and so on through 9. Negative degrees turn the car right, positive degrees turn the car left and 0 keeps the wheels straight.

The default discrete action space contains the following actions:

Default discrete action space		
Action number	**Steering**	**Speed**
0	-30 degrees	0.4 m/s
1	-30 degrees	0.8 m/s
2	-15 degrees	0.4 m/s
3	-15 degrees	0.8 m/s
4	0 degrees	0.4 m/s
5	0 degrees	0.8 m/s
6	15 degrees	0.4 m/s
7	15 degrees	0.8 m/s
8	30 degrees	0.4 m/s
9	30 degrees	0.8 m/s

Continuous action space

A continuous action space allows the agent to select an action from a range of values for each state. Just as with a discrete action space, this means for every incrementally different environmental situation, the agent's neural network selects a speed and direction for the car based on input from its camera(s) and (optional) LiDAR sensor. However, in a continuous action space, you can define the range of options the agent picks its action from.

In this example, the AWS DeepRacer vehicle in a continuous action space approaching a turn can choose a speed from 0.75 m/s to 4 m/s and turn left, right, or go straight by choosing a steering angle from -20 to 20 degrees.

Discrete vs. continuous

The benefit of using a continuous action space is that you can write reward functions that train models to incentivize speed/steering actions at specific points on a track that optimize performance. Picking from a range of actions also creates the potential for smooth changes in speed and steering values that, in a well trained model, may produce better results in real-life conditions.

In the discrete action space setting, limiting an agent's choices to a finite number of predefined actions puts the onus on you to understand the impact of these actions and define them based on the environment (track, racing format) and your reward functions. However, in a continuous action space setting, the agent learns to pick the optimal speed and steering values from the min/max bounds you provide through training.

Though providing a range of values for the model to pick from seems to be the better option, the agent has to train longer to learn to choose the optimal actions. Success is also dependent upon the reward function definition.

Reward function

As the agent explores the environment, the agent learns a value function. The value function helps your agent judge how good an action taken is, after observing the environment. The value function uses the reward function that you write in the DeepRacer on AWS console to score the action. For example, in the follow the center line sample reward function in the DeepRacer on AWS console, a good action would keep the agent near the center of the track and be scored higher than a bad action, which would move the agent away from the center of the track.

Over time, the value function helps the agent learn policies that increase the total reward. The optimal, or best policy, would balance the amount of time the agent spends exploring the

environment with the amount of time it spends exploiting, or making the best use of, what the policy has learned through experience.

In the *Time trial - follow the center line* sample reward function example (see Reward function examples), the agent first takes random actions to explore the environment, which means it doesn't do a very good job of staying in the center of the track. Over time, the agent begins to learn which actions keep it near the center line, but if it does this by continuing to take random actions, it will take a long time to learn to stay near the center of the track for the entire lap. So, as the policy begins to learn good actions, the agent begins to use those actions instead of taking random actions. However, if it always uses or exploits the good actions, the agent won't make any new discoveries, because it's no longer exploring the environment. This trade-off is often referred to as the exploration vs exploitation problem in RL.

Experiment with the default action spaces and sample reward functions. Once you've explored them all, put your knowledge to use by designing your own custom action spaces and custom reward functions.

Training algorithms

Proximal Policy Optimization (PPO) versus Soft Actor Critic (SAC)

The algorithms SAC and PPO both learn a policy and value function at the same time, but their strategies vary in three notable ways:

PPO	SAC
Works in both discrete and continuous action spaces	Works in a continuous action space
On-policy	Off-policy
Uses entropy regularization	Adds entropy to the maximization objective

Stable vs. data hungry

The information learned by the PPO and SAC algorithms' policies while exploring an environment is utilized differently. PPO uses on-policy learning which means that it learns its value function

from observations made by the current policy exploring the environment. SAC uses off-policy learning which means that it can use observations made by previous policies' exploration of the environment. The trade-off between off-policy and on-policy learning is often stability vs. data efficiency. On-policy algorithms tend to be more stable but data hungry, whereas off-policy algorithms tend to be the opposite.

Exploration vs. exploitation

Exploration vs. exploitation is a key challenge in RL. An algorithm should exploit known information from previous experiences to achieve higher cumulative rewards, but it also needs to explore to gain new experiences that can be used in finding the optimum policy in the future. As a policy is trained over multiple iterations and learns more about an environment, it becomes more certain about choosing an action for a given observation. However, if the policy doesn't explore enough, it will likely stick to information already learned even if it's not at an optimum. The PPO algorithm encourages exploration by using entropy regularization, which prevents agents from converging to local optima. The SAC algorithm strikes an exceptional balance between exploration and exploitation by adding entropy to its maximization objective.

Entropy

In this context, "entropy" is a measure of the uncertainty in the policy, so it can be interpreted as a measure of how confident a policy is at choosing an action for a given state. A policy with low entropy is very confident at choosing an action, whereas a policy with high entropy is unsure of which action to choose.

The SAC algorithm's entropy maximization strategy has similar advantages to the PPO algorithm's use of entropy as a regularizer. Like PPO, it encourages wider exploration and avoids convergence to a bad local optimum by incentivizing the agent to choose an action with higher entropy. Unlike entropy regulation, entropy maximization has a unique advantage. It tends to give up on policies that choose unpromising behavior, which is another reason that the SAC algorithm tends to be more data efficient than PPO.

Tune the amount of entropy in SAC by using the SAC alpha hyperparameter. The maximum SAC alpha entropy value (1.0) favors exploration. The minimum value (0.0) recovers the standard RL objective and neutralizes the entropy bonus that incentivizes exploration. A good SAC alpha value to begin experimenting with is 0.5. Tune accordingly as you iterate on your models.

Try both PPO and SAC algorithms, experiment with their hyperparameters, and explore with them in different action spaces.

User workflow

Training a model with DeepRacer on AWS involves the following general tasks:

1. The solution initializes the simulation with a virtual track, an agent representing the vehicle, and the background. The agent embodies a policy neural network that can be tuned with hyperparameters as defined in the PPO algorithm.
2. The agent acts (as specified with a steering angle and a speed) based on a given state (represented by an image from the front camera).
3. The simulated environment updates the agent's position based on the agent action and returns a reward and an updated camera image. The experiences collected in the form of state, action, reward, and new state are used to update the neural network periodically. The updated network models are used to create more experiences.
4. You can monitor the training in progress along the simulated track with a first-person view as seen by the agent. You can display metrics such as rewards per episode, the loss function value, the entropy of the policy. CPU or memory utilization can also be displayed as training progresses. In addition, detailed logs are recorded for analysis and debugging.
5. The neural network model is periodically saved to persistent storage.
6. The training stops based on a time limit.
7. You can evaluate the trained model in a simulator. To do this, submit the trained model for time trials for a selected number of runs on the selected track.

After the model is successfully trained and evaluated, it can be uploaded to a physical agent (an AWS DeepRacer vehicle). The process involves the following steps:

1. Download the trained model from its persistent storage (an Amazon S3 bucket).
2. Use the vehicle's device control console to upload the trained model to the device. Use the console to calibrate the vehicle for mapping the simulated action space to the physical action space. You can also use the console to check the throttling parity, view the front camera feed, load a model into the inference engine, and watch the vehicle driving on a real track.
3. The vehicle's device control console is a web server hosted on the vehicle's compute module. The console is accessible from the vehicle IP address with a connected Wi-Fi network and a web browser on a computer or a mobile device.
4. Experiment with the vehicle driving under different lighting, battery levels, and surface textures and colors.

5. The device's performance in a physical environment may not match the performance in a simulated environment due to model limitations or insufficient training. The phenomenon is referred to as the *sim2real* performance gap. To reduce the gap, see Simulated-to-real performance gaps.

Simulated-to-real performance gaps

Because the simulation cannot capture all aspects of the real world accurately, the models trained in simulation may not work well in the real world. Such discrepancies are often referred to as simulated-to-real (*sim2real*) performance gaps.

Efforts have been made in DeepRacer on AWS to minimize the *sim2real* performance gap. For example, the simulated agent is programmed to take about 10 actions per second. This matches the frequency the AWS DeepRacer device runs inference with, about 10 inferences per second. As another example, at the start of each episode in training, the agent's position is randomized. This maximizes the likelihood that the agent learns all parts of the track evenly.

To help reduce *sim2real* performance gaps, make sure to use the same or similar color, shape and dimensions for both the simulated and real tracks. To reduce visual distractions, use barricades around the real track. Also, carefully calibrate the ranges of the device's speed and steering angles so that the action space used in training matches the real world. Evaluating model performance in a different simulation track than the one used in training can show the extent of the sim2real performance gap.

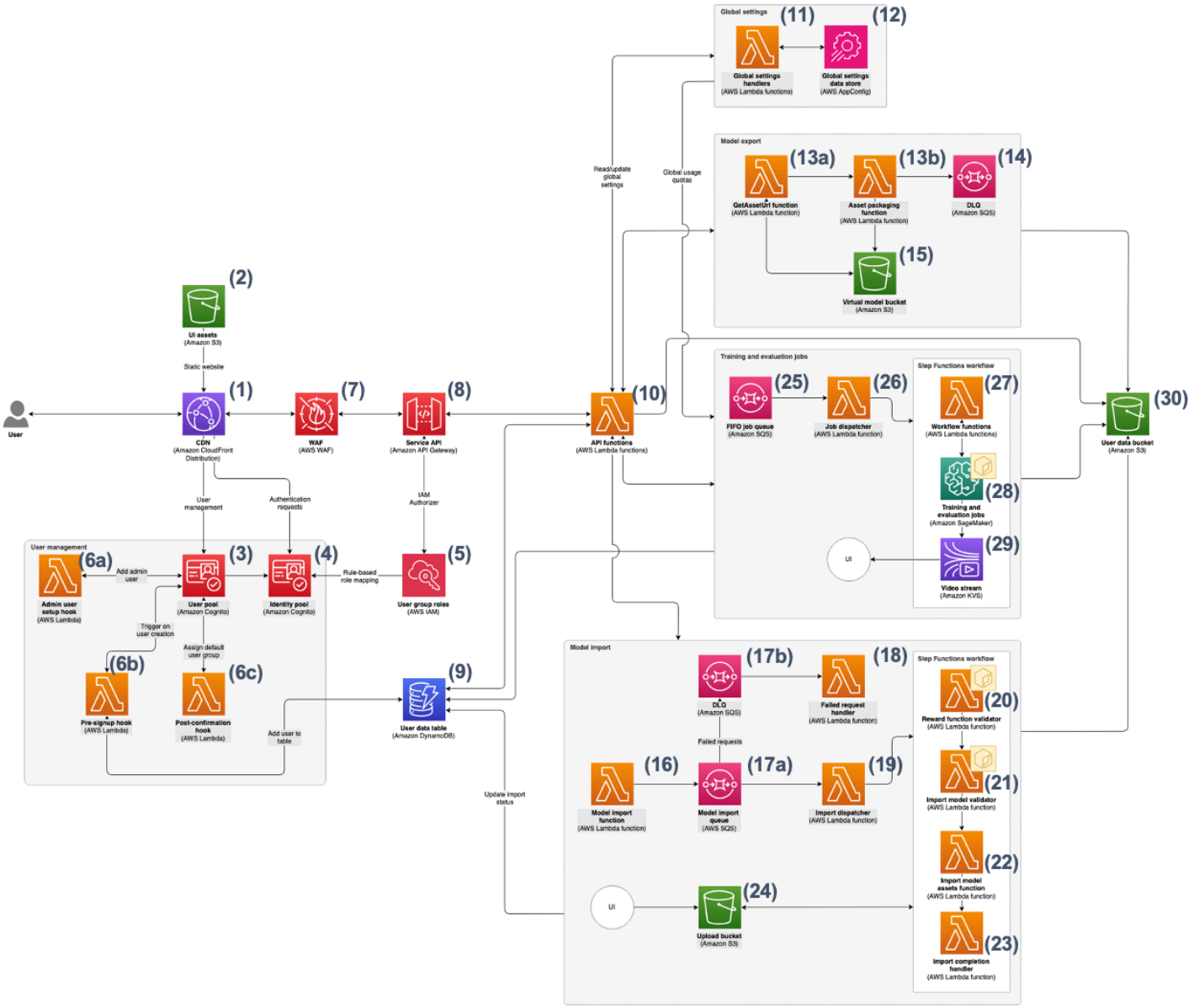
Architecture overview

This section provides an overview of the architecture of this solution.

Topics

Architecture diagram

Deploying this solution with the default parameters deploys the following components in your AWS account.



Architectural components

1. A user accesses the DeepRacer on AWS user interface through an Amazon CloudFront distribution, which delivers static web assets from the UI assets bucket and video streams from simulations.
2. The user interface assets are hosted in an Amazon S3 bucket that stores the static web assets comprising the user interface.
3. An Amazon Cognito user pool manages users and user group membership.
4. An Amazon Cognito identity pool manages federation and rule-based role mapping for users.

5. AWS IAM roles define permissions and level-of-access for each user group in the system, used for access control and authorization.
6. AWS Lambda registration hooks execute pre- and post-registration actions including assigning new users as racers, handling initial admin profile creation, and more.
7. AWS WAF provides intelligent protection for the API against common attack vectors and allows customers to define custom rules based on individual use cases and usage patterns.
8. Amazon API Gateway routes API requests to their appropriate handler using a defined Smithy model.
9. A single AWS DynamoDB table is responsible for storing and managing profiles, training jobs, models, evaluation jobs, submissions, and leaderboards.
10. AWS Lambda functions are triggered in response to requests routed from the API and are responsible for CRUD operations, dispatching training/evaluation jobs, and more.
11. A global settings handler (AWS Lambda function) reads and writes application-level settings to the configuration.
12. An AWS AppConfig hosted configuration stores application-level settings, such as usage quotas.
13. Model export handlers (AWS Lambda functions) retrieve the asset URL and package assets for use in exporting models from the system.
14. An Amazon SQS dead-letter queue catches failed export jobs from the asset packaging function.
15. A virtual model bucket stores exported models and provides access to them via pre-signed URL.
16. A model import handler (AWS Lambda function) receives requests to import a model onto the system and creates a new import job.
17. A model import queue (Amazon SQS) receives jobs from the model import function and holds them until they are accepted by the dispatcher; a DLQ handles failed jobs.
18. A failed request handler (AWS Lambda function) manages failed requests and updates their status to reflect their current state.
19. An import dispatching function takes a job from the queue and dispatches it to the workflow.
20. A reward function validator (AWS Lambda function) checks the reward function and validates/sanitizes the customer-provided code before it is saved to the system.
21. An imported model validator function checks and validates the imported model before it is saved to the system.
22. An imported model assets handler (AWS Lambda function) brings in model assets from the upload bucket.

23 An import completion handler (AWS Lambda function) handles status updates when a job is completed successfully.

24 An upload bucket (Amazon S3) stores uploaded (but not yet imported) assets from the user.

25 An Amazon SQS FIFO queue receives requests for training and evaluation jobs and stores them in FIFO order.

26 A job dispatcher function picks a job off the top of the FIFO queue and dispatches it to the workflow.

27 Workflow functions handle setting up the job, setting status, and other workflow tasks.

28 Amazon SageMaker AI training jobs perform the actual training and evaluation of the model using the reward function and hyperparameters provided.

29 Amazon Kinesis Video Streams handles presenting the simulation video to the user from the training job.

30 A user data bucket stores all user data including trained models, evaluation results, and other assets generated during the DeepRacer workflow.

Functional components

This solution implements a serverless, microservices-based architecture that enables users to train and evaluate reinforcement learning models for autonomous racing. The architecture is organized around several key functional areas that work together to provide a complete reinforcement learning education platform.

User Interface and Authentication

Users access DeepRacer on AWS through a web-based console delivered via Amazon CloudFront, which provides fast, global distribution of the user interface assets. These static web assets are hosted in Amazon S3, ensuring reliable and scalable content delivery to users worldwide. Amazon Cognito manages user authentication and authorization, handling user registration, login, and session management.

When new users register, the system automatically creates user profiles and establishes proper permissions, ensuring a seamless onboarding experience. This authentication layer secures access to the platform while enabling users to maintain their own private workspace for models, training data, and race submissions.

API Layer and Request Processing

All user interactions with the system flow through Amazon API Gateway, which serves as the central entry point for backend operations. The API Gateway routes requests to appropriate AWS Lambda functions based on the endpoint accessed, providing a clean separation between the user interface and backend processing logic. AWS WAF protects the API layer from common security threats such as bot attacks, DDoS attempts, and malicious traffic patterns.

Data Management

The solution uses a combination of Amazon DynamoDB and Amazon S3 to handle different types of data storage needs. DynamoDB serves as the primary database for structured data including user profiles, model metadata, training job status, leaderboards, and race submissions. Amazon S3 handles file storage for larger assets such as trained model files, training logs, evaluation videos, and other user-generated content.

Model Training and Evaluation Engine

The core reinforcement learning functionality is centered around Amazon SageMaker AI training jobs, which provides the compute resources for running reinforcement learning training and evaluation jobs. When users initiate training jobs, the requests are queued in Amazon SQS to manage demand and ensure fair resource allocation. AWS Step Functions orchestrates the workflow of preparing training environments, monitoring job progress, and handling completion tasks. The system pulls a containerized simulation environment from Amazon ECR, which comprises the DeepRacer virtual simulator built on robotics simulation technology.

Real-time Simulation Streaming

During model training and evaluation, Amazon Kinesis Video Streams captures video from the simulation environment and streams it in real-time to the console. This allows users to watch their models learn and perform, providing immediate visual feedback on training progress and model behavior. The streaming capability delivers an engaging, visual experience that helps users understand how their models are developing and performing on the virtual race track.

Security and Validation

Before any user-provided code executes in the system, it passes through validation functions. These examine reward functions and imported models for security issues, ensuring that malicious

or harmful code cannot compromise the system. The functions operate within isolated network environments that prevent external communication, providing an additional security boundary.

Monitoring and Operations

Amazon CloudWatch provides comprehensive monitoring and logging across all system components, collecting metrics, logs, and performance data from Lambda functions, SageMaker instances, API Gateway, and other services. This enables cloud administrators to understand system performance, troubleshoot issues, and optimize resource usage.

AWS services

AWS service	Function	Description
Amazon API Gateway	Core	Hosts REST API endpoints in the solution.
AWS CloudFormation	Core	Used to deploy the solution.
Amazon CloudFront	Core	Serves the web content hosted in Amazon S3.
Amazon Cognito	Core	Handles user management and authentication for the API.
Amazon DynamoDB	Core	Stores all user data related to user profiles, models, leaderboards, and submissio ns in a single table.
Amazon Elastic Container Registry	Core	Stores the Simulation Application (SimApp) image as a public container image, which is used by SageMaker instances to run the DeepRacer simulation application.

AWS service	Function	Description
Amazon Kinesis Video Streams	Core	Streams videos from SageMaker AI training jobs to the user console, providing real-time visual feedback of model performance.
Amazon S3	Core	Hosts static web assets for the user console and stores user-generated artifacts such as model files, training logs, and evaluation videos.
Amazon SageMaker	Core	Runs the Simulation Application (SimApp) for training and evaluating DeepRacer models.
Amazon SQS	Core	Provides a first-in-first-out job queue that holds simulation jobs before they are forwarded to the job dispatcher.
AWS Lambda	Core	Powers various functions including API request handling, model validation, reward function validation, job dispatching, and workflow management.
AWS Step Functions	Core	Manages workflow functions that orchestrate training and evaluation jobs on SageMaker instances.

AWS service	Function	Description
AWS WAF	Core	Provides system protectio n against bot spam, DDoS attacks, credential stuffing, and other common attack vectors.
Amazon CloudWatch	Core	Provides monitoring and logging capabilities for all components of the DeepRacer on AWS solution.
AWS Identity and Access Management (IAM)	Core	Manages access control and permissions for various components of the DeepRacer on AWS solution.
Amazon Virtual Private Cloud (VPC)	Optional	Can be used to provide network isolation for SageMaker AI training jobs for enhanced security.

AWS Well-Architected design considerations

This solution follows best practices from the AWS Well-Architected Framework, which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework benefit this solution.

Topics

- Operational excellence
- Security
- Reliability
- Performance efficiency

- Cost optimization
- Sustainability

Operational excellence

This section describes how we architected this solution using the principles and best practices of the operational excellence pillar.

- All resources are defined as infrastructure as code using AWS CloudFormation templates generated from AWS CDK constructs.
- The solution pushes metrics to Amazon CloudWatch at various stages to provide observability into AWS Lambda functions, Amazon SageMaker, AWS Step Functions, Amazon S3 buckets, and other solution components.

Security

This section describes how we architected this solution using the principles and best practices of the security pillar.

- Amazon Cognito authenticates and authorizes web console users and API requests.
- All interservice communications use AWS Identity and Access Management (IAM) roles with least privilege access, containing only the minimum permissions required.
- All data storage, including S3 buckets and DynamoDB tables, encrypts data at rest using AWS managed keys.
- Logging, tracing, and versioning are enabled where applicable for audit and compliance purposes.

Reliability

This section describes how we architected this solution using the principles and best practices of the reliability pillar.

- The solution uses AWS serverless services wherever possible (examples: Lambda, API Gateway, Amazon S3, AWS Step Functions and Amazon DynamoDB) to ensure high availability and recovery from service failure.

- Data is stored in DynamoDB and Amazon S3, so it persists in multiple Availability Zones by default.

Performance efficiency

This section describes how we architected this solution using the principles and best practices of the performance efficiency pillar.

- The solution uses a serverless architecture with the ability to scale horizontally as needed.
- The solution can be launched in any region that supports the AWS services in this solution, which include: AWS Lambda, Amazon API Gateway, Amazon S3, AWS Step Functions, Amazon DynamoDB, and Amazon Cognito.
- The solution uses managed services throughout to reduce the operational burden of resource provisioning and management.
- The solution is automatically tested and deployed daily to achieve consistency as AWS services change, as well as reviewed by solution architects and subject matter experts for areas to experiment and improve.

Cost optimization

This section describes how we architected this solution using the principles and best practices of the cost optimization pillar.

- The solution uses a serverless architecture; therefore, you are only charged for what you use.
- Amazon DynamoDB scales capacity on demand, so you only pay for the capacity you use.
- Amazon SageMaker allows you to pay only for the compute resources you use, with no upfront expenses.

Sustainability

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

- The solution uses managed, serverless services to minimize the environmental impact of the backend services compared to continually operating on-premises services.
- Serverless services allow you to scale up or down as needed.

Plan your deployment

This section provides an overview of the cost, security, service quotas, and other key factors to consider prior to deploying the solution in your AWS account.

Topics

Supported AWS regions

DeepRacer on AWS is available in the following AWS Regions.

Region Name	Region Code
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1

Region Name	Region Code
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Spain)	eu-south-2
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1

For the most current availability of AWS services by Region, see the AWS Regional Services List.

Important

CloudFront Access Logging Limitation

CloudFront access logging is automatically disabled in the following regions due to lack of support for standard logging (legacy):

- Africa (Cape Town) - `af-south-1`
- Asia Pacific (Hong Kong) - `ap-east-1`
- Europe (Spain) - `eu-south-2`
- Middle East (Bahrain) - `me-south-1`

If you deploy the solution in one of these regions, the CloudFront distribution will function normally but will not generate access logs. If access logging is required for your use case, you can manually configure CloudFront Standard Logging V2 after deployment. For more information, refer to the CloudFront Standard Logging V2 documentation.

Cost

You are responsible for the cost of AWS services that are used while operating this solution. As of this revision, the monthly cost of operating this solution for a small group of 25 users with limited model training and evaluation demand is about $7.78 per user (or $194.37 total).

> **Note**
>
> The cost of operating DeepRacer on AWS depends on how you choose to use the solution. The following examples provide cost breakdown for single instance and multiple instances deployment configurations in the US East (N. Virginia) Region. AWS services listed in the example tables below are billed on a monthly basis.

The cost of operating the solution is dependent on the number of users registered to the system, the number of models they train, the number of evaluations they run, and the number of race submissions they perform. The cost of the solution is also dependent on the amount of time each model is trained for and the underlying instance type that is used for these jobs. The solution is configured to use an instance of type `ml.c5.4xlarge` by default. However, you may choose to change this based on your desired performance profile by updating the appropriate field in the solution's CloudFormation template or CDK project and deploying that configuration.

Operating cost is figured and expressed on a monthly basis. The estimates provided in this section are only accurate for use cases in which all training jobs, evaluations, and submissions occur within the same month. This is an important consideration to account for when planning usage over the course of a quarter, semester, or other longer timeframe.

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, refer to the pricing webpage for each AWS service used in this solution.

Compute usage formula

The training job compute usage for the examples in this section is figured using the following formula:

- Training hours per month = (25 users) * (3 models/user) * (1.5 hrs/training) = 112.5 hours
- Evaluation hours per month = [(25 users) * (3 models/user) * (3 evaluations/model)] * (20 mins/evaluation) = 75 hours

- Submission hours per month = [(25 users) * (3 submissions/user)] * (20 mins/submission) = 25 hours
- Total hours = (training hours) + (evaluation hours) + (submission hours) = 212.5 hours

Total SageMaker cost = [(212.5 hours) * ($0.816/instance hour)] + ($1.40 storage cost) = $174.80

Example 1 (25 users)

You are planning a deployment of DeepRacer on AWS that will serve 25 users, with each user creating and training 3 models. Each user will then evaluate each of their models 3 times, before submitting them to a race. With a training time of 90 minutes, the cost will be $194.37, or $7.78 per user.

Cost summary

AWS service	Dimensions	Cost per month (USD)
Amazon SageMaker	[(212.5 hours) * ($0.816/instance hour)] + ($1.40 storage cost)	$174.80
AWS WAF	1 Web ACL	$5.00
Amazon S3	(0.33GB average model size) * (75 models) * ($0.023/GB)	$0.57
Other services	AWS Lambda, Amazon DynamoDB, Amazon CloudWatch, AWS CodeBuild, Amazon ECR	$14.00
Monthly total	**$7.78/user**	**$194.37**

Example 2 (100 users)

You are planning a deployment of DeepRacer on AWS that will serve 100 users, with each user creating and training 3 models. Each user will then evaluate each of their models 3 times, before

submitting them to a race. With a training time of 90 minutes, the cost will be $716.28, or $7.17 per user.

Cost summary

AWS service	Dimensions	Cost per month (USD)
Amazon SageMaker	[(850 hours) * ($0.816/instance hour)] + ($1.40 storage cost)	$695.00
AWS WAF	1 Web ACL	$5.00
Amazon S3	(0.33GB average model size) * (300 models) * ($0.023/GB)	$2.28
Other services	AWS Lambda, Amazon DynamoDB, Amazon CloudWatch, AWS CodeBuild, Amazon ECR	$14.00
Monthly total	**$7.17/user**	**$716.28**

Example 3 (250 users)

You are planning a deployment of DeepRacer on AWS that will serve 250 users, with each user creating and training 3 models. Each user will then evaluate each of their models 3 times, before submitting them to a race. With a training time of 90 minutes, the cost will be $1,760.09, or $7.04 per user.

Cost summary

AWS service	Dimensions	Cost per month (USD)
Amazon SageMaker	[(2,125 hours) * ($0.816/instance hour)] + ($1.40 storage cost)	$1,735.40

AWS service	Dimensions	Cost per month (USD)
AWS WAF	1 Web ACL	$5.00
Amazon S3	(0.33GB average model size) * (750 models) * ($0.023/GB)	$5.69
Other services	AWS Lambda, Amazon DynamoDB, Amazon CloudWatch, AWS CodeBuild, Amazon ECR	$14.00
Monthly total	**$7.04/user**	**$1,760.09**

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about security on AWS, visit AWS Cloud Security

Security best practices

DeepRacer on AWS is designed with security best practices in mind. However, the security of a solution differs based on your specific use case. The following are additional recommendations to enhance the security posture of DeepRacer on AWS.

Use a dedicated account for deployment

We strongly recommend using a dedicated account that's separate from any production workloads for deploying and hosting DeepRacer on AWS. This separation helps prevent mixing different levels of data sensitivity, and reduces the potential blast radius in the event of a security incident.

Activate Ubuntu Pro to apply ESM patches

See Mitigate OS vulnerabilities with Ubuntu Pro.

Infrastructure security

Console UI

This solution deploys a web console hosted in an Amazon S3 bucket. To enhance security and reduce latency, the solution configures an Amazon CloudFront distribution with an origin access control (OAC). This OAC provides controlled public access to the solution's website bucket contents, ensuring that users can only access the web console through CloudFront and not directly from S3. For more information, see Restricting access to an Amazon S3 origin in the *Amazon CloudFront Developer Guide*.

The CloudFront distribution is configured to only accept HTTPS requests. If an HTTP request is received, it will redirect it to HTTPS to promote encryption in-transit.

CloudFront activates additional security mitigations to append HTTP security headers to each viewer response. For additional details, please see Adding or removing HTTP headers in CloudFront responses.

This solution uses the default CloudFront certificate, which has a minimum supported security protocol of TLS v1.0. To enforce the use of TLS v1.2 or TLS v1.3, you must use a custom SSL certificate instead of the default CloudFront certificate. For more information, refer to How do I configure my CloudFront distribution to use an SSL/TLS certificate.

Authentication and authorization

This solution uses Amazon Cognito and several other services for managing authentication and authorization. Authentication is handled by an Amazon Cognito user pool, which is configured with three user pool groups, one for each user type (i.e. admins, race facilitators, and racers). The AWS Amplify Auth plugin is used for authenticating users from the console and managing sessions.

Each user pool group has a dedicated IAM role mapped to it using rule-based role mapping in the identity pool. These roles define what resources a user in a given user pool group can or cannot access.

Authorization is handled via the issuance of temporary credentials from an Amazon Cognito identity pool, based on the role associated with their user pool group. These credentials are assumed by the user upon successful authentication into the console, and are used for signing requests to the back-end. When a request is received by the API, an IAM authorizer is used to authorize the request before proxying it to the appropriate Lambda function for servicing.

Identity and access management

AWS Identity and Access Management (IAM) roles are used to grant specific permissions to various resources that comprise DeepRacer on AWS. The following IAM roles are created:

1. Lambda execution roles: Allows AWS Lambda functions to access other AWS services such as Amazon S3, Amazon DynamoDB, and Amazon CloudWatch Logs.
2. SageMaker execution role: Allows SageMaker AI training jobs to access necessary resources like ECR images and S3 buckets.
3. Step Functions execution roles: Permits Step Functions to invoke Lambda functions and manage SageMaker jobs.
4. API Gateway execution roles: Enables Amazon API Gateway to invoke AWS Lambda functions in response to requests.
5. User roles: Allows for granular access control to be applied by user type.

These roles follow the principle of least privilege, granting only the permissions necessary for each component to perform its functions.

Log retention and monitoring

By default, DeepRacer on AWS retains all security-relevant logs for 10 years, which aligns with AWS security best practices. Security-relevant logs include logs emitted by AWS Lambda functions that support API services as well as authentication and authorization services. All other logs are retained for 2 years. You can customize the log retention period for one or more logs through the CloudWatch Logs console.

All logs are encrypted at-rest using AWS KMS customer-managed keys.

Amazon API Gateway

This solution deploys an Amazon API Gateway REST API and uses the default API endpoint and SSL certificate. The default API endpoint supports TLSv1 security policy. It is recommended to use the TLS_1_2 security policy to enforce TLSv1.2+ with your own custom domain name and custom SSL certificate.

For more information:

- Choose a security policy for your custom domain in API Gateway
- Custom domain name for public REST APIs in API Gateway

AWS CloudTrail

AWS CloudTrail is not automatically enabled by DeepRacer on AWS. AWS recommends enabling CloudTrail to monitor API calls and administrative actions in your account.

Amazon DynamoDB

All user data stored in Amazon DynamoDB is encrypted at-rest using customer managed keys (CMK) stored in AWS KMS.

AWS Key Management System

This solution creates one KMS Customer Managed Key (CMK) for the purpose of log encryption.

AWS Lambda functions

By default, all AWS Lambda functions that are configured by this solution use the most recent, stable version of the language runtime. No sensitive data or secrets are logged. Service interactions are carried out with the least required privilege. Roles that define these privileges are not shared between functions.

Amazon SageMaker

This solution uses the Amazon SDK to create Amazon SageMaker training jobs. These training jobs are responsible for servicing requests from users to train models, evaluate models, and simulate models in a competition.

Amazon S3

This solution deploys S3 buckets with default S3 bucket security configurations. For encryption of objects at rest, consider using customer managed CMKs instead of the default key for encrypting objects. Customer managed keys are recommended for customers who want full control over the lifecycle and usage of their keys.

It's a best practice to use modern encryption protocols for data in transit. To enforce the use of TLS version 1.2 or later for connections to S3, update your bucket's security policy.

It is recommended that S3 server access logging provides detailed records for the requests that are made to a bucket. Amazon S3 server access logging provides detailed records for the requests made to the bucket. S3 Access Logs can be enabled and saved in another S3 bucket.

For more information:

- Using server-side encryption with customer-provided keys (SSE-C)
- How do I enforce TLS 1.2 or later for my S3 buckets?
- Security best practices for Amazon S3

AWS Web application firewall (WAF)

The solution deploys AWS WAF to protect against common web exploits and bot traffic. It includes rules to mitigate against common vulnerabilities and allows for custom rule creation.

Data protection

DeepRacer on AWS uses an Amazon DynamoDB table and Amazon S3 buckets for storing models, profiles, training and evaluation outputs, and other assets uploaded to or generated by the solution. The following data protection settings are configured by default to mitigate against loss of data, unauthorized access, and other issues:

- For the **table**, point-in-time recovery is enabled with a backup recovery period of 35 days, allowing for the table to be rolled back seamlessly in the event of a data issue. In addition, the table is configured to be retained in the event of a stack deletion or update-replace event. Data is also encrypted at rest using an AWS managed key.
- For the model storage and upload **buckets**, bucket encryption, logging, and versioning are enabled by default. Access logging is also configured and all public access is blocked.

Vulnerability analysis and management

DeepRacer on AWS and its dependencies are continuously monitored by AWS for security vulnerabilities. Customers may choose to use Amazon Inspector or other automated vulnerability management product for monitoring their deployments.

Mitigate OS vulnerabilities with Ubuntu Pro

DeepRacer on AWS uses a container image that is based on Ubuntu 24.04 and it is possible for Common Vulnerabilities and Exposures “CVEs” to appear. Customers may at their sole option, obtain an Ubuntu Pro 24.04 license which may provide additional security assurance and follow the provided instructions linked below to modify the solution to use Ubuntu Pro 24.04 as the operating system for applicable containers.

Choosing to obtain and use Ubuntu Pro 24.04 is solely at the customer's option and is not required to use any features of DeepRacer on AWS. Obtaining the Ubuntu Pro 24.04 license may only provide access to patched versions of core Ubuntu packages and may not provide access to patched versions of third-party or open-source dependencies. Use of Ubuntu Pro 24.04 may reduce the number of CVEs reported against core packages by providing access to patched versions and does not guarantee a reduction of CVEs at any given time.

Customers are solely responsible for securing and complying with any Ubuntu Pro 24.04 licenses and AWS is not responsible for any licensing or support of Ubuntu Pro 24.04. For instructions on implementing Ubuntu Pro 24.04, please see below.

Activating Ubuntu Pro

To take advantage of ESM patches offered by Ubuntu Pro, please follow the procedure outlined below which will activate Ubuntu Pro on the SageMaker image that has been deployed for you by DeepRacer on AWS.

Prerequisites

- A laptop or cloud workstation that has access to the AWS account where DeepRacer on AWS is deployed
- Installed copies of Docker and the AWS CLI
- A valid Ubuntu Pro token

Procedure

1. Log in to access the ECR repositories in your AWS account:

```
aws ecr get-login-password --region $REGION | \
  docker login --username AWS --password-stdin $ACCOUNT_ID.dkr.ecr.
$REGION.amazonaws.com
```

2. Pull the image from the remote repository to your local machine:

```
docker pull $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/$REPOSITORY_NAME\:
$IMAGE_VERSION
```

3. Create a temporary Dockerfile with Ubuntu Pro activation commands:

```
cat << EOF > Dockerfile.ubuntu-pro
```

```
FROM $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/$REPOSITORY_NAME\:$IMAGE_VERSION

# Install ubuntu-advantage-tools and attach token
RUN apt-get update && \
    apt-get install -y ubuntu-advantage-tools && \
    ua attach $UBUNTU_PRO_TOKEN && \
    apt-get update && \
    apt full-upgrade -y && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
EOF
```

4. Build a new image with Ubuntu Pro activated:

```
docker build -t $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/$REPOSITORY_NAME\:
$IMAGE_VERSION -f Dockerfile.ubuntu-pro .
```

5. Push the new image back to your ECR repository:

```
docker push $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/$REPOSITORY_NAME\:
$IMAGE_VERSION
```

Uploaded artifacts

DeepRacer on AWS allows users to upload models downloaded from other instances to promote portability and allow submission of externally-trained models to races. All artifacts that are uploaded to DeepRacer on AWS are thoroughly scanned and validated using functions that are isolated from the rest of your AWS account using a VPC with least privilege permissions. Only after an artifact package passes these validations is it allowed to be stored in the system.

Service quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Request a service quota increase based on anticipated usage

DeepRacer on AWS uses a combination of Amazon SageMaker AI training jobs and AWS Lambda functions for servicing compute-related tasks, including but not limited to creating, training, and

evaluating models. AWS Lambda functions are also used for servicing general requests, such as creating a new account, making changes to account settings, and running/participating in races.

Considering your anticipated usage in advance and right-sizing your service quotas can improve user experience by reducing the amount of time it takes for training and evaluation jobs to be run. If your deployment is expected to serve consistently high demand (i.e. large number of users) or is subject to burst traffic (i.e. used for events, classes, workshops etc.), training and evaluation jobs will be constrained by the service quota that is in place, and those jobs will remain in the queue until capacity is available to run them.

Note

If you are operating or plan to operate multiple deployments of DeepRacer on AWS in the same account or same region (within the same account), it is important to consider the total anticipated usage across all deployments when evaluating the amount to increase the service quota by.

Note

For some services, smaller increases are automatically approved, while larger requests are submitted to AWS Support. AWS Support can approve, deny, or partially approve your requests. Larger increase requests take more time to process.

Amazon SageMaker AI training jobs

Amazon SageMaker AI training jobs are responsible for running training and evaluation jobs, and also support community races. At the time of writing, the default applied account-level quota value is 8. This means that DeepRacer on AWS can dispatch up to 8 training jobs, evaluation jobs, or race submissions (i.e. evaluating one model in a given race) at a time. If demand exceeds this limit at any point, jobs will remain queued until an actively-running job is completed and capacity becomes available. The queue for jobs in DeepRacer on AWS is managed in FIFO (first-in-first-out) order, and the amount of time that a job spends in the queue depends on the number of jobs that can be processed currently (i.e. the service quota), and the number of jobs that have entered the queue ahead of it.

As a result, it is recommended to consider in advance the number of jobs that may need to be run concurrently at any given point in time in order to deliver preferable throughput. If you decide that you would like to request a service quota increase, you may do so by:

1. Accessing the **AWS Management Console**
2. Searching for and selecting **Service Quotas** from the search bar at the top
3. Selecting **Amazon SageMaker** from the list of services
4. Searching for and selecting **ml.c5.4xlarge for training job usage** in the Service quotas table, and clicking **Request increase at account level**
5. Enter the number of jobs that you would like to be able to service concurrently into the **Increase quota value** box, and, if applicable, review it against the value provided for **Utilization**. If you are working with a new deployment, this value will most likely appear as 0.
6. When you are ready to submit the request, click **Request**.

AWS Lambda functions

AWS Lambda functions are the primary compute resource used for servicing requests in DeepRacer on AWS, including dispatching and monitoring training and evaluation jobs. Similar to that of the Amazon SageMaker AI training jobs noted in the previous section, AWS Lambda functions have an applied account-level quota of 1,000 concurrent executions. This represents the maximum number of events that functions can process simultaneously in the current region. In the event that this limit is reached, requests will be queued and serviced once capacity becomes available.

If you anticipate needing to be able to service more than 1,000 requests concurrently, requesting a service quota increase for the number of requests that you would like to be able to handle at any given point is recommended. This will allow requests to be serviced as soon as they are received, or with minimal wait time in the queue. If you decide that you would like to request a service quota increase, you may do so by:

1. Accessing the **AWS Management Console**
2. Searching for and selecting **Service Quotas** from the search bar at the top
3. Selecting **AWS Lambda** from the list of services
4. Searching for and selecting **Concurrent executions** in the Service quotas table, and clicking **Request increase at account level**

5. Enter the number of requests that you would like to be able to service concurrently into the **Increase quota value** box, and, if applicable, review it against the value provided for **Utilization**. If you are working with a new deployment, this value will most likely appear as 0.
6. When you are ready to submit the request, click **Request**.

Amazon Virtual Private Cloud (VPC)

DeepRacer on AWS configures one Amazon Virtual Private Cloud (VPC) per deployment to provide network isolation for functions that handle imported models and other user-supplied artifacts. At the time of writing, the default account-level quota is 5 VPCs per region. If you plan to host more than 5 deployments of DeepRacer on AWS in a single region, requesting a service quota increase for at least the number of deployments you expect to host is recommended.

1. Accessing the **AWS Management Console**
2. Searching for and selecting **Service Quotas** from the search bar at the top
3. Selecting **Amazon Virtual Private Cloud (VPC)** from the list of services
4. Searching for and selecting **VPCs per Region** in the Service quotas table, and clicking **Request increase at account level**
5. Enter the number of VPCs that you would like to be able to deploy into the **Increase quota value** box, and, if applicable, review it against the value provided for **Utilization**. If you are working with a new deployment, this value will most likely appear as 0.
6. When you are ready to submit the request, click **Request**.

Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the services implemented in this solution. For more information, see AWS service quotas.

Use the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

AWS Service	Documentation Link
AWS Lambda	AWS Lambda service quotas

AWS Service	Documentation Link
Amazon S3	Amazon S3 service quotas
Amazon SageMaker	Amazon SageMaker service quotas
Amazon DynamoDB	Amazon DynamoDB service quotas
Amazon API Gateway	Amazon API Gateway service quotas
Amazon CloudFront	Amazon CloudFront service quotas
Amazon Cognito	Amazon Cognito service quotas
AWS Step Functions	AWS Step Functions service quotas
Amazon SQS	Amazon SQS service quotas
Amazon Kinesis Video Streams	Amazon Kinesis Video Streams service quotas

Deploy the solution

AWS Launch Wizard is the recommended deployment method for this solution. It provides:

- A guided configuration experience with detailed help panels at each step
- A centralized page to monitor the health of all your deployments
- Indication when there is a more recent version of the solution available for deployment or upgrade

Alternatively, you can deploy the solution directly using an AWS CloudFormation template.

Overview

Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Before you launch the solution, review the cost, architecture, security, and other considerations discussed earlier in this guide.

Time to deploy: Approximately 30 minutes

> **Note**
>
> This solution includes data collection metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered through this survey. Data collection is subject to the AWS Privacy Notice.

> **Note**
>
> You are responsible for the cost of the AWS services used while running this solution. For more details, visit the Cost section in this guide and refer to the pricing webpage for each AWS service used in this solution.

Deploy using AWS Launch Wizard

This solution features a guided deployment process using AWS Launch Wizard. Follow these steps to deploy DeepRacer on AWS into your account.

1. Sign in to the AWS Management Console and select the button below to start the deployment process.

2. If there are more than one deployment patterns available for the solution, select the one that's most applicable to your use case.
3. Select a version to deploy. The latest version is recommended.
4. Click on the **Launch deployment wizard** button.

You will then follow a series of steps to collect the information needed to deploy the solution. It will take approximately 30 minutes to provision the required resources.

Select your deployment from the Deployment list to view its status.

Deploy using AWS CloudFormation

This solution uses AWS CloudFormation templates and stacks to automate its deployment. The CloudFormation template specifies the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the template.

AWS CloudFormation template

This solution includes an AWS CloudFormation template that you can download before deploying the solution.

deepracer-on-aws.template: Use this template to launch the solution and all associated components. The default configuration deploys AWS Lambda functions, Amazon API Gateway, Amazon Cognito, Amazon DynamoDB, Amazon SageMaker AI, Amazon S3 buckets, Amazon CloudFront, AWS Step Functions, Amazon CloudWatch, Amazon Elastic Container Registry, and AWS Systems Manager, but you can also customize the template based on your specific needs.

Launch the stack

This automated AWS CloudFormation template deploys DeepRacer on AWS.

1. Sign in to the AWS Management Console and select the button to launch the CloudFormation template.

   Alternatively, you can download the template as a starting point for your own implementation.

2. The template is launched in the US East (N. Virginia) region by default. To launch this solution in a different AWS region, use the region selector in the console navigation bar.

   > **Note**
   >
   > This solution uses Amazon Cognito, which is currently available in specific AWS Regions only. Therefore, you must launch this solution in an AWS Region where Amazon Cognito is available. For the most current service availability by Region, refer to the AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.

4. On the **Specify stack details** page, assign a name to your solution stack.

   > **Important**
   >
   > This solution provisions log groups that are set to be retained following stack deletion. If you have previously deployed DeepRacer on AWS, be sure to provide a unique stack name to prevent the deployment from failing due to log groups with the same name already existing.

5. Under **Parameters**, review the parameters for the template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
AdminEmail	*<Requires input>*	Email address for the initial admin user. This user will be automatically added to the admin group.
CustomDomain	<Optional input>	Custom domain URL for CORS allowlist (only if you have or plan to map a custom domain to CloudFront).
Namespace	*<Requires input>*	The namespace for this deployment of DeepRacer. Lowercase alphanumeric characters of length between 3 and 12.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Be sure to check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack.

   You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a **CREATE_COMPLETE** status in approximately five minutes.

Deploy using AWS CDK commands

You can use the AWS Cloud Development Kit (CDK) and its CLI commands to deploy the solution into your account. To do this, follow the instructions in the README.md in our GitHub repository.

Post-deployment steps

After DeepRacer on AWS has successfully deployed, complete the following steps to get started:

Set up the admin account

During deployment, you provided the email address of the individual who will serve as the admin of the deployment. An email containing temporary credentials has been sent to that email address. If you will be the admin of the deployment, click the link in the email to open the web console, and use your email address and the temporary password in the email to log in. Once logged in, you will be asked to provide an alias and a new password. Once those are provided, you will be brought to the home page for the web console.

> **Note**
>
> The invitation email will be sent from *no-reply@verificationemail.com*.

If someone else will be the admin of the deployment, they will receive an email containing temporary credentials for them to set up their admin account.

Once your/their account is set up, you/they can begin inviting users, creating races, and managing the deployment.

Set usage limits

DeepRacer on AWS uses AWS resources in order to deliver its features and intended functionality. These resources are billed as they are used. Deployments that have more users and higher training volumes will accrue higher variable costs than those with less users and lower training volumes. For more information on the cost of operating this solution, please see the Cost section.

DeepRacer on AWS allows admins to set usage limits at both the deployment level and the user level. Before inviting users to the instance, we recommend developing a plan that takes into account your budget and expected usage. You can reference the Cost and usage management section in this guide to learn more about how to apply these limits.

Invite users

You can invite users to the deployment via the Manage instance page. For more information on how to invite users, please see the Invite a user section.

After completing these initial configuration steps, you can begin using your DeepRacer on AWS deployment. For detailed information on how to use all features of the solution, see the Use the solution guide.

Use the solution

DeepRacer on AWS provides the same set of reinforcement learning fundamentals as the original AWS DeepRacer service, in a format that allows customers to host it from their own AWS accounts.

Once the solution is deployed, users interact with it exclusively through a dedicated web console. Users can train and evaluate reinforcement learning models with a virtual simulator, and submit their trained models to community races where their models will be evaluated against those submitted by other participants. The AWS DeepRacer race car is fully supported, and users can export trained models from DeepRacer on AWS, and upload them to race cars for real-life competitions.

Topics

Create an account

To create an account on DeepRacer on AWS, you must first be invited by the admin of a DeepRacer on AWS deployment. When invited, you will receive an invitation email containing a link to the DeepRacer on AWS console, and a temporary password to log in with. Click the link or paste it into your browser, and log in using your email address and the temporary password that was sent in the email.

> **Note**
>
> The invitation email will be sent from *no-reply@verificationemail.com*.

From here, you will be asked to provide an **alias**, which is similar to a username and will be used to identify you in community races, and your permanent **password**, which you will use to log in to DeepRacer on AWS going forward.

After submitting these, you will be brought to the home page, which features a quick-start tutorial and a graph that shows the amount of compute you've used and the number of models you've stored on the deployment.

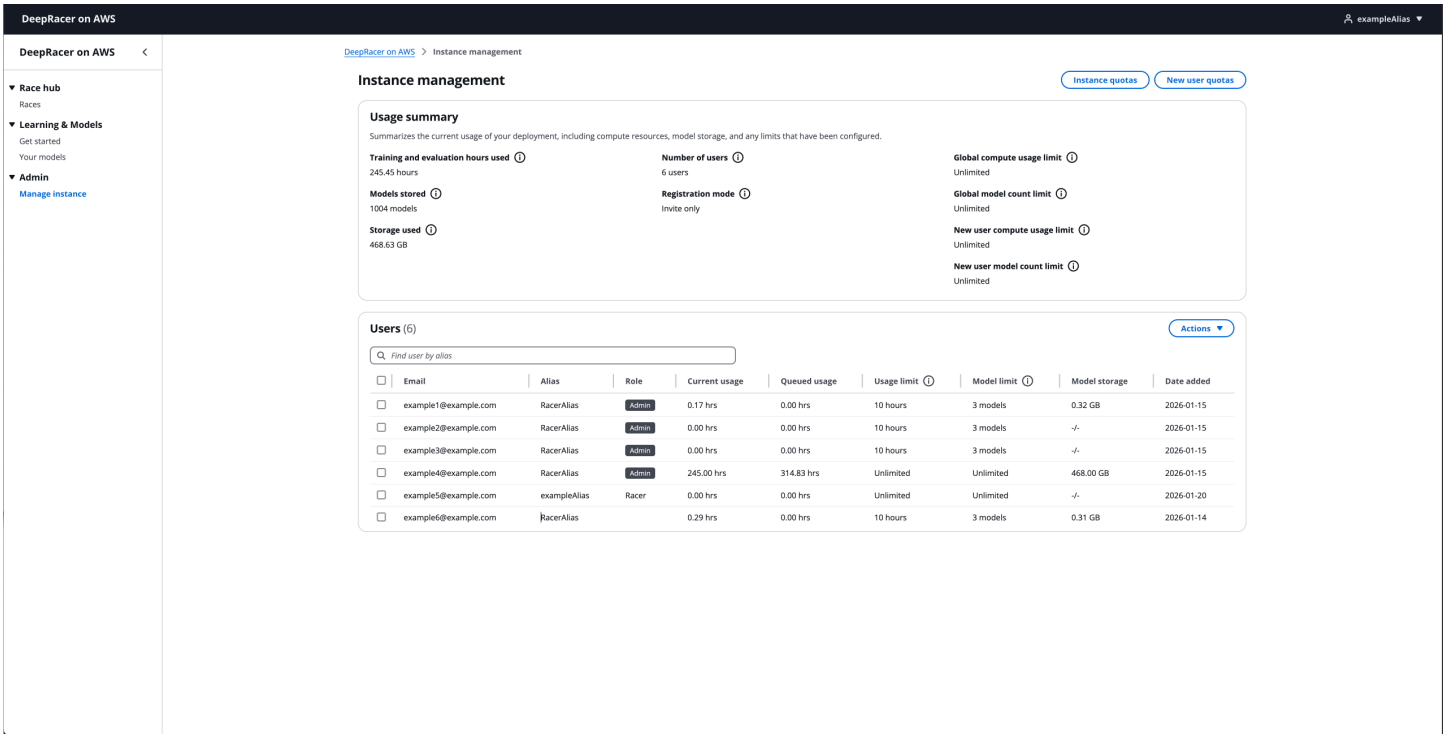
Types of users

Racers are the most common type of user in DeepRacer on AWS, and any account that is created will automatically be created as a racer. Racers can learn the fundamentals of reinforcement learning; build, train, and evaluate models; and they can submit their trained models to live races that are hosted on the deployment. They can also import or export models from the deployment, allowing them to take a model to another deployment and use it there.

Racers can be promoted by an admin to either a race facilitator or to an admin (co-admin). Race facilitators and admins, in addition to their elevated permissions, also share the same abilities as racers.

Race facilitators are users who have been given elevated permissions by the admin of a deployment to help with creating and managing races. Race facilitators inherit the same permissions as racers, allowing them to also create, train, and evaluate models, and submit them to community races. Admins can also perform the same functions as a race facilitator, allowing them to create and manage races, in addition to having their own elevated privileges.

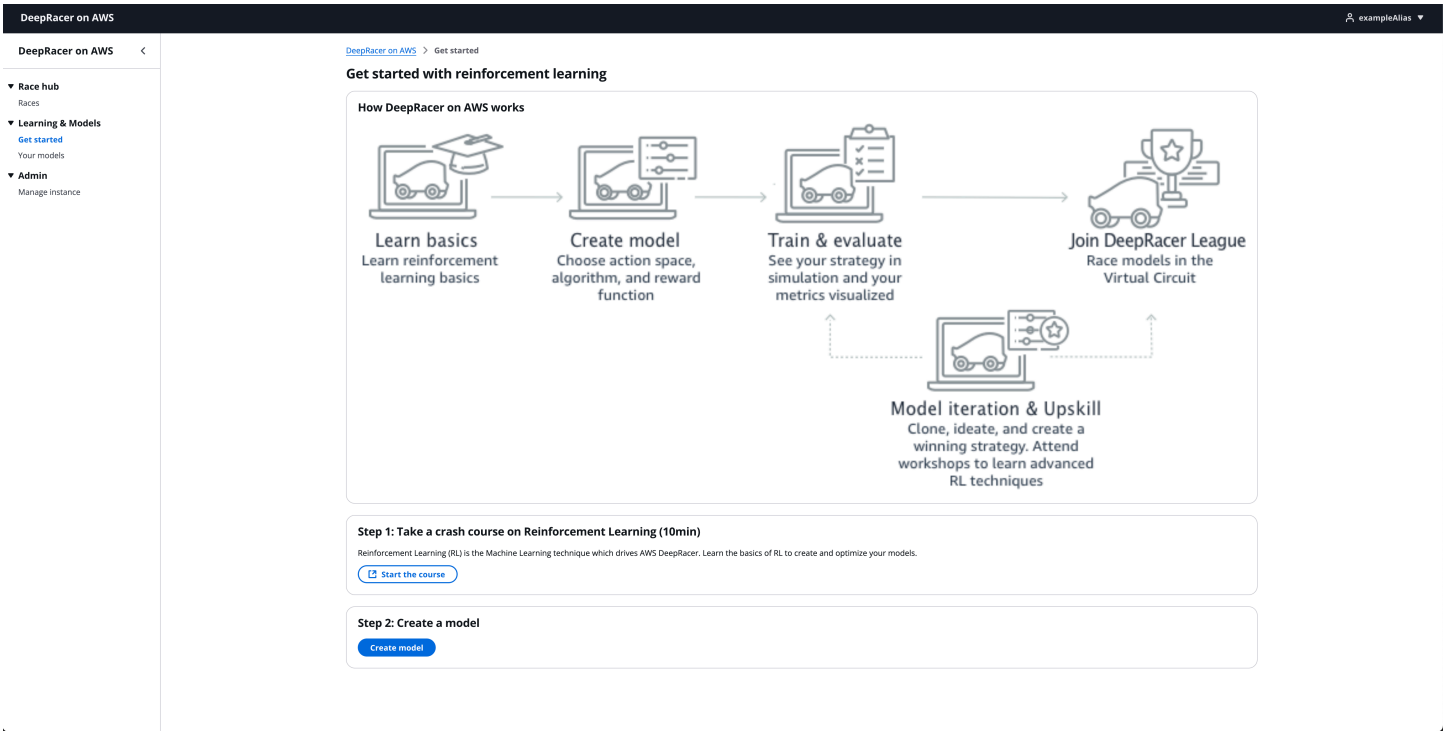
Admins have the highest level of permission on a given deployment of DeepRacer on AWS. Admins are able to manage the overall deployment, including managing users, resource utilization, and other operational aspects of the solution. Admins, in addition to their elevated permissions, also share the same abilities as both racers and race facilitators.



Take the crash course

If you are just getting acquainted with DeepRacer on AWS and reinforcement learning, we recommend taking the built-in crash course on reinforcement learning. This is a tutorial that takes about 10 min to complete, and will help you learn the basics of reinforcement learning to create and optimize your models. Once you are done with the tutorial, you will be guided through creating your first model.

From the home page, click **Get started** under Learning & Models, and click **Start the course**.



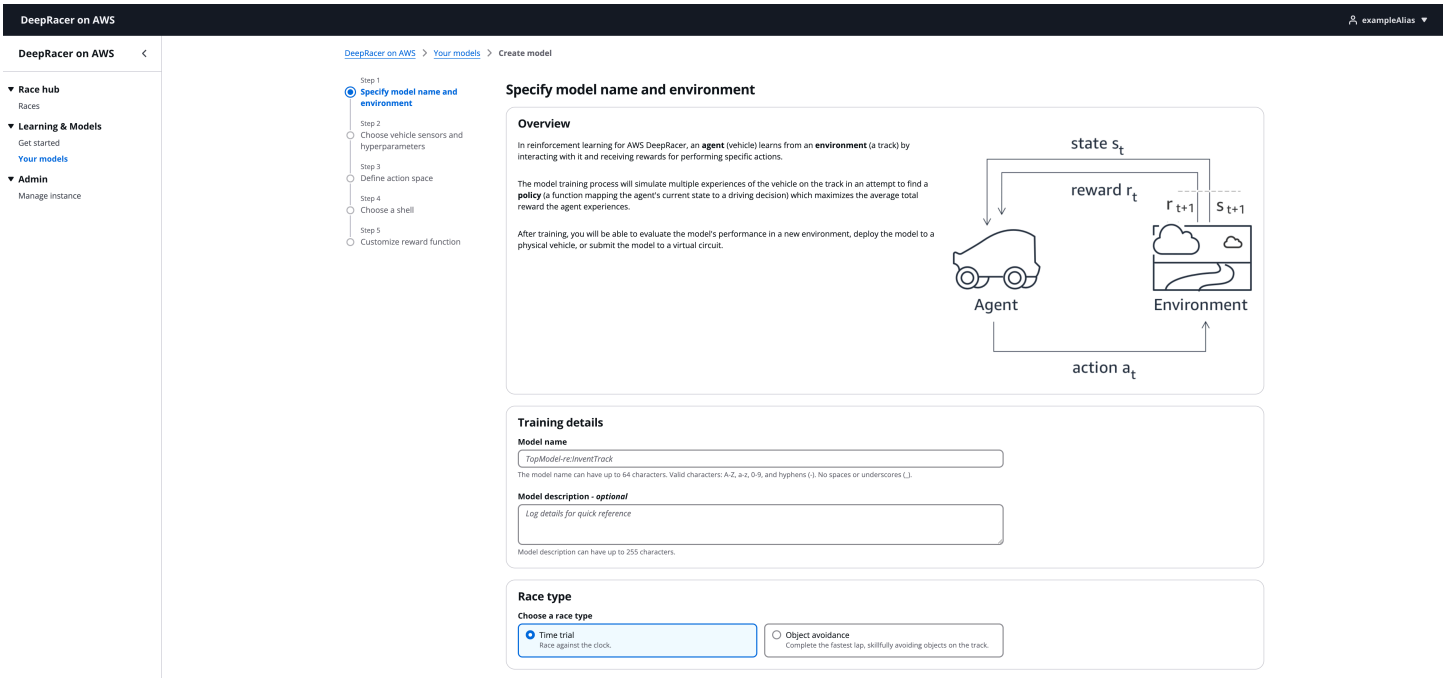
Create a model

A model is a reinforcement learning neural network that enables the race car, either virtual or physical, to make real-time driving decisions on its own. The model is created by training the vehicle through trial and error in a simulated environment, where it learns to navigate a track by optimizing for rewards defined in the reward function.

To create a model, find the **Your models** page in the left sidebar and click **Create model** in the upper right corner of the screen. You'll be brought to a multi-step wizard to help you create the model.

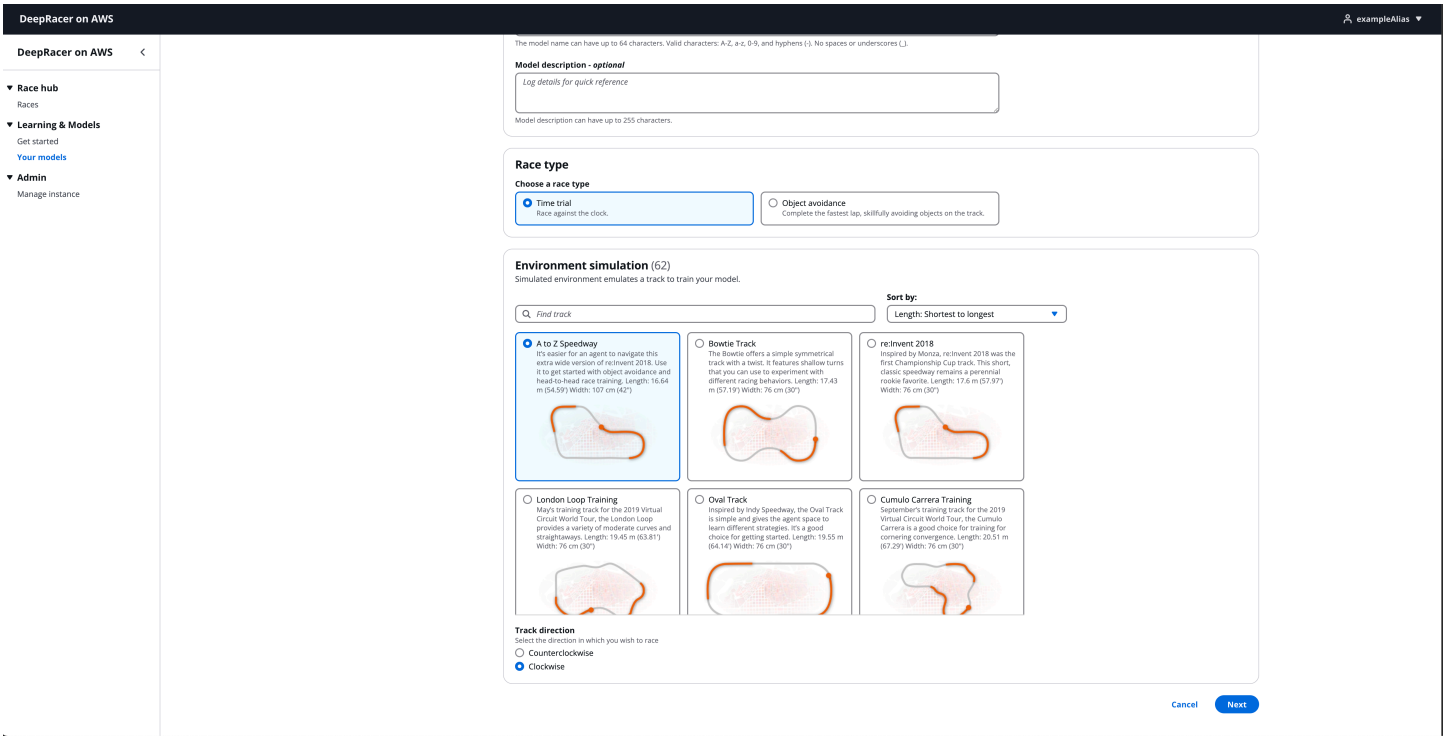
Model name and environment

First, you'll give your model a **name** and a description (optional).



Then, you'll choose a **race type** to optimize your model for.

- **Time trial** - aims to go around the track as quickly as possible, while staying on the track.
- **Object avoidance** - aims to complete the fastest lap while avoiding objects on the track.

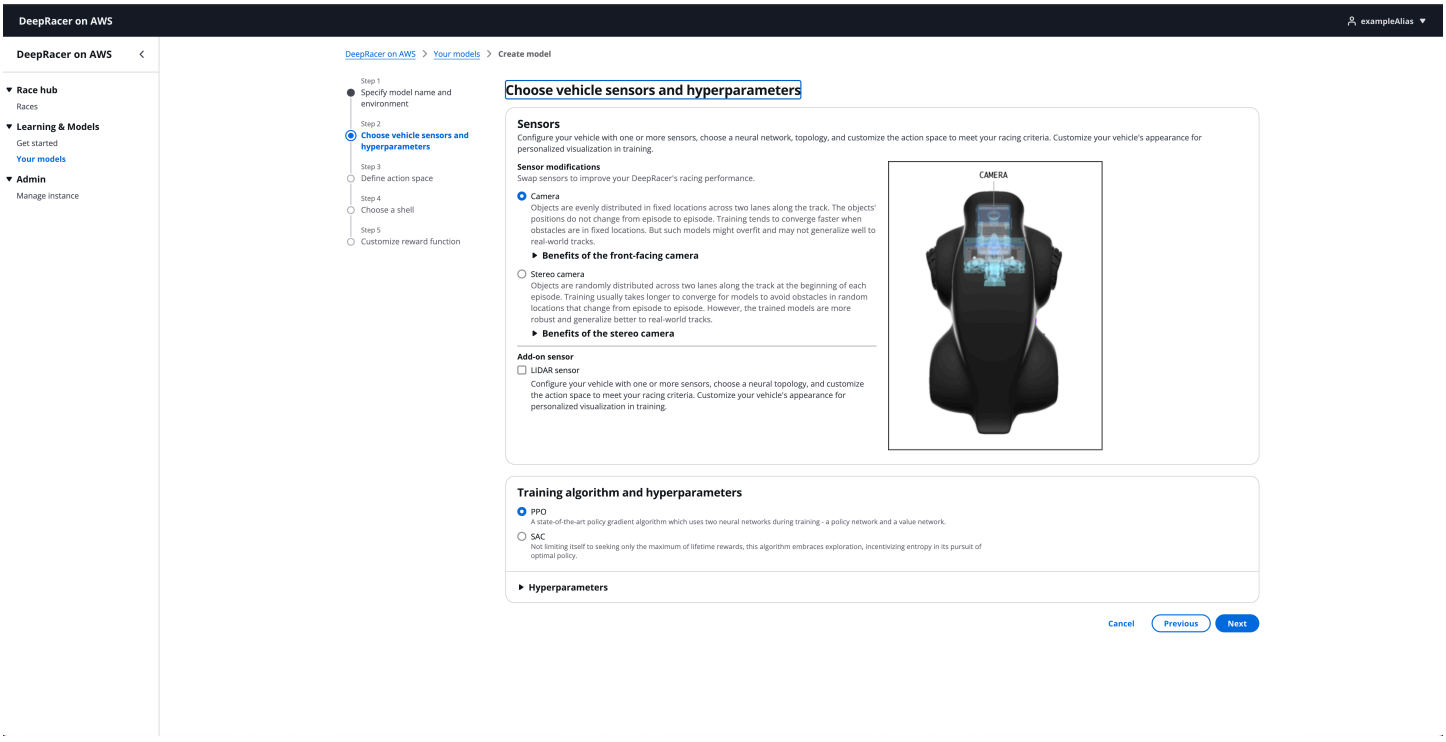


If you select object avoidance, you'll be presented with additional options that allow you to select the whether you'd like the objects to avoid to be in fixed or random locations, as well as the number of objects to place on the track. If you choose to have the objects placed in a fixed location, you can specify where you'd like them to be placed by selecting:

- **Lane placement** - whether the object is placed inside or outside of the lane.
- **Location** - a function of the % distance between the track's start and finish lines.

Next, you'll choose the **environment**, or track that your model will be trained on. DeepRacer on AWS comes with a variety of different tracks to choose from. If you plan on entering your model into a competition, it's best to train your model on the same race track that will be used for that competition. Some tracks offer the option to pick which direction the car will travel (i.e. clockwise or counterclockwise).

When you're ready to move on to the next step, click **Next**.



Vehicle sensors and hyperparameters

Sensors

Front-facing camera

A single-lens front-facing camera can capture images of the environment in front of the host vehicle, including track borders and shapes. It's the least expensive sensor and is suitable for handling simpler autonomous driving tasks, such as obstacle-free time trials on well-marked tracks. With proper training, it can avoid stationary obstacles at fixed locations on the track. However, the obstacle location information is built into the trained model and, as a result, the model is likely to be overfitted and may not generalize to other obstacle placements. With stationary objects placed at random locations or other moving vehicles on the track, the model is unlikely to converge.

In the real world, the AWS DeepRacer vehicle comes with a single-lens front-facing camera as the default sensor. The camera has a 120-degree wide angle lens and captures RGB images that are then converted to grayscale images of 160 x 120 pixels at 15 frames per second (fps). These sensor properties are preserved in the simulator to maximize the chance that the trained model transfers well from simulation to the real world.

Front-facing stereo camera

A stereo camera has two or more lenses that capture images with the same resolution and frequency. Images from both lenses are used to determine the depth of observed objects. The depth information from a stereo camera is valuable for the host vehicle to avoid crashing into obstacles or other vehicles in the front, especially in a more dynamic environment. However, added depth information makes training converge more slowly.

On the AWS DeepRacer physical vehicle, the double-lens stereo camera is constructed by adding another single-lens camera and mounting each camera on the left and right sides of the vehicle. The AWS DeepRacer software synchronizes image captures from both cameras. The captured images are converted into grayscale, stacked, and fed into the neural network for inference. The same mechanism is duplicated in the simulator in order to train the model to generalize well to a real-world environment.

LiDAR sensor

An optional LiDAR sensor uses rotating lasers to send out pulses of light outside the visible spectrum and times how long it takes each pulse to return. The direction of and distance to the

objects that a specific pulse hits are recorded as a point in a large 3D map centered around the LiDAR unit.

For example, LiDAR helps detect blind spots of the host vehicle to avoid collisions while the vehicle changes lanes. By combining LiDAR with mono or stereo cameras, you enable the host vehicle to capture sufficient information to take appropriate actions. However, a LiDAR sensor costs more compared to cameras. The neural network must learn how to interpret the LiDAR data. Thus, training will take longer to converge.

On the AWS DeepRacer physical vehicle, a LiDAR sensor is mounted on the rear and tilted down by 6 degrees. It rotates at an angular velocity of 10 rotations per second and has a range of 15cm to 2m. It can detect objects behind and beside the host vehicle as well as tall objects unobstructed by the vehicle parts in the front. The angle and range are chosen to make the LiDAR unit less susceptible to environmental noise.

Configuration options

You can configure your AWS DeepRacer vehicle with the following combination of the supported sensors:

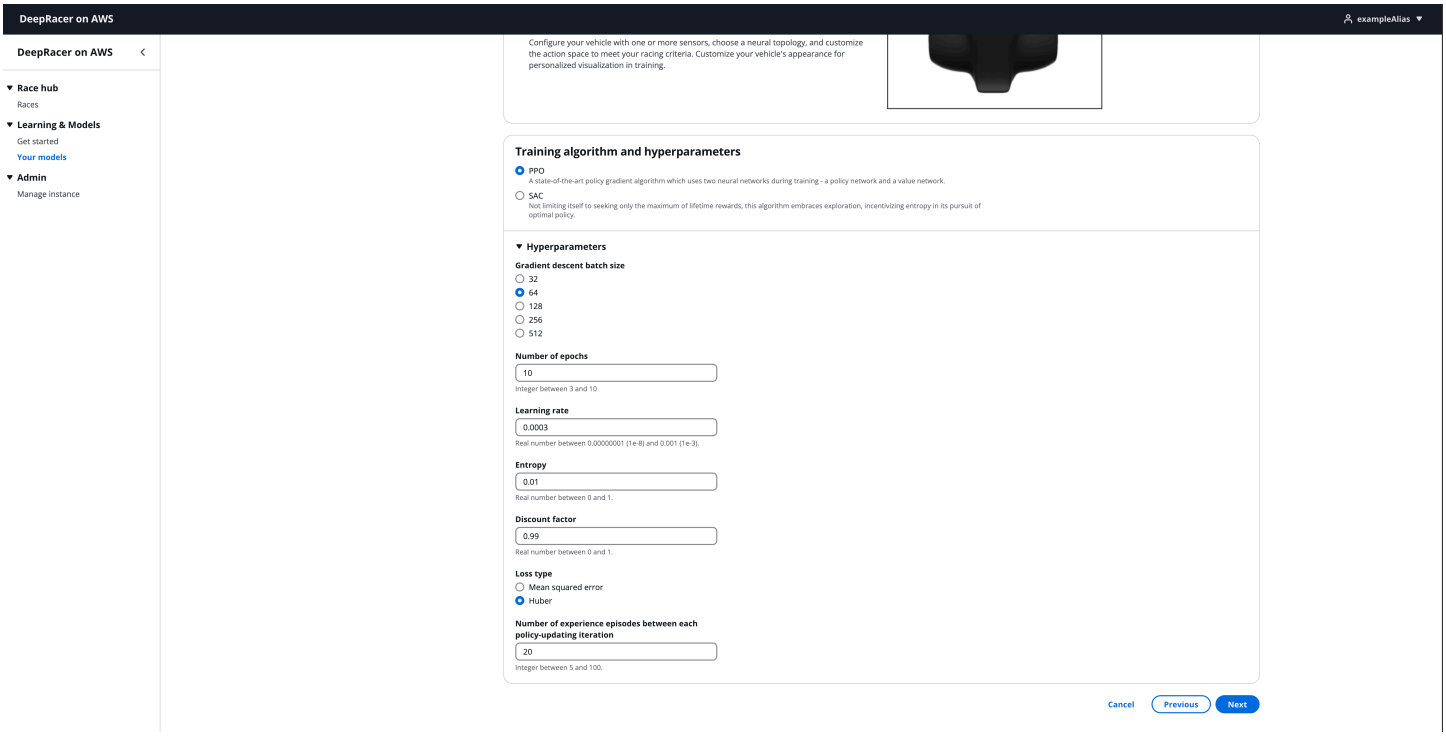
- **Front-facing single-lens camera only** - suitable for time trials, as well as obstacle avoidance with objects at fixed locations.
- **Front-facing stereo camera only** - suitable for obstacle avoidance with objects at fixed or random locations.
- **Front-facing single-lens camera w/ LiDAR** - suitable for obstacle avoidance.
- **Front-facing stereo camera w/ LiDAR** - suitable for obstacle avoidance, but probably not the most economical for time trials.

As you add more sensors to make your AWS DeepRacer vehicle transition from time trials to object avoidance, the vehicle collects more data about the environment to feed into the underlying neural network during training. This makes training more challenging because the model is required to handle increased complexities. In the end, your tasks of learning to train models become more demanding.

To learn progressively, you should start training for time trials first before moving on to object avoidance racing.

You should experiment with different sensors on your AWS DeepRacer vehicle to provide it with sufficient capabilities to observe its surroundings for a given race type. The next section describes the supported sensors that can enable the supported types of autonomous racing events.

Training algorithms and Hyperparameters



- **PPO** - proximal policy optimization - teaches the vehicle by letting it practice driving and gradually improving its skills through small, careful adjustments based on what it just learned, similar to how you might refine your technique after each practice lap.
- **SAC** - soft actor-critic - allows the vehicle to learn from both its current driving attempts and past experiences, encouraging it to try new approaches while still aiming for the best lap times, making it more flexible but requiring more fine-tuning to get right.

If you select the PPO algorithm, you'll be asked to provide:

- **Gradient descent batch size** - how many training examples the model looks at together before updating what it has learned, where looking at more examples at once gives smoother learning but takes longer to process.
- **Number of epochs** - the number of times the training algorithm passes through and updates the model using the same batch of collected experience data before gathering new experiences.

Higher epoch values allow the model to learn more thoroughly from each batch but risk over-fitting to that specific data.

- **Learning rate** - how quickly the model adjusts its driving strategy based on what it learns, where higher values mean faster learning but risk making the car's behavior unstable, while lower values mean slower but steadier improvement.
- **Entropy** - encourages the model to maintain exploration and creativity in its action selection by penalizing overly deterministic behavior, preventing the agent from prematurely settling on repetitive or sub-optimal driving patterns.
- **Loss type** - determines the function used to calculate prediction errors during training:
  - **Huber loss** - treats small mistakes gently (helping the model learn smoothly) but doesn't overreact to occasional big mistakes, making it more forgiving when the training data has some unusual or extreme values.
  - **Mean squared error** - penalizes bigger mistakes much more than smaller ones, which helps the model focus on avoiding large errors but can sometimes make it too sensitive to unusual data points.
- **Number of experience episodes between each policy-updating iteration** - specifies how many track runs the vehicle collects and stores as training data before using that batch of experiences to update the model's neural network weights, balancing data collection efficiency with training frequency.

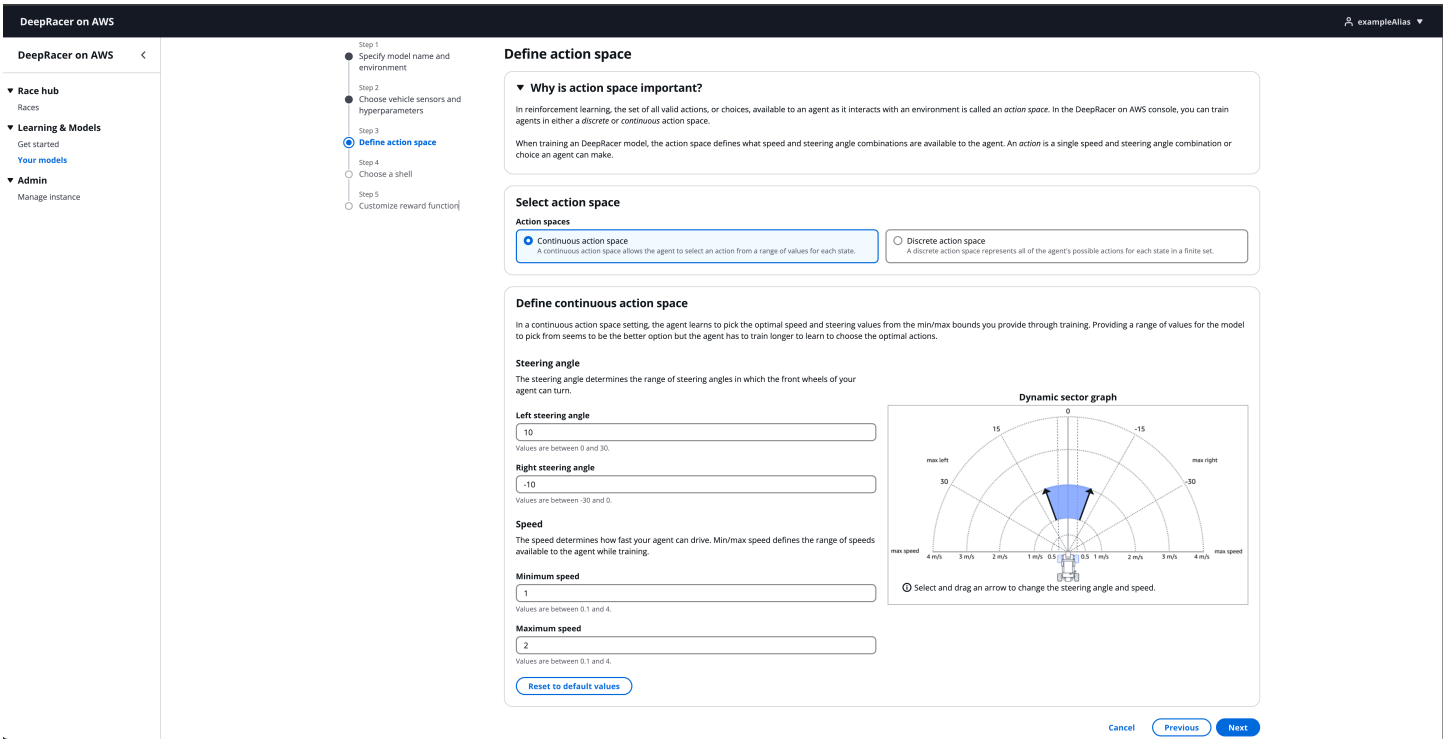
If you select the SAC algorithm, you'll be asked to provide:

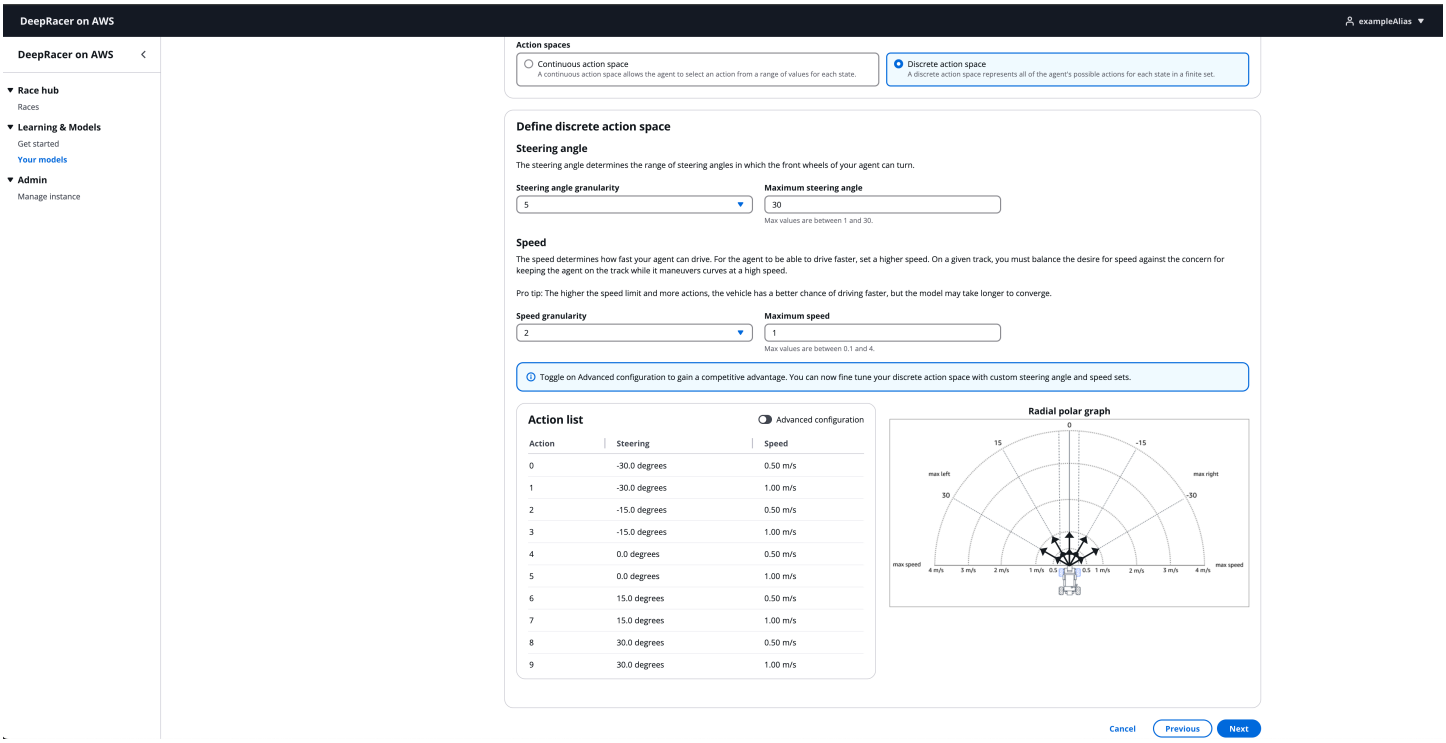
- **Gradient descent batch size** - how many training examples the model looks at together before updating what it has learned, where looking at more examples at once gives smoother learning but takes longer to process.
- **Learning rate** - how quickly the model adjusts its driving strategy based on what it learns, where higher values mean faster learning but risk making the car's behavior unstable, while lower values mean slower but steadier improvement.
- **SAC alpha value** - controls how much the model balances between trying new driving approaches versus sticking with strategies it already knows work well, helping prevent the car from getting stuck doing the same thing over and over.
- **Discount factor** - determines how much the model cares about rewards it might get in the future versus rewards it gets right now, where higher values make the car think more about long-term success (like completing the whole lap) rather than just immediate gains.

- **Loss type** - determines the function used to calculate prediction errors during training:
  - **Huber loss** - treats small mistakes gently (helping the model learn smoothly) but doesn't overreact to occasional big mistakes, making it more forgiving when the training data has some unusual or extreme values.
  - **Mean squared error** - penalizes bigger mistakes much more than smaller ones, which helps the model focus on avoiding large errors but can sometimes make it too sensitive to unusual data points.

When you're ready to move on to the next step, click **Next**.

Action space





In reinforcement learning, the **action space** is the complete set of choices available to the vehicle as it drives around the track. In the DeepRacer on AWS console, you can train your car using either a continuous action space or a discrete action space.

Continuous action spaces

In a continuous action space, the car learns to pick the best speed and steering angle from a range of values you set (i.e. choosing any speed between 1 and 4 meters per second, rather than just picking from fixed options like "slow," "medium," or "fast"). Giving the model a range of values to choose from provides more flexibility and can lead to smoother, more optimized driving, but it also means the car needs more training time to figure out which combinations of speed and steering work best for different parts of the track.

On this page, you'll start by defining the **left steering angle** and **right steering angle**. This determines how sharply the vehicle can turn its wheels. Smaller angles allow only gentle turns suitable for straighter sections, while larger angles enable the sharp turns needed for tight corners. Setting the right steering angle range is crucial because too narrow means the car can't navigate sharp curves and will drive off track, while too wide causes unnecessary zig-zagging that slows lap times.

Next, you'll select the **minimum speed** and **maximum speed**, which determines how fast the vehicle moves around the track. Lower speeds provide more control and stability through corners

but result in slower lap times, while higher speeds can complete laps faster but make it harder for the car to stay on track through turns. Setting the right speed range is important because too slow means the car will never achieve competitive lap times even if it stays on track, while too fast can cause the car to overshoot corners and drive off the track before it can react.

Discrete action spaces

A discrete action space means the car chooses from a fixed menu of specific driving options rather than picking any value it wants. You define exactly which steering-and-speed combinations are available (for example, five steering angles like sharp left, slight left, straight, slight right, and sharp right, each paired with three different speeds), and during training, the model learns which of these specific combinations work best for different parts of the track. This approach makes training simpler and faster because the car has fewer choices to evaluate, but it also means the driving behavior is limited to only the combinations you've predefined rather than being able to fine-tune to any possible steering angle and speed.

On this page, you'll start by defining the **steering angle granularity**, which is the number of different steering angle options available to the car between its minimum and maximum turning angles, where higher granularity means more steering choices (like having 7 different turn angles instead of just 3), giving the car finer control but requiring more training time to learn which option works best. Then, you'll select the **maximum steering angle** you would like your car to have, between 1 and 30 degrees.

Next, you'll select your **speed granularity**, which is the number of different speed options available to the car between its minimum and maximum speeds, where higher granularity means more speed choices (like having 5 different speeds instead of just 2), allowing the car to fine-tune its velocity for different track sections but requiring more training time to learn the optimal speed for each situation. You'll also select the **maximum speed** for the car.

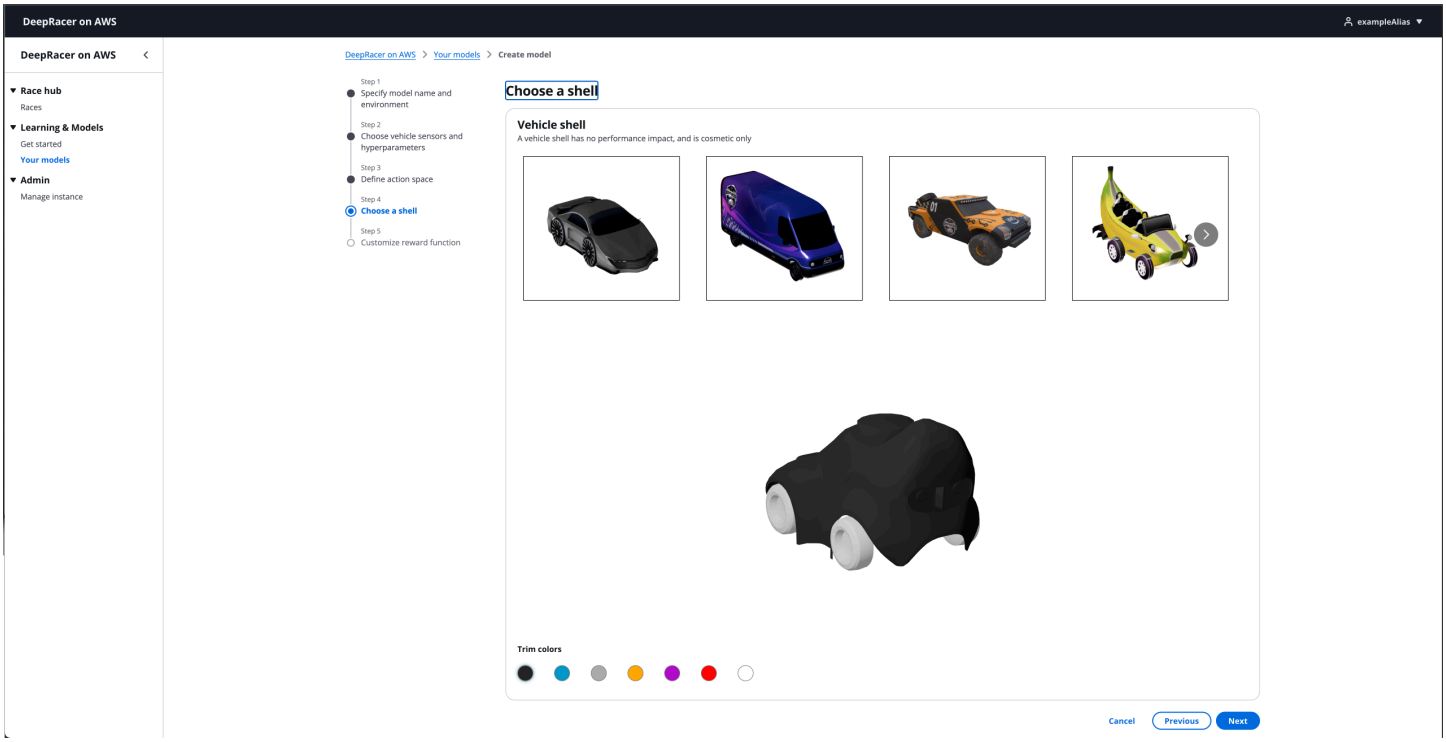
In the **Action list**, you'll see a list of actions, steering angles, and speeds populated based on your selections thus far. You can toggle the Advanced configuration switch to fine tune each action or add/remove actions as desired.

When you're ready to move on to the next step, click **Next**.

Vehicle shell

On this page, you'll select a **vehicle shell**, which has no performance impact and is cosmetic only. This will be what you see navigating around the track in training, evaluations, and races. You can pick different types of shells and different colors for those shells.

When you're ready to move on to the next step, click **Next**.



Reward function

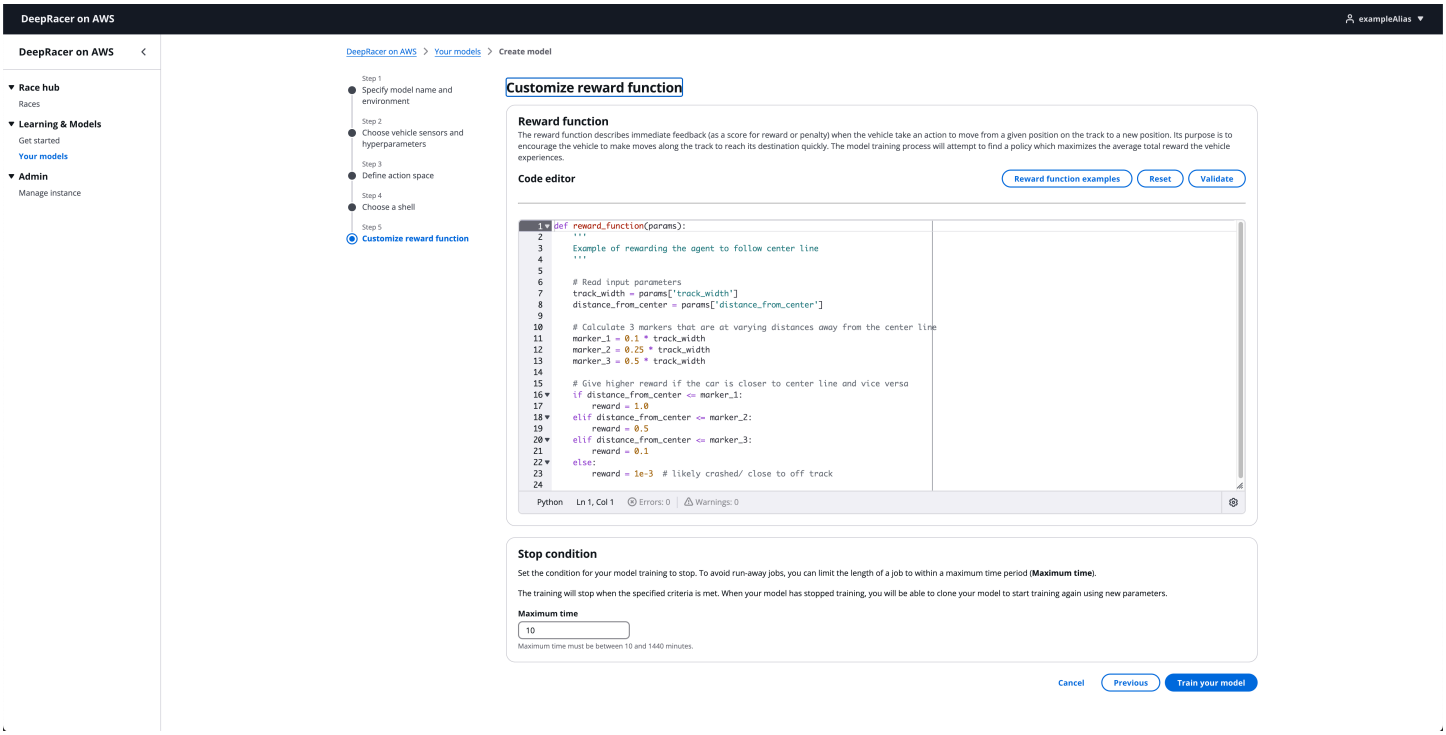
The reward function describes immediate feedback (as a score for reward or penalty) when the vehicle take an action to move from a given position on the track to a new position. Its purpose is to encourage the vehicle to make moves along the track to reach its destination quickly. The model training process will attempt to find a policy which maximizes the average total reward the vehicle experiences.

In the editor, you can see a selection of sample reward functions by clicking **Reward function examples**. Reward functions are defined as Python code, and you can validate your code at any point by clicking **Validate**. Alternatively, if you would like to reset the code that's in the editor to the default code, you can click **Reset**.

Once you have your reward function defined, you will be asked to set a stop condition. This is the condition for your model training to stop. To avoid run-away jobs, you can limit the length of a job to within a maximum time period (**Maximum time**).

The training will stop when the specified criteria is met. When your model has stopped training, you will be able to clone your model to start training again using new parameters.

When you're ready to proceed, click **Train your model**.



Input parameters

The AWS DeepRacer reward function takes a dictionary object as the input.

```
def reward_function(params) :

    reward = ...

    return float(reward)
```

The `params` dictionary object contains the following key-value pairs:

```
{
    "all_wheels_on_track": Boolean,        # flag to indicate if the agent is on the
 track
    "x": float,                            # agent's x-coordinate in meters
    "y": float,                            # agent's y-coordinate in meters
    "closest_objects": [int, int],         # zero-based indices of the two closest
 objects to the agent's current position of (x, y).
    "closest_waypoints": [int, int],       # indices of the two nearest waypoints.
    "distance_from_center": float,         # distance in meters from the track center
    "is_crashed": Boolean,                 # Boolean flag to indicate whether the agent
 has crashed.
```

```
    "is_left_of_center": Boolean,          # Flag to indicate if the agent is on the
 left side to the track center or not.
    "is_offtrack": Boolean,                # Boolean flag to indicate whether the agent
 has gone off track.
    "is_reversed": Boolean,                # flag to indicate if the agent is driving
 clockwise (True) or counter clockwise (False).
    "heading": float,                      # agent's yaw in degrees
    "objects_distance": [float, ],         # list of the objects' distances in meters
 between 0 and track_length in relation to the starting line.
    "objects_heading": [float, ],          # list of the objects' headings in degrees
 between -180 and 180.
    "objects_left_of_center": [Boolean, ], # list of Boolean flags indicating whether
 elements' objects are left of the center (True) or not (False).
    "objects_location": [(float, float),], # list of object locations [(x,y), ...].
    "objects_speed": [float, ],            # list of the objects' speeds in meters per
 second.
    "progress": float,                     # percentage of track completed
    "speed": float,                        # agent's speed in meters per second (m/s)
    "steering_angle": float,               # agent's steering angle in degrees
    "steps": int,                          # number steps completed
    "track_length": float,                 # track length in meters.
    "track_width": float,                  # width of the track
    "waypoints": [(float, float), ]        # list of (x,y) as milestones along the
 track center

}
```

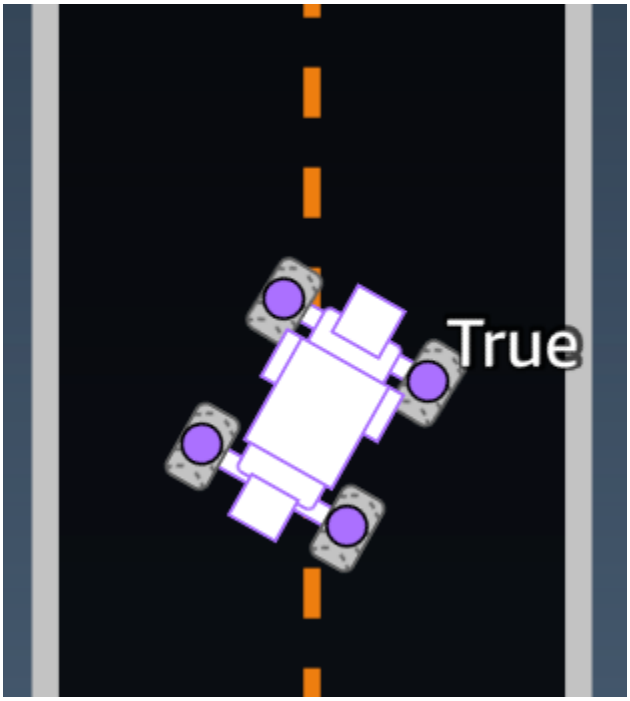
A more detailed technical reference of the input parameters is as follows.

all_wheels_on_track

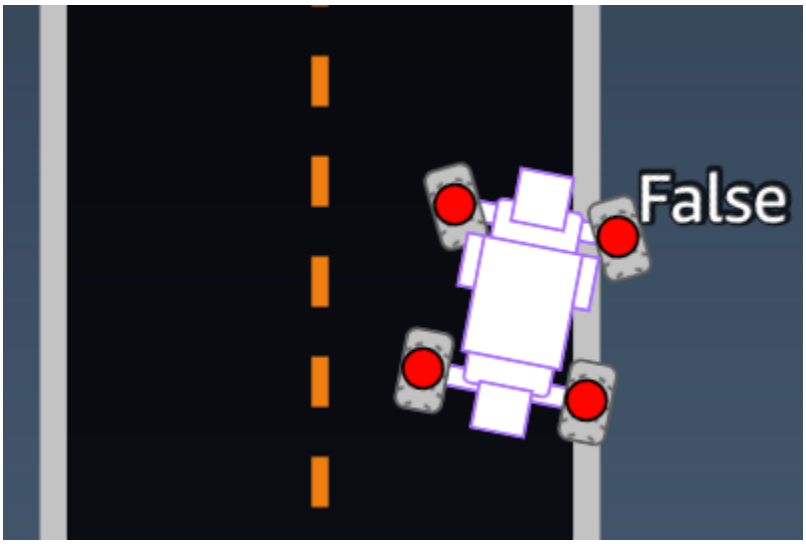
Type: `Boolean`

Range: `(True:False)`

A `Boolean` flag to indicate whether the agent is on-track or off-track. It's off-track (`False`) if any of its wheels are outside of the track borders. It's on-track (`True`) if all of the wheels are inside the two track borders. The following illustration shows that the agent is on-track.



The following illustration shows that the agent is off-track.



Example: *A reward function using the `all_wheels_on_track` parameter*

```
def reward_function(params):
    ################################################################################
    '''
    Example of using all_wheels_on_track and speed
    '''

    # Read input variables
```

```
    all_wheels_on_track = params['all_wheels_on_track']
    speed = params['speed']

    # Set the speed threshold based your action space
    SPEED_THRESHOLD = 1.0

    if not all_wheels_on_track:
        # Penalize if the car goes off track
        reward = 1e-3
    elif speed < SPEED_THRESHOLD:
        # Penalize if the car goes too slow
        reward = 0.5
    else:
        # High reward if the car stays on track and goes fast
        reward = 1.0

    return float(reward)
```

closest_waypoints

Type: `[int, int]`

Range: `[(0:Max-1),(1:Max-1)]`

The zero-based indices of the two neighboring `waypoints` closest to the agent's current position of `(x, y)`. The distance is measured by the Euclidean distance from the center of the agent. The first element refers to the closest waypoint behind the agent and the second element refers the closest waypoint in front of the agent. `Max` is the length of the waypoints list. In the illustration shown in waypoints, the `closest_waypoints` would be `[16, 17]`.

Example: A reward function using the `closest_waypoints` parameter.

The following example reward function demonstrates how to use `waypoints` and `closest_waypoints` as well as `heading` to calculate immediate rewards.

DeepRacer on AWS supports the following libraries: `math`, `random`, `NumPy`, `SciPy`, and `Shapely`. To use one, add an import statement above your function definition for the library you would like to use.

```
# Place import statement outside of function (supported libraries: math, random, numpy,
 scipy, and shapely)
# Example imports of available libraries
#
```

```
# import math
# import random
# import numpy
# import scipy
# import shapely

import math

def reward_function(params):
    #############################################################################
    '''
    Example of using waypoints and heading to make the car point in the right direction
    '''

    # Read input variables
    waypoints = params['waypoints']
    closest_waypoints = params['closest_waypoints']
    heading = params['heading']

    # Initialize the reward with typical value
    reward = 1.0

    # Calculate the direction of the center line based on the closest waypoints
    next_point = waypoints[closest_waypoints[1]]
    prev_point = waypoints[closest_waypoints[0]]

    # Calculate the direction in radius, arctan2(dy, dx), the result is (-pi, pi) in
 radians
    track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] -
 prev_point[0])
    # Convert to degree
    track_direction = math.degrees(track_direction)

    # Calculate the difference between the track direction and the heading direction of
 the car
    direction_diff = abs(track_direction - heading)
    if direction_diff > 180:
        direction_diff = 360 - direction_diff

    # Penalize the reward if the difference is too large
    DIRECTION_THRESHOLD = 10.0
    if direction_diff > DIRECTION_THRESHOLD:
        reward *= 0.5
```

```
    return float(reward)
```

closest_objects

Type: `[int, int]`

Range: `[(0:len(objects_location)-1), (0:len(objects_location)-1)]`

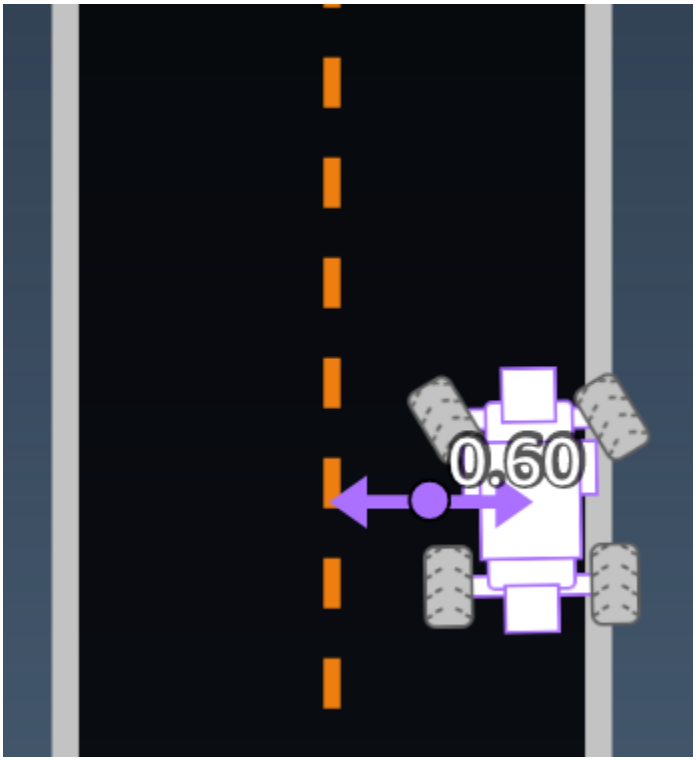
The zero-based indices of the two closest objects to the agent's current position of (x, y). The first index refers to the closest object behind the agent, and the second index refers to the closest object in front of the agent. If there is only one object, both indices are 0.

distance_from_center

Type: `float`

Range: `0:~track_width/2`

Displacement, in meters, between the agent center and the track center. The observable maximum displacement occurs when any of the agent's wheels are outside a track border and, depending on the width of the track border, can be slightly smaller or larger than half the `track_width`.



Example: *A reward function using the `distance_from_center` parameter*

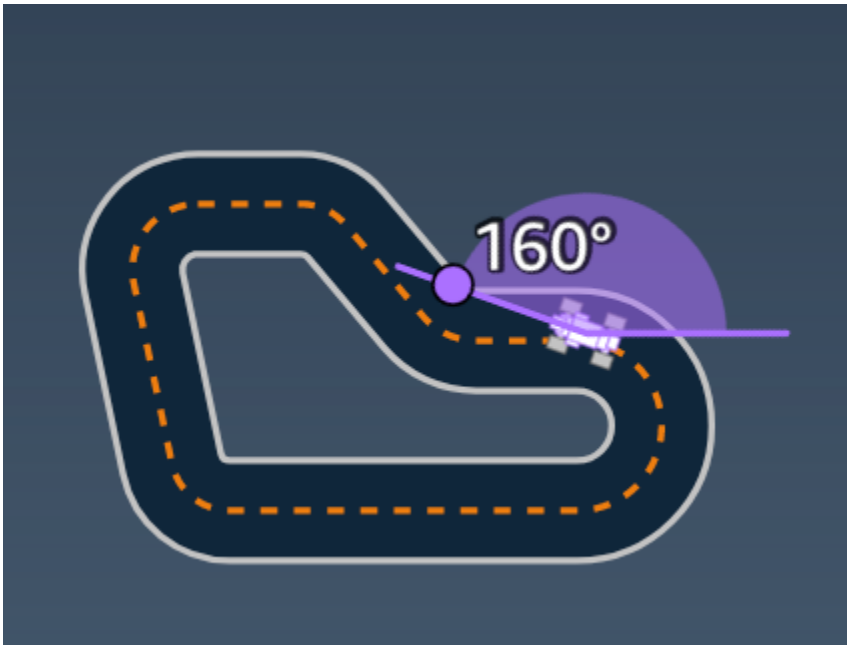
```
def reward_function(params):
    ################################################################################
    '''
    Example of using distance from the center
    '''

    # Read input variable
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Penalize if the car is too far away from the center
    marker_1 = 0.1 * track_width
    marker_2 = 0.5 * track_width

    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    else:
        reward = 1e-3  # likely crashed/ close to off track

    return float(reward)
```

heading

Type: `float`

Range: `-180:+180`

Heading direction, in degrees, of the agent with respect to the x-axis of the coordinate system.



Example: See the closest_waypoints reward function for an example that uses the `heading` parameter.

is_crashed

Type: `Boolean`

Range: `(True:False)`

A Boolean flag to indicate whether the agent has crashed into another object (`True`) or not (`False`) as a termination status.

is_left_of_center

Type: `Boolean`

Range: `(True:False)`

A Boolean flag to indicate if the agent is on the left side to the track center (`True`) or on the right side (`False`).

is_offtrack

Type: `Boolean`

Range: `(True:False)`

A Boolean flag to indicate whether the agent has off track (`True`) or not (`False`) as a termination status.

is_reversed

Type: `Boolean`

Range: `(True:False)`

A Boolean flag to indicate if the agent is driving on clock-wise (`True`) or counter clock-wise (`False`).

It's used when you enable direction change for each episode.

objects_distance

Type: `[float, … ]`

Range: `[(0:track_length), … ]`

A list of the distances between objects in the environment in relation to the starting line. The ith element measures the distance in meters between the ith object and the starting line along the track center line.

Note: abs | (var1) - (var2)| = how close the car is to an object, WHEN var1 = ["objects_distance"] [index] and var2 = params["progress"]*params["track_length"]

To get an index of the closest object in front of the vehicle and the closest object behind the vehicle, use the `closest_objects` parameter.

objects_heading

Type: `[float, … ]`

Range: `[(-180:180), … ]`

List of the headings of objects in degrees. The ith element measures the heading of the ith object. For stationary objects, their headings are 0. For a bot vehicle, the corresponding element's value is the vehicle's heading angle.

objects_left_of_center

Type: `[Boolean, … ]`

Range: `[True|False, … ]`

List of Boolean flags. The ith element value indicates whether the ith object is to the left (`True`) or right (`False`) side of the track center.

objects_location

Type: `[(x,y), … ]`

Range: `[(0:N,0:N), … ]`

List of all object locations, each location is a tuple of [(x, y)].

The size of the list equals the number of objects on the track. Note the object could be the stationary obstacles, moving bot vehicles.

objects_speed

Type: `[float, … ]`

Range: `[[(0:12.0), … ], … ]`

List of speeds (meters per second) for the objects on the track. For stationary objects, their speeds are 0. For a bot vehicle, the value is the speed you set in training.

progress

Type: `float`

Range: `0:100`

Percentage of track completed.

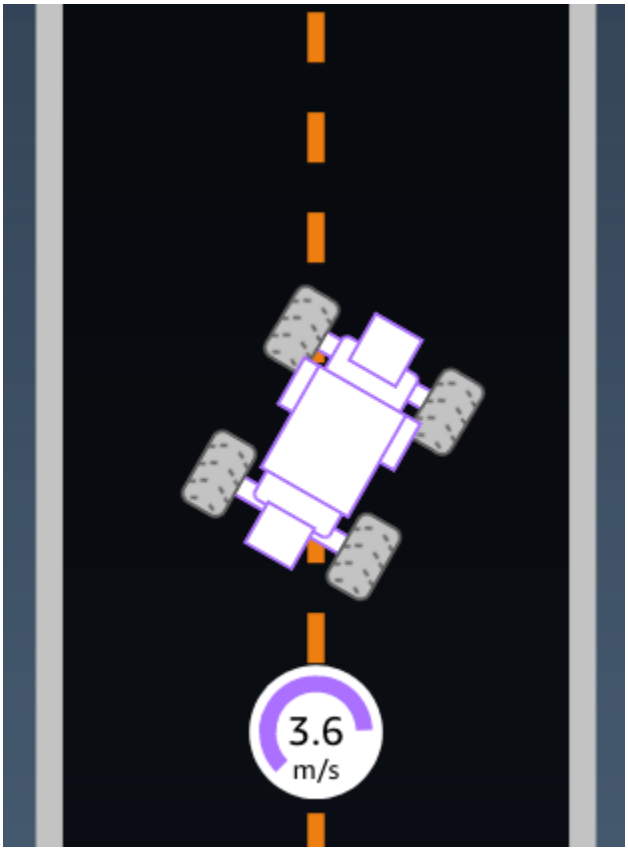
Example: See the steps example for a reward function that uses the `progress` parameter.

speed

Type: `float`

Range: `0.0:5.0`

The observed speed of the agent, in meters per second (m/s).



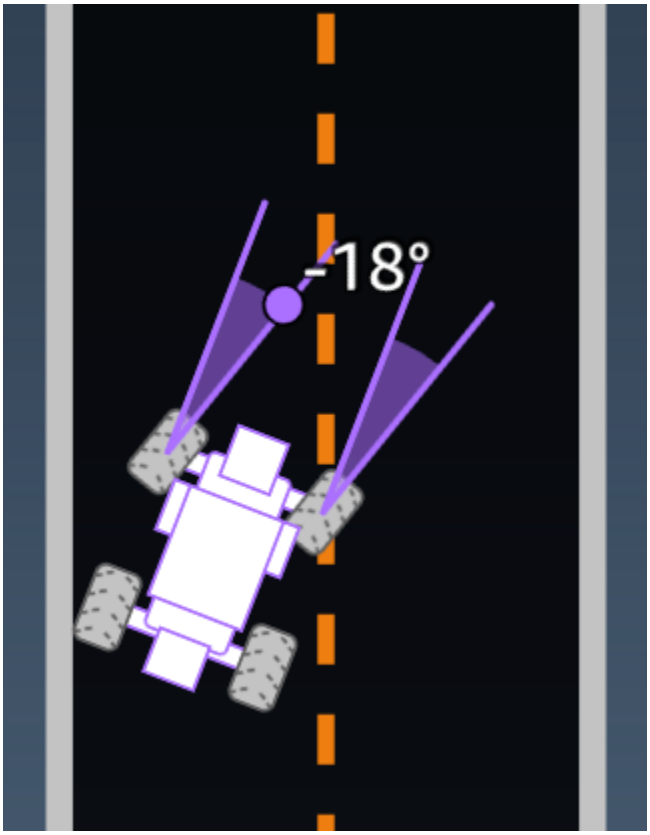
Example: See the all_wheels_on_track example for a reward function that uses the `progress` parameter.

steering_angle

Type: `float`

Range: `-30:30`

Steering angle, in degrees, of the front wheels from the center line of the agent. The negative sign (-) means steering to the right and the positive (+) sign means steering to the left. The agent center line is not necessarily parallel with the track center line as is shown in the following illustration.



Example: *A reward function using the `steering_angle` parameter*

```
def reward_function(params):
    '''
    Example of using steering angle
    '''

    # Read input variable
    abs_steering = abs(params['steering_angle']) # We don't care whether it is left or
 right steering

    # Initialize the reward with typical value
    reward = 1.0

    # Penalize if car steer too much to prevent zigzag
    ABS_STEERING_THRESHOLD = 20.0
    if abs_steering > ABS_STEERING_THRESHOLD:
        reward *= 0.8

    return float(reward)
```

steps

Type: `int`

Range: `0:Nstep`

Number of steps completed. A step corresponds to an action taken by the agent following the current policy.

Example: *A reward function using the `steps` parameter*

```
def reward_function(params):
    #############################################################################
    '''
    Example of using steps and progress
    '''

    # Read input variable
    steps = params['steps']
    progress = params['progress']

    # Total num of steps we want the car to finish the lap, it will vary depends on the
 track length
    TOTAL_NUM_STEPS = 300

    # Initialize the reward with typical value
    reward = 1.0

    # Give additional reward if the car pass every 100 steps faster than expected
    if (steps % 100) == 0 and progress > (steps / TOTAL_NUM_STEPS) * 100 :
        reward += 10.0

    return float(reward)
```

track_length

Type: `float`

Range: `[0:Lmax]`

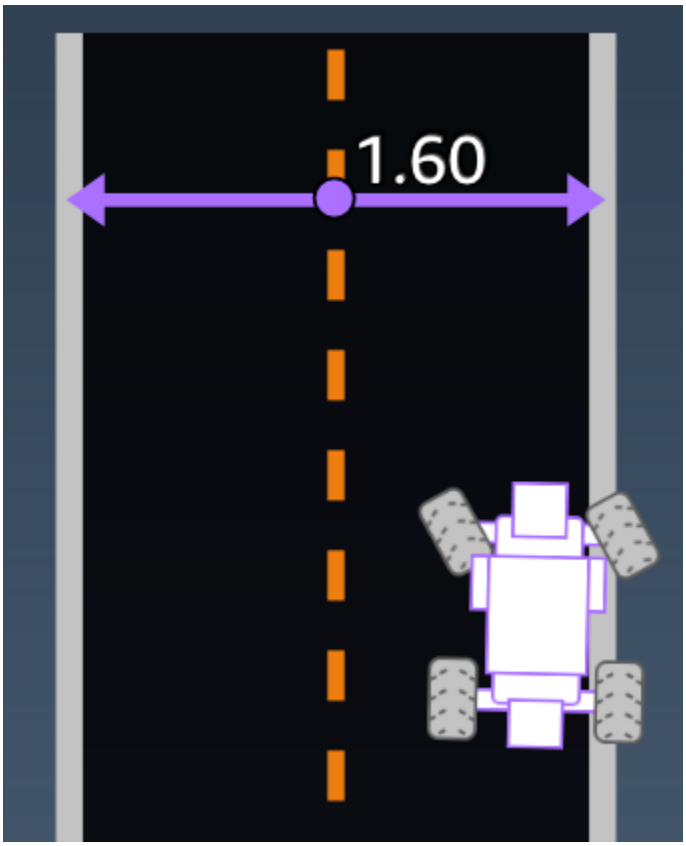
The track length in meters. `Lmax` is track-dependent.

track_width

Type: `float`

Range: `0:Dtrack`

Track width in meters.



Example: *A reward function using the `track_width` parameter*

```
def reward_function(params):
    ################################################################################
    '''
    Example of using track width
    '''

    # Read input variable
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Calculate the distance from each border
    distance_from_border = 0.5 * track_width - distance_from_center
```

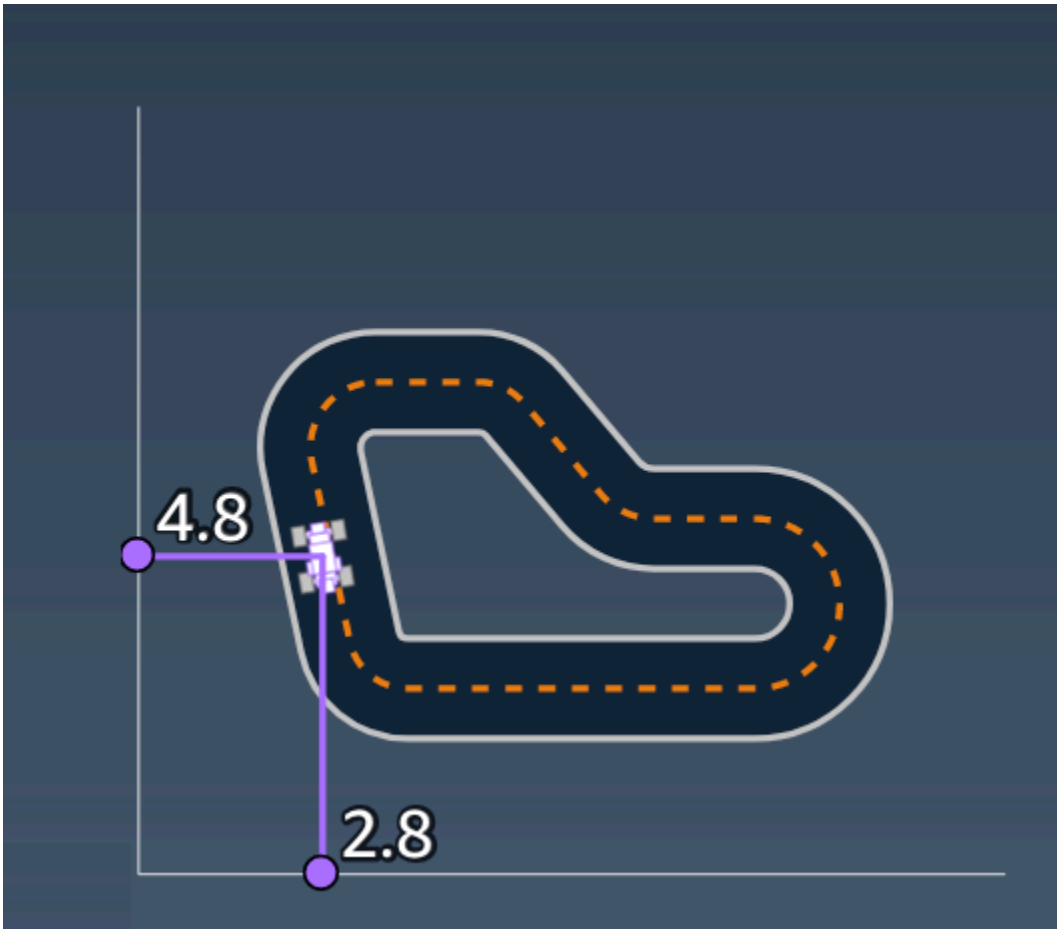
```
    # Reward higher if the car stays inside the track borders
    if distance_from_border >= 0.05:
        reward = 1.0
    else:
        reward = 1e-3 # Low reward if too close to the border or goes off the track

    return float(reward)
```

x, y

Type: `float`

Range: `0:N`

Location, in meters, of the agent center along the x and y axes, of the simulated environment containing the track. The origin is at the lower-left corner of the simulated environment.

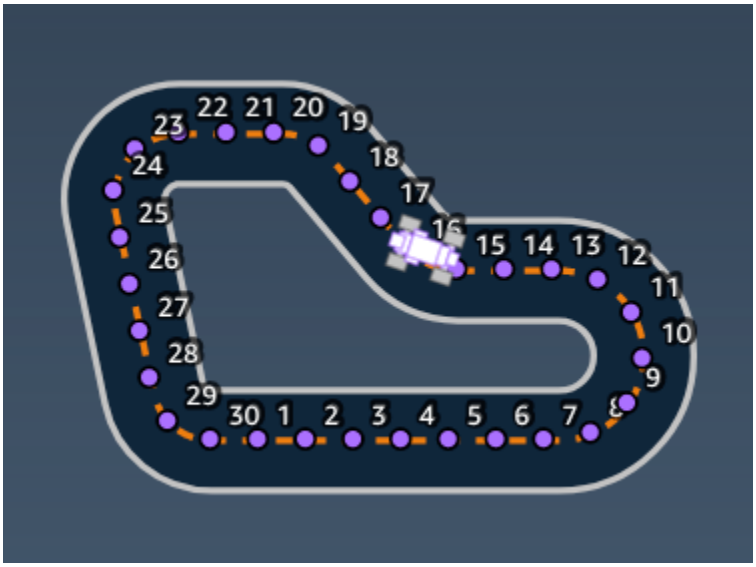


waypoints

Type: `list` of `[float, float]`

Range:

An ordered list of track-dependent `Max` milestones along the track center. For a looped track, the first and last waypoints are the same. For a straight or other non-looped track, the first and last waypoints are different.



Sample reward functions

Example 1: Follow the center line in time trials

This example determines how far away the agent is from the center line, and gives higher reward if it is closer to the center of the track, encouraging the agent to closely follow the center line.

```
def reward_function(params):
    '''
    Example of rewarding the agent to follow center line
    '''

    # Read input parameters
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Calculate 3 markers that are increasingly further away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
```

```
        reward = 1
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    return reward
```

Example 2: Stay inside the two borders in time trials

This example simply gives high rewards if the agent stays inside the borders, and lets the agent figure out the best path to finish a lap. It's easy to program and understand, but likely takes longer to converge.

```
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside the two borders of the track
    '''

    # Read input parameters
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']

    # Give a very low reward by default
    reward = 1e-3

    # Give a high reward if no wheels go off the track and
    # the car is somewhere in between the track borders
    if all_wheels_on_track and (0.5*track_width - distance_from_center) >= 0.05:
        reward = 1.0

    # Always return a float value
    return reward
```

Example 3: Prevent zig-zag in time trials

This example incentivizes the agent to follow the center line but penalizes with lower reward if it steers too much, which helps prevent zig-zag behavior. The agent learns to drive smoothly in the simulator and likely keeps the same behavior when deployed to the physical vehicle.

```
def reward_function(params):
    '''
    Example of penalize steering, which helps mitigate zig-zag behaviors
    '''

    # Read input parameters
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    abs_steering = abs(params['steering_angle']) # Only need the absolute steering
 angle

    # Calculate 3 marks that are farther and father away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    # Steering penality threshold, change the number based on your action space setting
    ABS_STEERING_THRESHOLD = 15

    # Penalize reward if the car is steering too much
    if abs_steering > ABS_STEERING_THRESHOLD:
        reward *= 0.8

    return float(reward)
```

Example 4: Stay in one lane without crashing into stationary obstacles or moving vehicles

This reward function rewards the agent for staying inside the track's borders and penalizes the agent for getting too close to objects in front of it. The agent can move from lane to lane to avoid crashes. The total reward is a weighted sum of the reward and penalty. The example gives more weight to the penalty in effort to avoid crashes. Experiment with different averaging weights to train for different behavior outcomes.

```
import math
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside two borders
    and penalizing getting too close to the objects in front
    '''
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    objects_location = params['objects_location']
    agent_x = params['x']
    agent_y = params['y']
    _, next_object_index = params['closest_objects']
    objects_left_of_center = params['objects_left_of_center']
    is_left_of_center = params['is_left_of_center']

    # Initialize reward with a small number but not zero
    # because zero means off-track or crashed
    reward = 1e-3

    # Reward if the agent stays inside the two borders of the track
    if all_wheels_on_track and (0.5 * track_width - distance_from_center) >= 0.05:
        reward_lane = 1.0
    else:
        reward_lane = 1e-3

    # Penalize if the agent is too close to the next object
    reward_avoid = 1.0

    # Distance to the next object
    next_object_loc = objects_location[next_object_index]
    distance_closest_object = math.sqrt((agent_x - next_object_loc[0])**2 + (agent_y -
 next_object_loc[1])**2)

    # Decide if the agent and the next object is on the same lane
    is_same_lane = objects_left_of_center[next_object_index] == is_left_of_center
    if is_same_lane:
        if 0.5 <= distance_closest_object < 0.8:
            reward_avoid *= 0.5
        elif 0.3 <= distance_closest_object < 0.5:
            reward_avoid *= 0.2
        elif distance_closest_object < 0.3:
            reward_avoid = 1e-3  # Likely crashed
```

```
    # Calculate reward by putting different weights on
    # the two aspects above
    reward += 1.0 * reward_lane + 4.0 * reward_avoid
    return reward
```

Train and evaluate models

Training your model involves the vehicle learning optimal driving behaviors through trial and error in a simulated environment. Your vehicle uses its front-mounted camera to capture environmental states and responds with actions (speed and steering combinations). Your model is a neural network function that maps these observations to expected rewards, with training focused on maximizing cumulative rewards.

The reward function defines the criteria for optimal behavior. A simple example might give 0 points for staying on track, -1 for going off track, and +1 for finishing. More sophisticated reward functions can encourage faster driving by rewarding smaller steering corrections at higher speeds, or penalizing aggressive turns that might cause the vehicle to leave the track.

Training occurs through repeated episodes from start to finish, where the agent learns by maximizing expected cumulative rewards. DeepRacer on AWS supports **Proximal Policy Optimization (PPO)** and **Soft Actor Critic (SAC)** algorithms, integrated with SageMaker AI and TensorFlow frameworks.

Model training is inherently iterative. Since it's difficult to define perfect reward functions initially and hyperparameters require tuning, start with simple reward functions and progressively enhance them. Use the clone feature to build upon previous models, systematically adjusting variables and hyperparameters until results converge.

Evaluation is essential before deploying to physical vehicles. The simulator provides metrics on track completion rates, lap times, and performance comparisons via leaderboards, helping you assess model effectiveness before real-world deployment.

Training for time trial races

Start with time trials if you're new to DeepRacer on AWS or the physical vehicle. This provides a gentle introduction to reward functions, agents, and environments. Your goal: train a model to stay on track and complete laps quickly, then deploy it to your physical DeepRacer for testing.

Begin by creating a model with the default configuration (single front-facing camera, default reward function). Starting with defaults helps establish fundamentals before advancing to complex configurations.

Training progression:

1. **Initial training**: Use a simple track with regular shapes and minimal sharp turns. Train for 30 minutes with the default reward function, then evaluate to confirm the agent completes laps.
2. **Speed optimization**: Study reward function parameters and modify your reward function to incentivize faster driving. Extend training to 1-2 hours, comparing reward graphs to track improvement.
3. **Action space tuning**: Read about action space and increase top speed (e.g., 1 m/s) by creating a new model. Higher speeds improve lap times but increase training complexity as agents may overshoot curves. Consider reducing granularities and refining reward functions to accelerate convergence.
4. **Generalization testing**: Progress to complex tracks, repeating steps 1-3. Evaluate models on different tracks to test generalization capabilities.
5. *(Optional)* Experiment with hyperparameter variations and analyze training logs for optimization insights.

Training a model for object avoidance races

After training for time trials, move on to object avoidance—training models to complete fast laps while avoiding track obstacles. This challenge requires longer convergence times due to increased complexity.

Obstacle placement options:

- **Fixed locations**: Obstacles remain stationary throughout training. Easier convergence but risk of overfitting.
- **Random locations**: Obstacles change positions between episodes. Harder convergence but better generalization for real-world transfer.

Start with fixed locations to understand behaviors before tackling random placement. Simulator obstacles match AWS DeepRacer packaging dimensions (9.5" × 15.25" × 10.5"), simplifying real-world transfer.

Training progression:

1. **Fixed obstacle training**: Use default agent or customize sensors/action space. Limit speed to <0.8 m/s with 1-2 speed levels. Train 3 hours with 2 fixed objects on your target track (e.g., AWS DeepRacer Smile Speedway). Evaluate and monitor reward convergence.
2. **Reward optimization**: Study reward function parameters and experiment with reward function variations. Increase obstacles to 4, adjusting functions, speeds, or obstacle counts as needed until no significant improvement occurs.
3. **Random obstacle training**: Progress to stereo camera or LiDAR combinations (expect longer training). Use low top speed (e.g., 2 m/s) and shallow neural networks for faster convergence.
4. **Advanced training**: Train 4 hours with 4 randomly placed objects on simple tracks. If unsuccessful, modify reward functions, try different sensors, extend training time, or clone existing models to leverage prior learning.
5. *(Optional)* Experiment with higher speeds, more obstacles, and different sensor combinations to optimize performance.

Train a model

Training is how the model learns optimal driving behaviors by repeatedly interacting with the simulated track environment. During training, the vehicle explores different actions (steering angles and speeds) in response to various track states (camera images), receiving rewards based on a user-defined reward function that incentivizes desired behaviors like staying on the center-line or completing laps quickly. This trial-and-error allows the model to discover which actions maximize rewards, refining its neural network weights with each training iteration until it converges on a policy that consistently navigates the track successfully.

Training automatically begins right after you create the model. After creating a model, you will be redirected to the model detail view, and you will see that the model is in one of the following states as indicated by the label shown in the Training section.

- **Queued** - the model has been queued for training, and training will begin shortly.
- **Initializing** - the resources needed for performing the training job are warming up.
- **Training** - the model is actively being trained against the parameters you provided while creating the model.
- **Ready** - the model has completed training and can be either submitted for evaluation or submitted to a community race. It can also be exported.

- **Error** - the model has encountered an error during training.

During training, you can view the progress of how your vehicle attempts to complete the track through the video player in the Training section. After training is completed, you can submit your model for evaluation or to a virtual race.

Evaluate a model

Overview

Evaluation is the process of testing a trained reinforcement learning model's performance on a track to measure how well it has learned to navigate autonomously. During evaluation, the model is deployed in the simulator (or on a physical DeepRacer car) and runs through multiple laps without further learning, allowing you to assess key performance metrics such as lap completion rate, lap times, and consistency across different track conditions.

Evaluation is critical because it provides objective feedback on whether the model's learned behaviors translate into successful track navigation, helps identify which checkpoint or iteration produced the best-performing model, and guides decisions about whether to continue training, adjust the reward function, or deploy the model to competitions. Unlike training where the agent explores and learns through trial and error, evaluation focuses purely on measuring the model's current capabilities in a controlled testing environment.

Submitting a model for evaluation

To submit a trained model for evaluation, click the Evaluation tab in the model detail view, and either click **Start evaluation** or **Start new evaluation**. This will bring you to a page where you will be asked to provide:

- **Evaluation name** - pick a unique name for the evaluation (i.e. "my-evaluation-001").
- **Race type** - either time trial or object avoidance.
  - If you select object avoidance, you'll be asked to specify whether you'd like objects to be placed in fixed or random locations; how many objects you would like placed; and the locations of those objects if you opt to use fixed locations.
- **Race track** - the type of track to evaluate your model on.
- **Track direction** - the direction in which you would like your vehicle to travel.
- **Number of laps** - the number of laps you would like your vehicle to attempt.

When you're ready to proceed, click **Start evaluation**.

Monitoring progress

After submitting a model for evaluation, you will see a new entry in the **Evaluations selector** and details of the evaluation below. The evaluation status will show as "initializing" before beginning, this usually takes a few minutes. You can find the details of the evaluation you have configured below in the **Evaluation configuration** section.

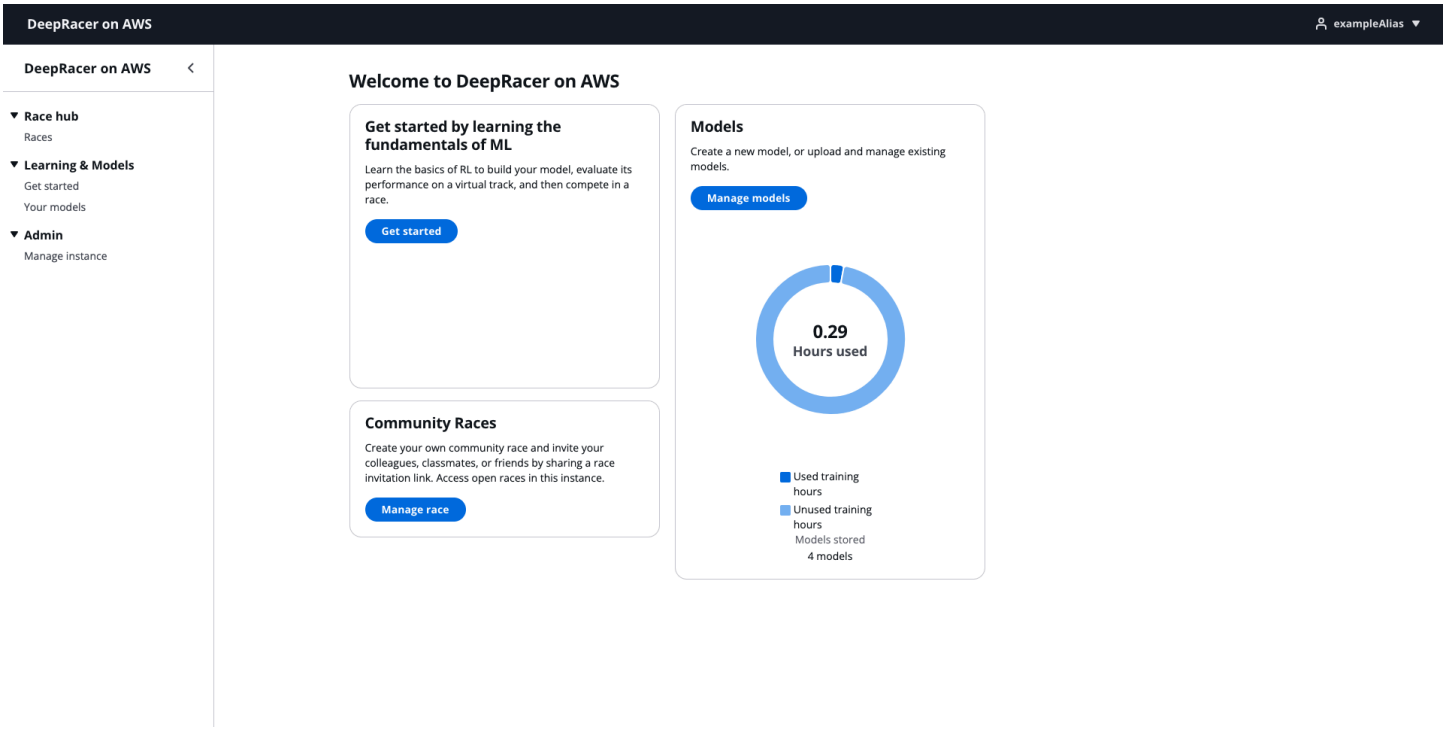
When the evaluation begins, you will be able to see your vehicle's progress around the selected track in the video player. You will also see lap results being posted in the **Evaluation results** table to the right of the video player as the car completes each lap around the track.

After the evaluation finishes, you can download the logs for that evaluation by clicking the **Download logs** button.

> **Note**
>
> Logs are exported in .tar.gz format. If you are using Windows, you will need to install 7-Zip or another program that is capable of extracting this type of file.

To view an existing evaluation, select the evaluation from the list in the **Evaluations selector**, then click **Load evaluation** button. This will load details of the selected evaluation.

Monitor usage



The usage graph on the home page shows how many **compute hours** you've used out of the number of compute hours that you've been allowed to use. Training and evaluating models on DeepRacer on AWS requires compute power, which is provided by Amazon SageMaker AI training jobs. Compute usage is measured in hours, and is accrued as-you-go.

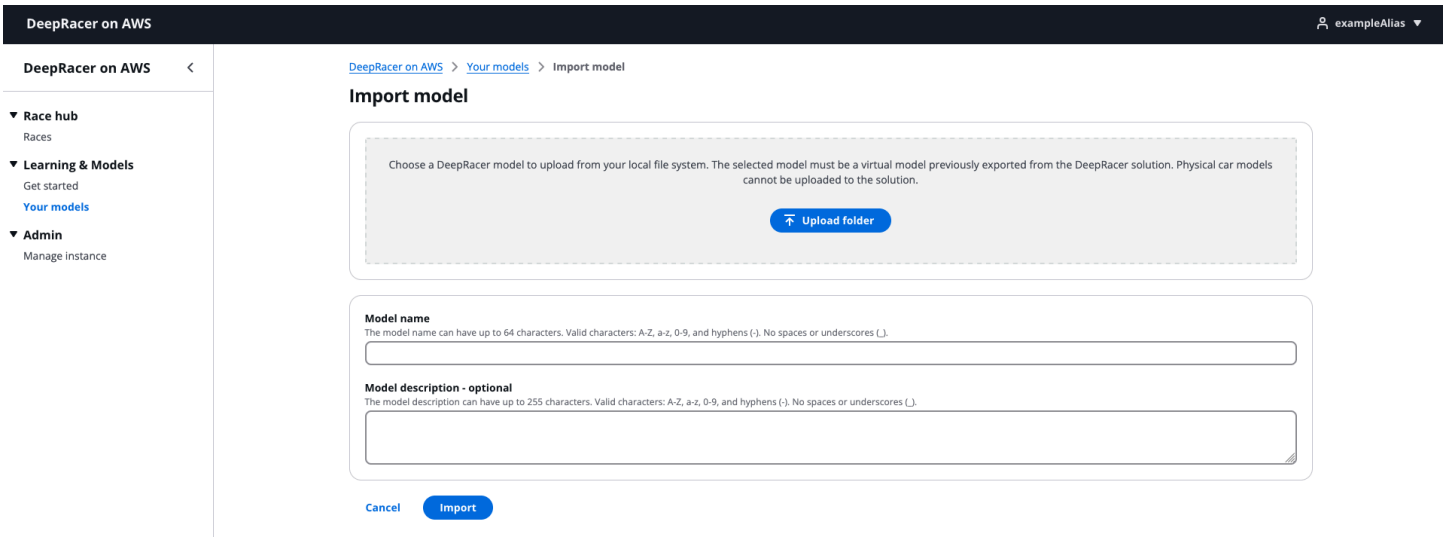
Depending on the configuration selected by your admin, you may have either been given a certain number of hours that you can use, or you may have unlimited hours. Compute hours are accrued throughout a given month, and are reset at the beginning of a new month.

If you exceed or are close to exceeding the number of compute hours that you've been allowed, you will need to reach out to your admin to see if more can be allocated.

Under the graph, you can see your current **model storage** defined by the number of models you currently have stored on the instance over the total number you can store. If your admin has not set a limit on the number of models you can store, you will see **Unlimited** for the model count.

Import and export models

Import a model



You can import a model that you've previously exported from either:

1. Your current DeepRacer on AWS deployment
2. Another DeepRacer on AWS deployment
3. The original AWS DeepRacer service

> **Note**
>
> Models that were exported from the AWS DeepRacer service a long time ago may not be compatible with DeepRacer on AWS as they were produced with an older version of the simulation application.

To do this, from the home page, click the **Your models** tab in the left sidebar, and click the **Import model** button towards the upper-right.

On the **Import model** screen, start by clicking **Upload folder**. This will open a file explorer where you can browse for an exported model on your local computer. Find the exported model folder that you would like to upload and double-click into it. You can then click **Upload** in the bottom-right of the explorer.

> **Note**
>
> If your exported model is in an archived or zipped format, you will need to extract it first before uploading.

After you have selected the folder to be uploaded, provide a name for your model and an optional description. When you are ready to proceed, click the **Import** button. You will be able to see the progress of all files being uploaded. Once all files are uploaded, you will be redirected to the model detail view where the status will show as **Importing**.

The model status will then resolve to **Ready** after the import process is completed.

Export a virtual model

> **Note**
>
> Virtual models are exported in .tar.gz format. If you are using Windows, you will need to install 7-Zip or another program that is capable of extracting this type of file.

You can export a virtual model from DeepRacer on AWS to use on another deployment of DeepRacer on AWS, or simply keep as a backup. To do this, from the home page, click the **Your models** tab in the left sidebar, and click the model that you would like to export. On the model detail view, click **Actions** > **Download virtual model**. This will begin preparing the model for download. Once it is ready, your browser will begin downloading it into its normal downloads folder.

Export a physical car model

> **Note**
>
> Physical models are exported in .tar.gz format. If you are using Windows, you will need to install 7-Zip or another program that is capable of extracting this type of file.

You can export a physical car model from DeepRacer on AWS to use on a DeepRacer-compatible physical car. To do this, from the home page, click the **Your models** tab in the left sidebar, and

click the model that you would like to export. On the model detail view, click **Actions** > **Download physical car model**. This will begin preparing the model for download. Once it is ready, your browser will begin downloading it into its normal downloads folder.

See Operate your vehicle for more information on how to upload the model to a physical car.

Manage your models

Clone a model

You can clone any model that you've created or imported, provided it is **not in an error state**, or **not currently being trained or evaluated**.

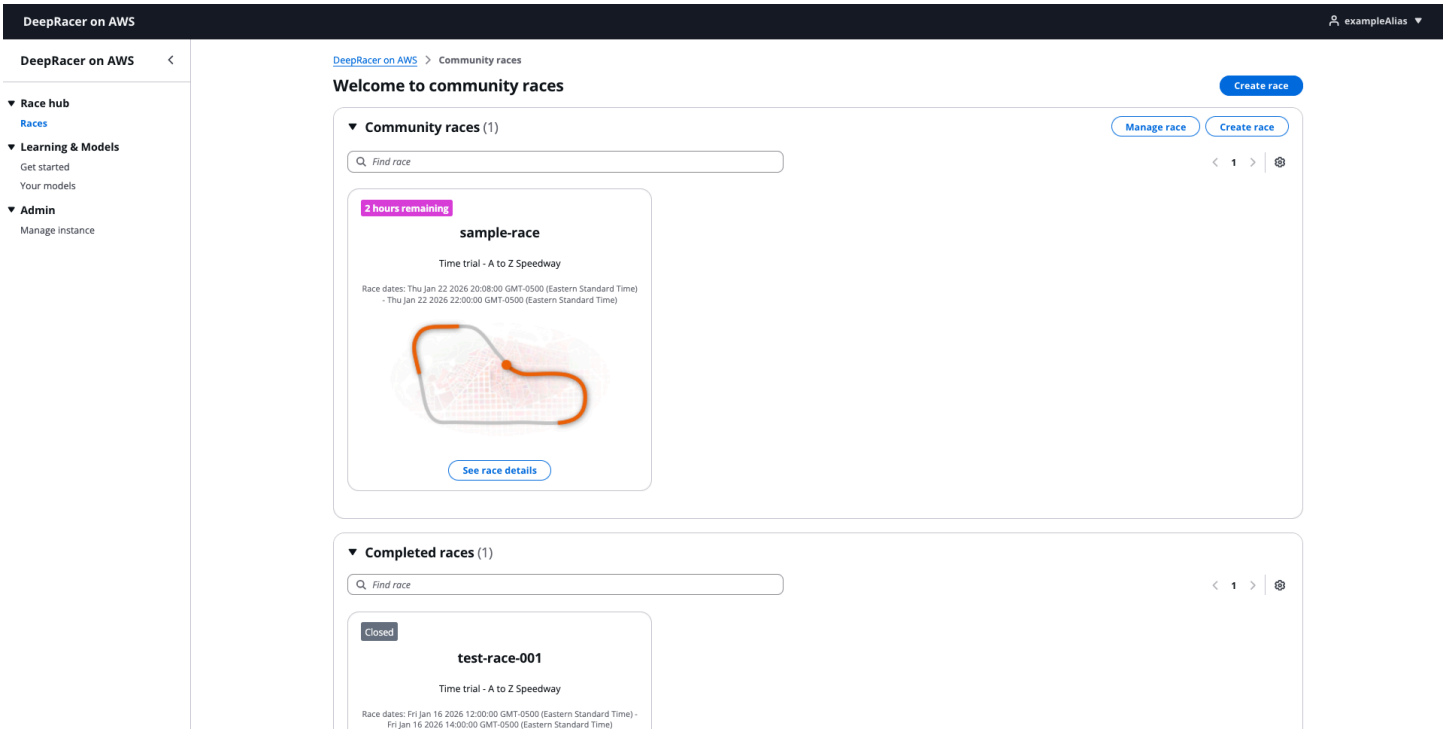
To do this, from the home page, click the **Your models** tab in the left sidebar, and select the model that you would like to clone. With the model selected, click **Actions** > **Clone**. This will open the same wizard that is used for creating a model, but with the attributes of the model you selected populated in the wizard fields.

If you would like to keep the exact same configuration as the original model, simply click through the wizard and click **Create** at the end to create the model. Otherwise, you can make adjustments to any aspect of the original model's configuration before creating it.

Delete a model

You can delete any model that you've created or imported. To do this, from the home page, click the **Your models** tab in the left sidebar, and select the model that you would like to delete. With the model selected, click **Actions** > **Delete**. You will be presented with a confirmation dialog. If you would like to proceed, click **Delete**. This action is permanent and cannot be undone.

Join a community race



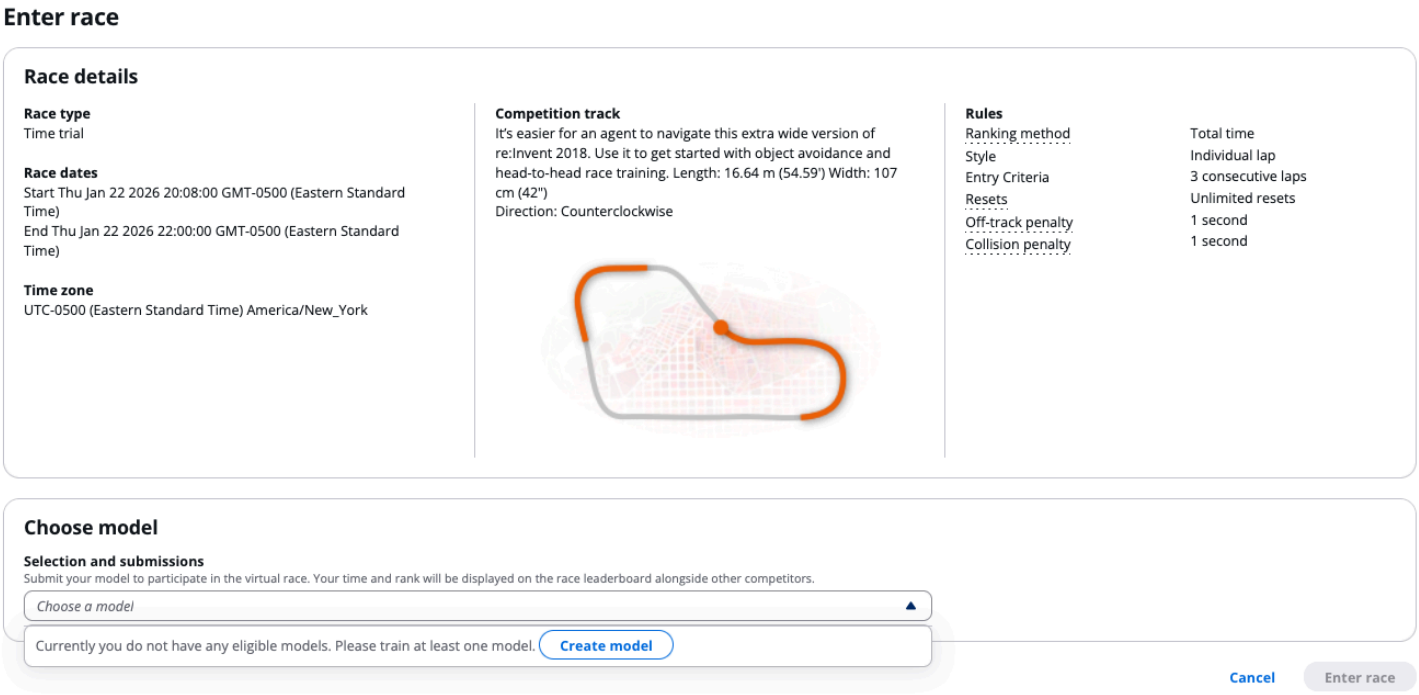
Community races allow you to test your trained DeepRacer model against other participants in competitive events organized by admins or race facilitators. When a community race is created and opened for submissions, you can select one of your trained models and submit it to the race, where it will be evaluated by running multiple laps on the designated track to record its best lap time. Your model competes against other racers' submissions, with results displayed on a leaderboard that ranks all participants based on their fastest lap times, allowing you to see how your model performs compared to others without needing to race in real-time. Community races provide an exciting way to benchmark your reinforcement learning skills, experiment with different reward functions and hyperparameters, and iterate on your models to climb the rankings throughout the competition period.

You can see whether there are any open races that are accepting submissions by clicking the Races tab in the left sidebar. Open races that are accepting submissions will be displayed in the **Community races** section.

Entering a race

You can enter a race by clicking the blue **Enter race** button at the upper-right corner of the race detail page. Doing so will bring you to a page which will reiterate the characteristics of that race in the **Race details** section, and will ask you to select which model you would like to submit to the

race under the **Choose model** section. In the model selection dropdown, you will see a list of all the models you have trained that are eligible for submission.



To enter the race, select a model from the list of options and click **Enter race**. You will be directed back to the race details page where you will be able to see your model in the **Your submissions** tab.

Monitoring progress

Races are open-format and models are evaluated as they are submitted, with the leaderboard being adjusted as each result is posted. When you first submit your model to the race, you will see its status as **Queued** in the **Your submissions** tab. Your model status will then change to **Initializing** as underlying resources are warmed up. Your model status will then change to **In progress**, indicating that your model is currently up for evaluation.

When your model finishes evaluation, its status will change to **Completed**, and you will be able to see the time it took for your model to complete the race. On the **Race leaderboard** tab, you will also be able to see your rank, time, gap to first place, and number of times your model went off track. In the panel to the left, you will see your alias, your fastest time and the name of your fastest model submitted.

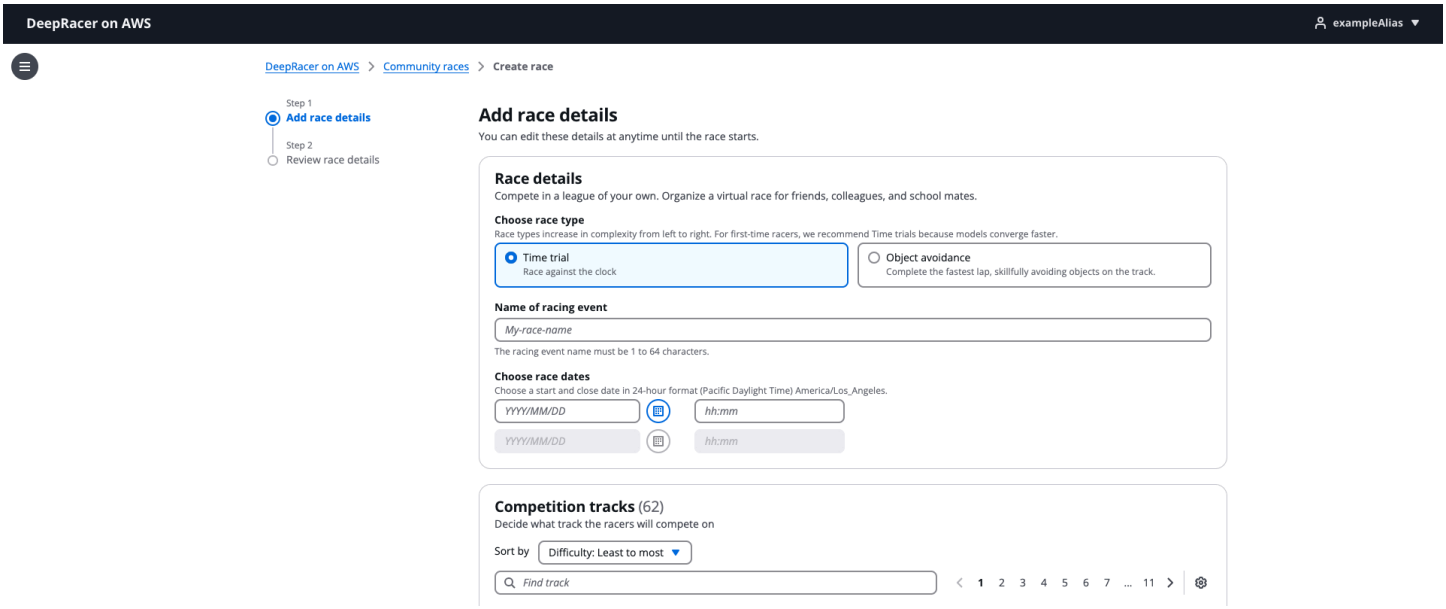
Completed races

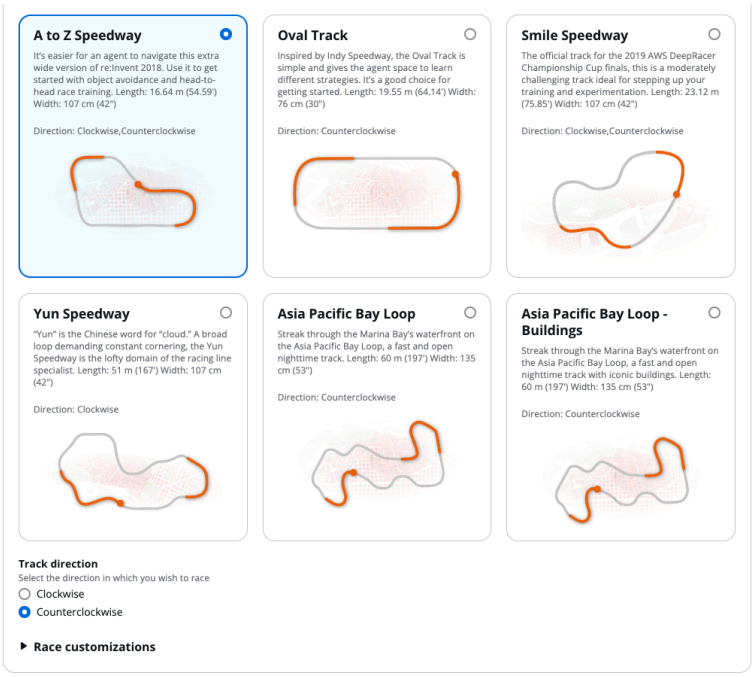
Completed races are races that have happened in the past. These are defined by races whose entry windows have expired, and are now shown as **Closed**. You can view past races that have occurred on the deployment, and their characteristics at a glance from the **Completed races** section. You can click **See race details** to view the in-depth characteristics of that race and its final results via its leaderboard. You can also view any submissions that you have made to that race and the outcome of those submissions.

Create and manage races

> **Note**
>
> Only admins and race facilitators can create and manage races.

Creating a race





Races allow participants to test their trained models against those of other participants in a competitive setting. Races are open to all users on a given deployment to participate in, and operate by offering a pre-selected window of time during which users can submit their models. When a user submits one or more trained models to a race, those models are evaluated as they arrive and, when completed, the race leaderboard is updated to include the results of those evaluations. When the time window that the race is open for expires, the race will close and the leaderboard will become final. No submissions are allowed once a race is closed.

To create a race, from the home page, click the **Races** tab in the left sidebar. Then, click **Create race** towards the upper-right corner. You will be asked to provide the following:

- **Race type** - either time trial or object avoidance. Time trial aims to go around the track as quickly as possible, while staying on the track. Object avoidance aims to complete the fastest lap while avoiding objects on the track.
- **Name** - give your race a name.
- **Race dates** - the period of time that your race will be open from; provide an open and close date/time in 24-hour format.
- **Competition track** - the track that your race will use.
- **Track direction** - the direction of travel that vehicles in your race will use.

You can also provide a series of customizations (by expanding the **Race customizations** section):

- **Description** - a description for your race.
- **Ranking method** - how submissions are ranked, either by total time, best lap time, or average lap time.
- **Minimum laps** - the number of laps required for a model to pass evaluation.
- **Off-track penalty** - the number of seconds added to a racer's time for driving off the track.
- **Max submissions per user** - the maximum number of submissions a user can submit to the race.

If you select an object avoidance race, you can also customize the following:

- **Collision penalty** - the number of seconds added to a racer's time for collision.
- **Number of objects** - the number of objects for a racer to avoid on the track.
- **Randomize obstacles** - whether obstacles should be randomly placed or in fixed positions.

If you opt not to randomize obstacles, you can define where the objects will be placed on the track.

When you are ready to proceed, click **Next**. This will direct you to a page where you can review your race settings. If you would like to make any changes, click the **Edit** button in the upper-right. If you would like to proceed with the configuration as shown, click **Submit**.

After clicking **Submit**, you will be directed to the race detail view, where you can view the race details, leaderboard, and any submissions you might make yourself.

Editing a race

You can edit the details of a race up until the time the race begins from the race detail view by clicking the **Edit race** button towards the upper-right corner of the screen. Once the race begins, this button will be greyed out and you will not be able to change the race configuration.

Deleting a race

You can delete a race up until the time the race begins from the race details view by clicking the **Delete race** button towards the upper-right corner of the screen. Once the race begins, this button will be greyed out and you will not be able to delete the race.

Cloning a race

You can clone or duplicate a race in the event that you would like to create a new race with the same configuration as a current or past race. To clone a race, from the home page, click the **Races**

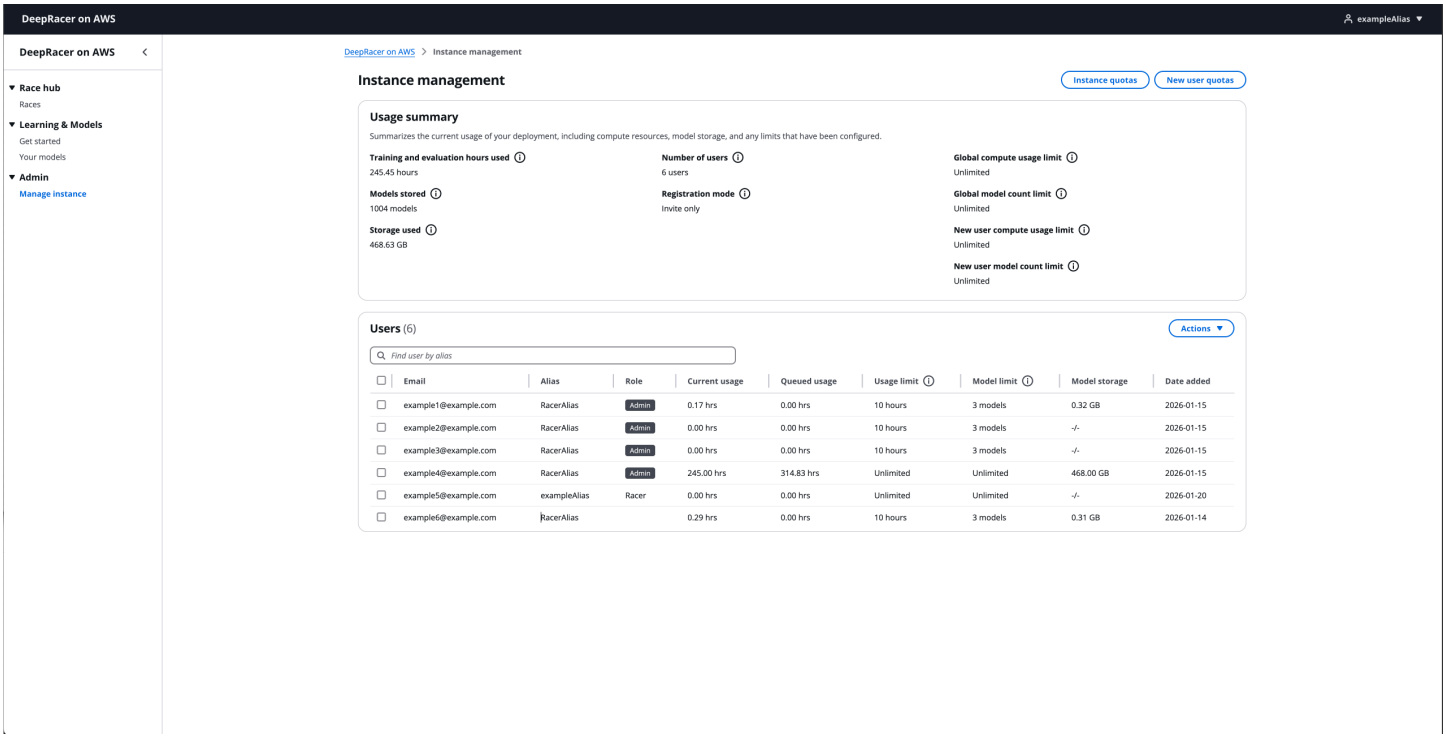
tab in the left sidebar. Then, click **Manage race** towards the upper-right corner. Select the race you would like to clone from the table, and click **Clone race** towards the upper-right corner. This will bring you to the same experience that is used for creating a race, but with all of the form fields pre-populated using the values of the race being cloned. You may choose to click through all of the steps to keep all settings the same, or make any changes as desired.

Admin functions

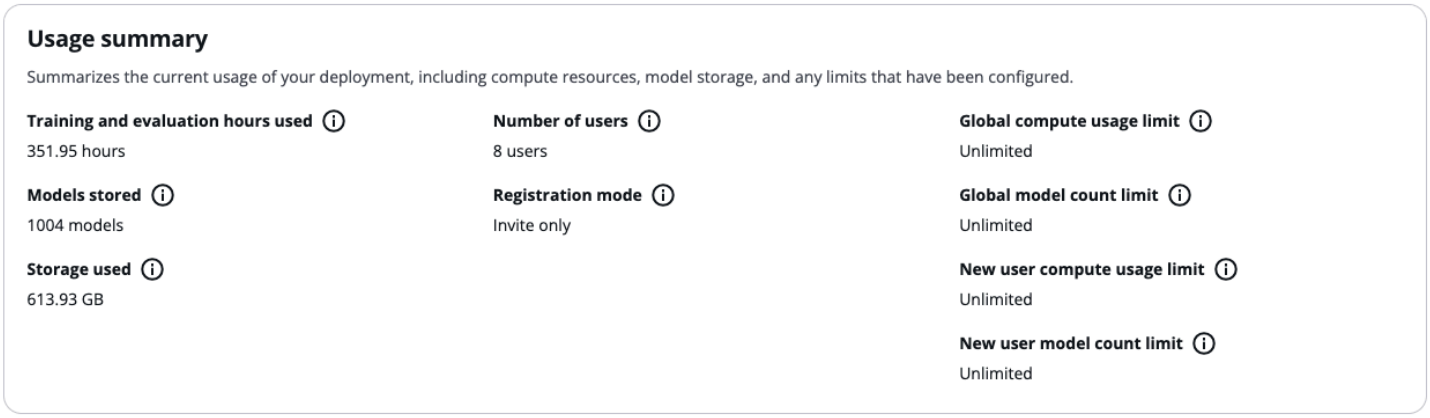
> **Note**
>
> Only admins can perform the functions in this section.

Manage instance page

The Manage instance page provides a consolidated interface for managing the different facets of a DeepRacer on AWS deployment.



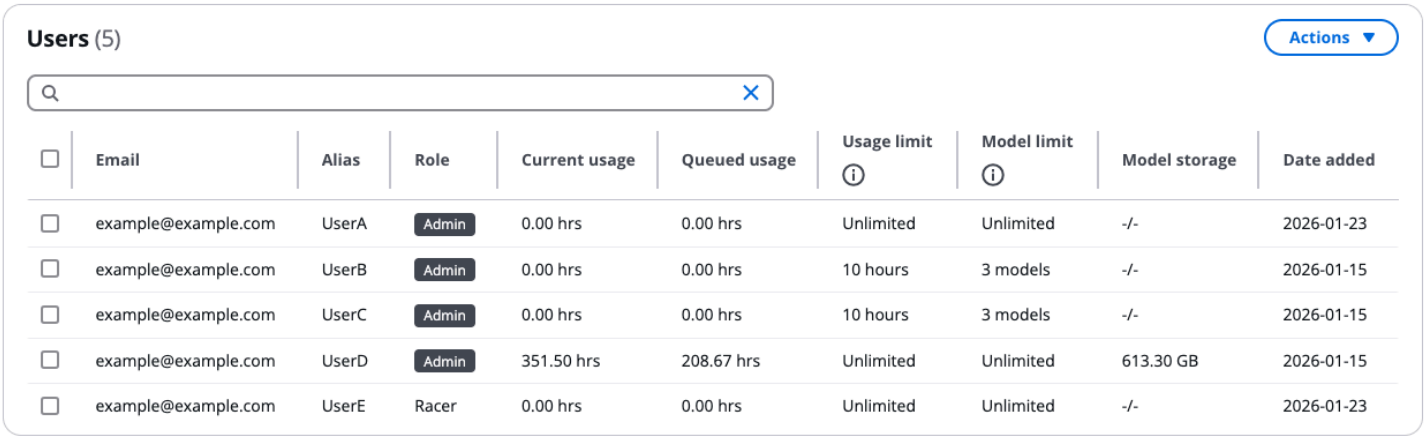
Usage summary



The usage summary on the **Manage instance** page provides key metrics for your deployment at-a-glance. From this view, you can see:

- **Training and evaluation hours used** - the number of hours that have been spent on training and evaluating machine learning users on your deployment.
- **Models stored** - the total number of models currently stored on your deployment across all users.
- **Storage used** - the total amount of storage space used by all models across all users on your deployment.
- **Number of users** - the total number of users who are registered on your deployment.
- **Registration mode** - how new users join your deployment. "Invite only" means users must be explicitly invited by an administrator.
- **Global compute usage limit** - the maximum total compute hours that can be used across all users in your deployment for training and evaluation.
- **Global model count limit** - the maximum total number of models that can be stored across all users in your deployment.
- **New user compute usage limit** - the default compute usage limit that will be assigned to new users.
- **New user model count limit** - the default model count limit that will be assigned to new users.

Users table



The users table on the **Manage instance** page shows all of the users who are registered to your deployment. This table allows you to search by email address or alias. It also shows the following attributes for each user:

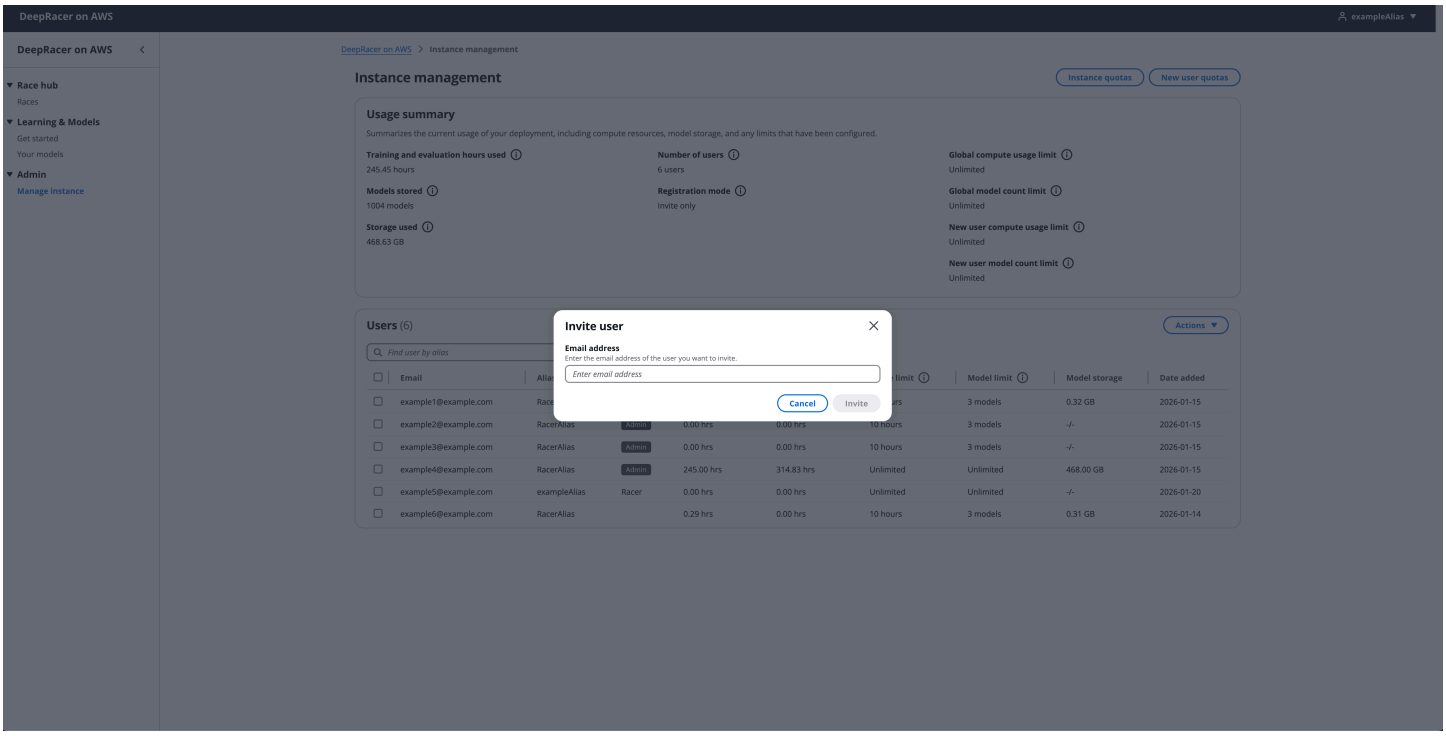
- **Email address** - the email address that the user used when creating their account.
- **Alias** - a custom alias selected by the user; used primarily for identifying models in races.
- **Role** - the user's role, either Admin, Race facilitator, or Racer.
- **Current usage** - the number of compute hours used for training and evaluation jobs.
- **Queued usage** - the number of compute hours that are expected to be used based on jobs that have been queued.
- **Usage limit** - the individual compute usage limit, if any, that has been set.
- **Model limit** - the individual model count limit, if any, that has been set.
- **Model storage** - the amount of storage currently being used by models and artifacts.
- **Date added** - the date when the user's account was created.

The **Actions** menu features various user management actions that can be performed when either 0 or 1 users in the table are selected. The sections in User management expand on these actions further.

User management

This section provides an overview of how to manage users in the DeepRacer on AWS solution.

Invite a user

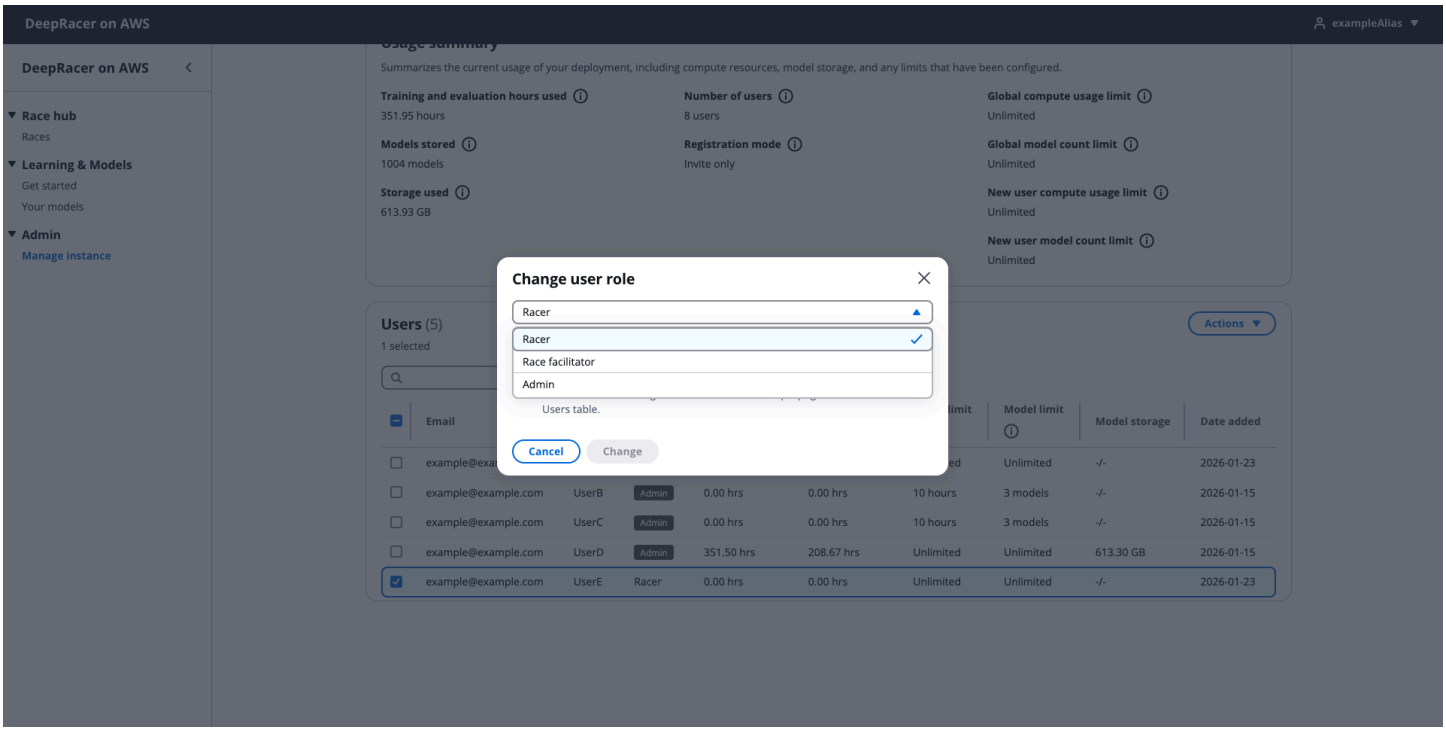


Admins can invite users to an instance via the Manage instance page by clicking **Actions > Invite user**. The **Actions** menu is in the upper-right corner of the Users section. Users are invited to the instance using their email address. Once you provide the email address of the user you'd like to invite, click **Invite**.

The invited user will receive an email with a temporary password which can be used to log in to the instance and set a preferred password. Once the preferred password is set, the temporary password will be invalidated and the preferred password can be used going forward.

Users who are invited to the instance are assigned to the **Racer** role by default.

Change user permissions

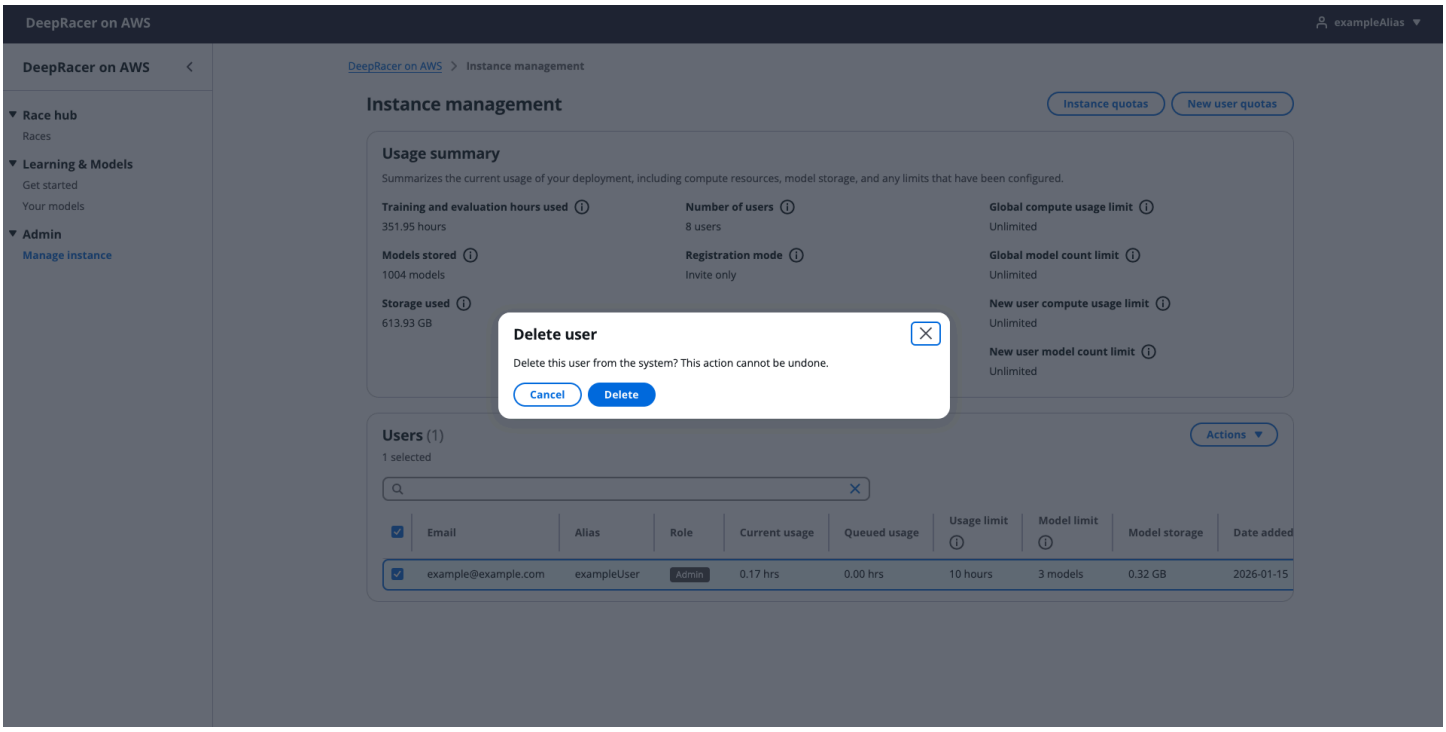


Admins can change the permission level that a given user has via the **Manage instance** page by selecting the user in the Users table, clicking **Actions > Change role**, and selecting the desired role from the list of options. Click **Change** to save your selection when ready.

If the user whose role you are updating is currently logged in, that user will need to log out and log back in for their role change to take effect. A successful role change will take 1-2 minutes to propagate and reflect in the Users table.

> **Note**
>
> For more information on each type of user and their permissions, see the Types of users section.

Delete a user



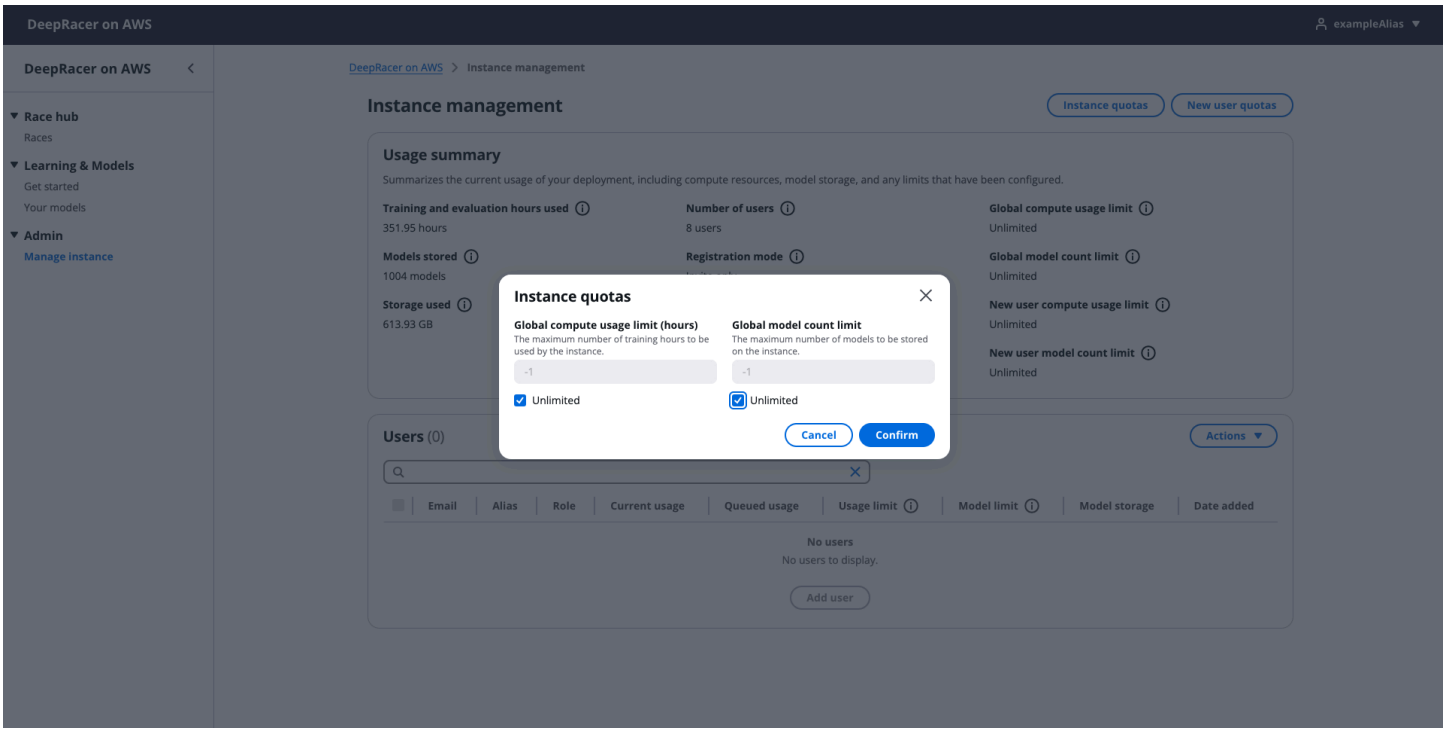
Admins can delete a user from the instance via the **Manage instance** page by selecting the user in the Users table and clicking **Actions > Delete user**. Click **Delete** in the pop-up modal to confirm your choice.

> **Note**
>
> Deleting a user from the system permanently deletes their profile and all of their data. The action cannot be undone.

Cost and usage management

DeepRacer on AWS provides a robust set of controls allowing admins to manage cost by setting limits on compute and storage resource utilization. Limits are effective from the time they are set, and will only allow or restrict actions that are performed after they are set.

Global limits



Global limits allow compute and storage usage limits to be set at the instance level. These limits prevent the instance as a whole from exceeding a certain number of compute hours or storage capacity in a given month. This can help admins stay within a certain operating budget, mitigate against unexpected volume, and/or control variable costs.

Global compute usage limit controls the number of compute hours that can be consumed by all users on a given instance in a given month through training and evaluating models. By default, this is set to **Unlimited**.

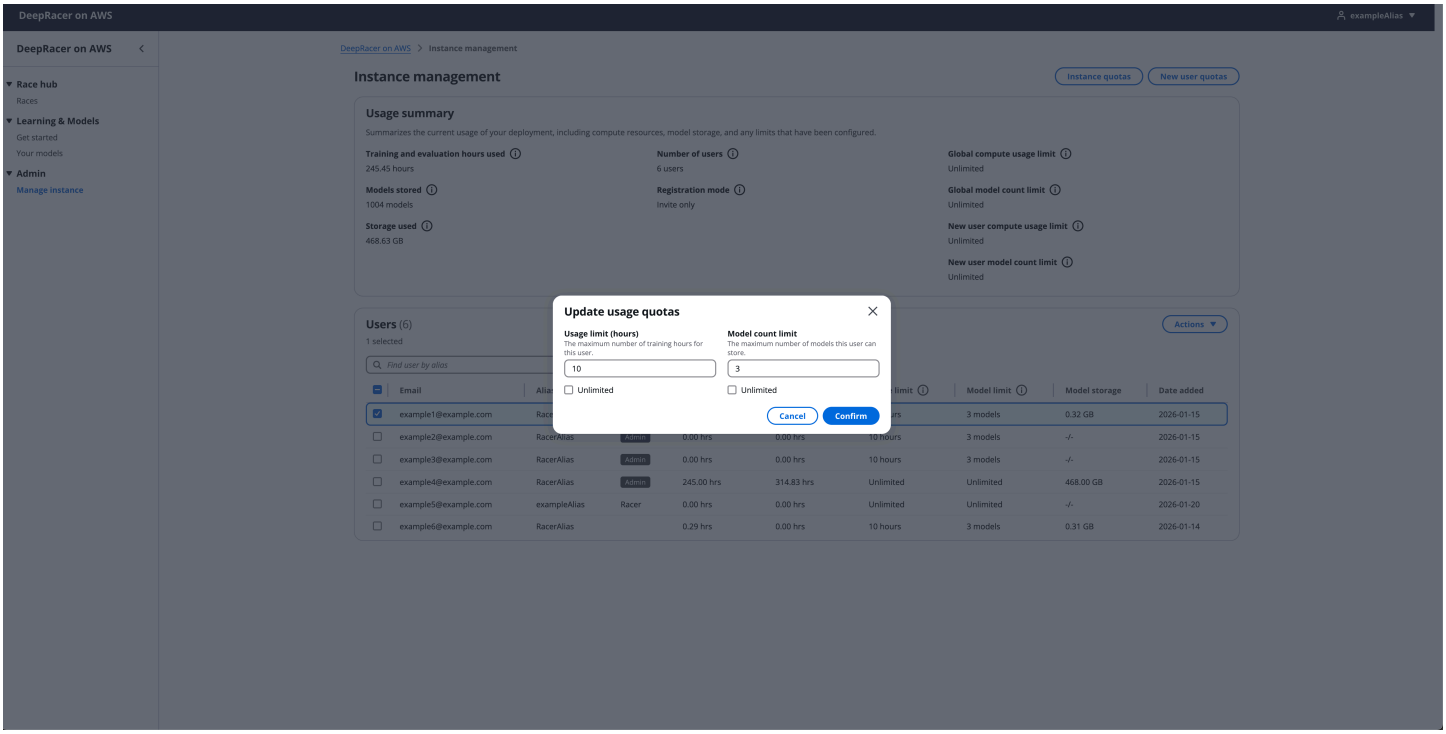
- This can be viewed on the **Manage instance** page in the **Usage summary** section under **Global compute usage limit**.
- This can be changed by clicking **Instance quotas** and either updating the value for **Global compute usage limit (hours),** or checking the box for **Unlimited**. Clicking **Confirm** will save your selection.

Global model count limit controls the number of models that can be created, imported, and otherwise stored on a given instance. By default, this is set to **Unlimited**.

- This can be viewed on the **Manage instance** page in the **Usage summary** section under **Global model count limit**.

- This can be changed by clicking **Instance quotas** and either updating the value for **Global model count limit,** or checking the box for **Unlimited**. Clicking **Confirm** will save your selection.

Individual limits



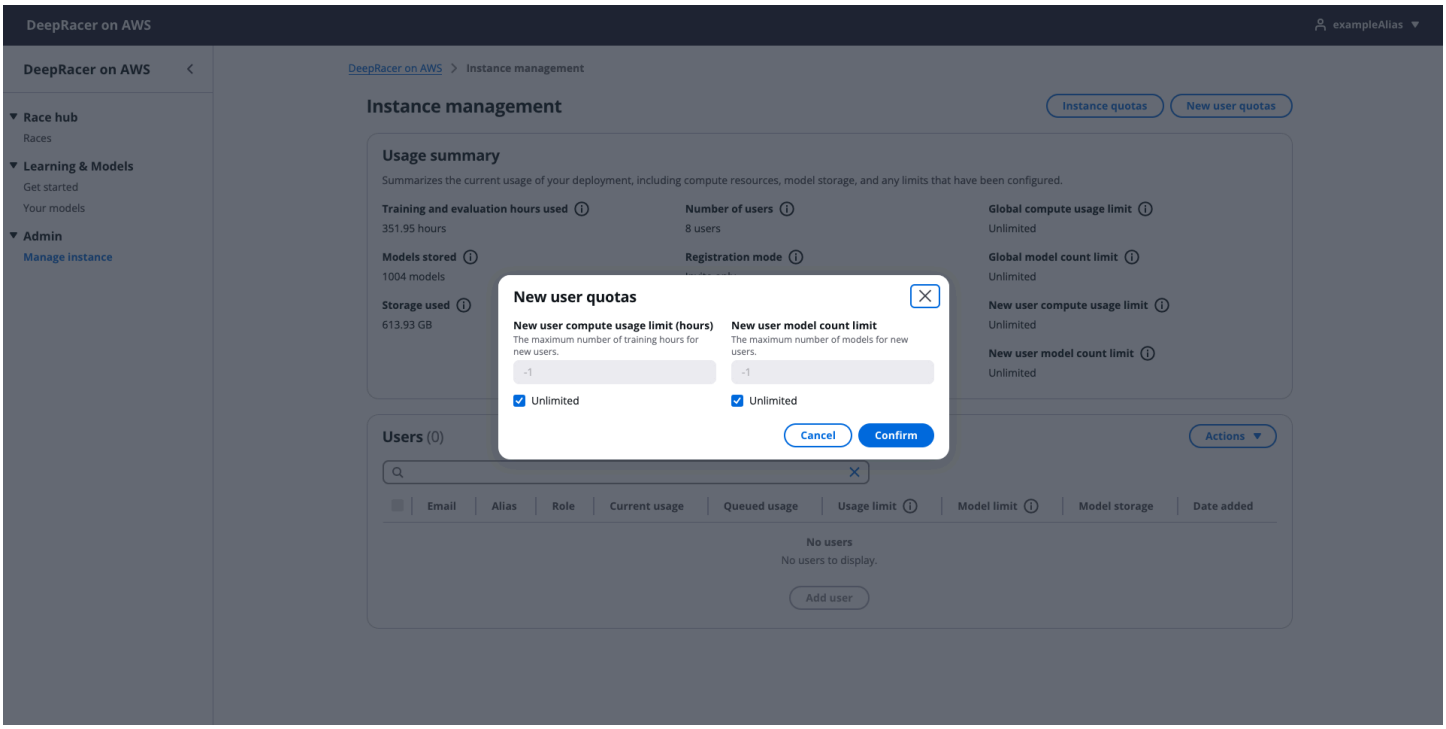
Individual limits allow compute and storage usage limits to be set at the individual user level. These limits prevent a specific user from exceeding a certain number of compute hours or storage capacity in a given month.

The **individual usage limit** for a given user controls the number of compute hours that can be consumed by that users in a given month through training and evaluating models. By default, this is set to **Unlimited**.

- This can be viewed on the **Manage instance** page in the **Users** table under the **Usage limit** column for a given user.
- This can be changed by selecting a given user's row in the table, clicking **Actions > Update usage quotas**, and setting the value for **Usage limit (hours)**. Clicking **Confirm** will save your selection.

The **individual model limit** for a given user controls the number of models that can be created, imported, and otherwise stored by that user on the instance. By default, this is set to **Unlimited**.

New user limits



New user limits are default limits that are applied to any new users that are created after the limit is set.

The **new user compute usage limit** automatically sets a default limit on the number of compute hours for all accounts that are created after the limit is established or updated. By default, this is set to **Unlimited**.

- This can be viewed on the **Manage instance** page in the **Users** table under the **New user compute usage limit** in the **Usage summary** section.
- This can be changed by clicking **New user quotas** and either updating the value for **New user compute usage limit (hours),** or checking the box for **Unlimited**. Clicking **Confirm** will save your selection.

The **new user model count limit** automatically sets a default limit on the number of models that an accounts created after the limit is established or updated can have. By default, this is set to **Unlimited**.

- This can be viewed on the **Manage instance** page in the **Users** table under the **New user model count limit** in the **Usage summary** section.

- This can be changed by clicking **New user quotas** and either updating the value for **New user model count limit (hours),** or checking the box for **Unlimited**. Clicking **Confirm** will save your selection.

Levels of precedence

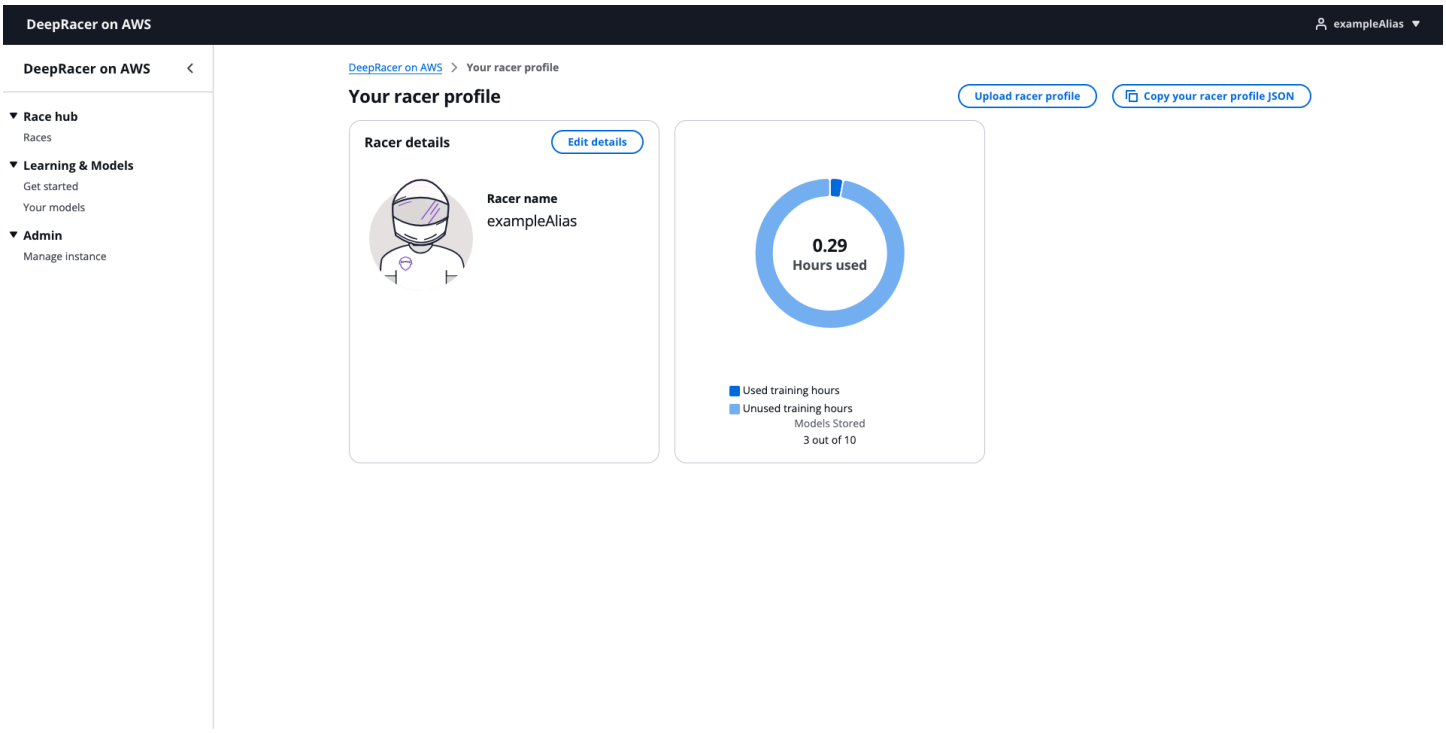
Global limits have the highest level of precedence in determining whether usage will be restricted. If the system is close to reaching a global limit, it will restrict jobs from being run even if a user has not neared or exceeded their individual limit. This is designed to help protect against cost overruns and unexpected resource utilization for the owner or operator of the solution.

Below are some examples of how levels of precedence work when it comes to limits in DeepRacer on AWS:

- If global usage is *unlimited* and individual usage is *unlimited*, then usage will not be restricted.
- If global usage is *limited* and individual usage is *unlimited*, then the global usage limit will control.
- If global usage is *limited* and individual usage is *limited*, then the limit with the least capacity will control.
- If global usage is *unlimited* and individual usage is *limited*, then the individual usage limit will control.

Profile and account

Profile

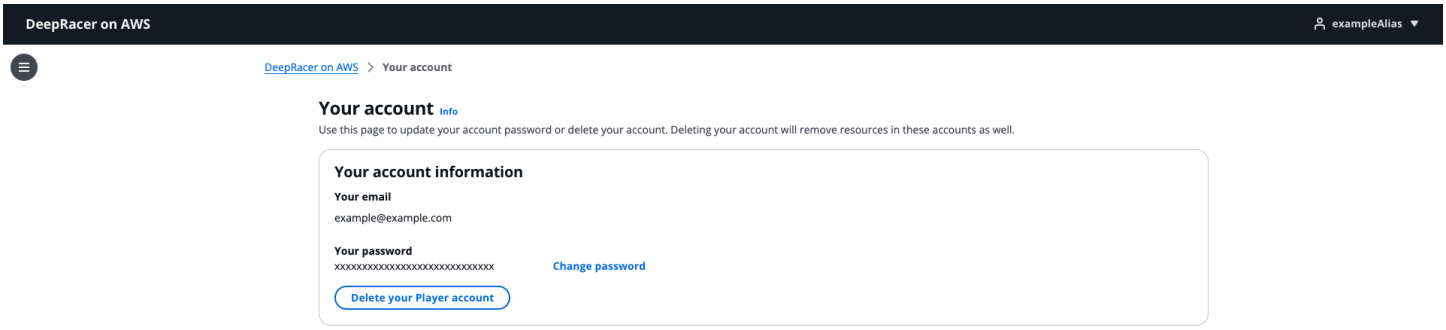


You can make changes to your profile, including your racer alias or avatar. To do this, click the dropdown menu in the upper-right corner of the screen, and click **Profile**. From here, you can edit your profile by clicking **Edit details** in the **Racer details** section. You can also view your resource usage on this page, similar to that on the home page.

You can also import or export a racer profile by using:

- **Upload racer profile** button - allows you to paste a racer configuration JSON object.
- **Copy your racer profile JSON** button - copies your racer configuration to the clipboard.

Account settings



You can view your account information and make account-level changes such as changing your password or deleting your account from the account settings page. To access this, click the dropdown menu in the upper-right corner of the screen, and click **Account**.

From here, you can:

- View the email address that your account is associated with.
- Change your password - you'll be asked to provide a new password and confirm it before continuing.
- Delete your account - you'll be asked to confirm deletion before continuing.

Use the vehicle

After you finish training and evaluating an AWS DeepRacer model in the AWS DeepRacer simulator, you can deploy the model to your AWS DeepRacer vehicle. You can set the vehicle to drive on a track and evaluate the model's performance in a physical environment. This mimics a real-world autonomous race.

Before driving your vehicle for the first time, you must set up the vehicle, install software updates, and calibrate its drive-chain sub-system.

To drive your vehicle on a physical track, you must have a track. For more information, see Build your physical track.

Topics

Get to know your vehicle

Your AWS DeepRacer vehicle is a machine learning-enabled, battery-powered, and Wi-Fi-connected 1/18th-scale model four-wheel drive car with a front-mounted 4-megapixel camera and an Ubuntu-based compute module.

The vehicle can drive autonomously by running inference that is based on a reinforcement learning model in its compute module. You can also drive the vehicle manually, without deploying any reinforcement learning model. If you have not already obtained an AWS DeepRacer vehicle, you can order one here.

The AWS DeepRacer vehicle is powered by a brushed motor. The driving speed is controlled by a voltage regulator that controls how fast the motor spins. The servo mechanism (servo) that operates the steering system is protected by the black cover in the AWS DeepRacer vehicle chassis.

Topics

Inspect your AWS DeepRacer vehicle

When you open your AWS DeepRacer vehicle box, you should find the following components and accessories:

Components	Comments
Vehicle Chassis [1]	Includes a front-mounted camera for capturing vehicle driving experiences and the compute module for autonomous driving. You can view images captured by the camera as a streaming video on the vehicle's device console. The chassis includes a brushed electric motor, an electronic speed controller (ESC), and a servo mechanism (servo)
Vehicle body shell [2]	Remove this when setting up the vehicle.
Micro-USB to USB-A cable [3]	Use this to support USB-OTG functionality.
Compute battery [4]	Use this to power the compute module that runs inference on a downloaded AWS DeepRacer reinforce ment learning model.
Compute battery connector cable [5]	Use this USB-C to USB-C cable to connect the compute module with the battery. If you have a Dell compute battery, this cable will be longer.
Power cable [6a]	Use this to connect the power adapter to a power outlet.
Power adapter [6b]	Use this to charge the compute battery and the compute module.
Pins (spare parts) [7]	Use to fasten the compute module to the vehicle chassis. These are extras.
Vehicle battery [8]	A 7.4v LiPo battery pack to power the motor.
Vehicle battery charge adapter [9a]	Use this to charge the vehicle battery that powers the vehicle drive chain.
Vehicle battery charge cable [9b]	Use this to connect the vehicle battery charger to a power outlet.
Battery unlock cable [10]	Use this if your battery enters lockout state.

To set up your AWS DeepRacer vehicle, you must also have the following items ready:

- A computer with a USB port and access to the internet.
- A Wi-Fi network connected to the internet.
- An AWS account.

Now follow the instructions in the next section to make sure your vehicle battery and the power bank are charged.

Charge and install your AWS DeepRacer batteries

Your AWS DeepRacer vehicle has two power sources: the vehicle battery and the compute module power bank.

The power bank keeps the compute module running. The compute module maintains the Wi-Fi connection, runs inference against a deployed AWS DeepRacer model, and issues a command for the vehicle to take an action.

The vehicle battery powers the motor to move the vehicle. It has two sets of cables. The two-wired set of the red and black cables is used to connect to the vehicle's ESC and the triple-wired blue (or black), white and red cables is to connect to the charger. For driving, only the two-wired cable set should be connected to the vehicle.

After fully charged, the battery voltage will drop as the batteries discharge. When the voltage drops, the available torque also drops. As a consequence, the same speed setting will result in slower speed on the track. When the battery is fully empty, the vehicle stops moving. For autonomous driving under normal conditions, the battery usually lasts 15-25 minutes. To ensure consistent behavior, it is recommended that you charge the battery after every 15 minutes of use.

To install and charge the vehicle battery and the power bank, follow the steps below.

1. Remove your AWS DeepRacer vehicle shell.
2. Remove the four vehicle chassis pins. Carefully lift vehicle chassis while keeping wires connected.
3. To charge and install the vehicle battery, do the following:
   a. To charge the battery, plug the three-wired cable set from the battery to the charger to connect the battery to the power adapter and then plug the power adapter to a wall outlet or to a USB port if a USB cable is used to charge the battery.

b. After the battery is charged, plug the two-wired cable set of the vehicle battery cable into the black and red cable connector on your vehicle.

c. To secure the vehicle battery, tie the battery under the vehicle chassis with the attached straps.

   Make sure to keep all the cables inside the vehicle.

d. To check if the vehicle battery is charged, do the following:

   i. Slide the vehicle power switch to turn on the vehicle.

   ii. Listen for two short beeps.

      If you don't hear the beeps, the vehicle is not charged. Remove the battery from the vehicle and repeat Step 1a above to recharge the battery.

   iii. When not using the vehicle, slide the vehicle power switch back to turn off the vehicle battery.

4. To check the power bank charging level, do the following:

   a. Press the power button on the power bank.

   b. Check the four LED lights next to the power button to determine the charging level.

      If all the four LED lights are lit, the power bank is fully changed. If none of the LED lights are lit, the power bank needs to be charged.

   c. To charge the power bank, insert the USB C plug from the power adapter into the USB C port of the power bank. It takes some time for the power bank to be fully charged. When it is charged, repeat Step 4 to confirm that the power bank is fully charged.

5. To install the power bank, do the following:

   a. Insert the power bank into its holder with the power button and USB C port facing the back of the vehicle.

   b. Use the strap to tie the power bank to the vehicle chassis securely.

> **Note**
>
> Do not connect the power bank to the compute module in this step.

Test your AWS DeepRacer compute module

Test the compute module to verify that it can be started successfully. To test the module by using an external power source, follow the steps below:

To test your vehicle's compute module

1. Connect the compute module to a power source. Connect the power cord to the power adapter, plug the power cord to a power outlet, and insert the power adapter's USB C plug into the USB C port on the compute module.
2. Turn on the vehicle's compute module by pressing the power button on the compute module.
3. To verify the compute module's status, check that the LED lights are shown as follows:
   - **Solid blue**

     The compute module is started, connected to the specified Wi-Fi, and ready to go.

     In this state, you can login to the compute module after you attach it to a monitor using an HDMI cable, a USB mouse and a USB keyboard. For the first-time login, use `deepracer` for both the username and password. You will then be asked to reset the password for future logins. For security reasons, choose a strong password phrase for the new password.
   - **Blinking red**

     The compute module is in setup mode.
   - **Solid yellow**

     The compute module is initializing.
   - **Solid red**

     The compute module failed to connect to the Wi-Fi network.
4. When you're done with the test, press the power button on the compute module to turn it off and then unplug it from the external power source.

Turn off your AWS DeepRacer vehicle

To turn off your AWS DeepRacer vehicle, unplug the vehicle from the external power source. You can also press the power button on the device until power indicator is off.

LED indicators

Your AWS DeepRacer vehicle has two sets of LED indicators for the vehicle status and for customizable visual identification of your vehicle, respectively.

The details are discussed as follows.

AWS DeepRacer vehicle system LED indicators

The AWS DeepRacer vehicle system LED indicators are located on the left side of the vehicle chassis when the vehicle is in the forward position in front of you.

The three system LEDs are positioned after the RESET button. The first LED (on the left side of your field of view) shows the status of the system power. The second (middle) LED is reserved for future use. The last (right) LED shows the status of the Wi-Fi connection.

LED type	Color	Status
Power	Off	There is no power supply.
	Blinking yellow	BIOS and OS are being loaded.
	Steady yellow	OS is loaded.
	Steady blue	An application is running.
	Blinking blue	A software update is in progress.
	Steady red	An error is encountered while the system is being booted or an application is being started.
Wi-Fi	Off	There is no Wi-Fi connection.
	Blinking blue	The vehicle is connecting to the Wi-Fi network.
	Steady red for 2 seconds and then off	The Wi-Fi connection failed.
	Steady blue	The Wi-Fi connection is established.

AWS DeepRacer vehicle identification LEDs

The AWS DeepRacer vehicle custom LEDs are located at the tail of the vehicle. They're used to help identifying your vehicle in races when multiple vehicles are present. You can use the AWS DeepRacer device console to set them as a supported color of your choosing.

Set up your vehicle

The first time you open your AWS DeepRacer vehicle, you must set it up to connect to a Wi-Fi network. Complete this setup to get the vehicle's software updated and to get the IP address to access the vehicle's device console.

This section walks you through the steps to perform the following tasks:

- Connect your laptop or desktop computer to your vehicle.
- Setup the vehicle's Wi-Fi connection.
- Update the vehicle's software.
- Get the vehicle's IP address.
- Test drive the vehicle.

Use a laptop or desktop computer to perform the setup tasks. We'll refer to this setup computer as your computer, to avoid possible confusion with the vehicle's compute module, which is running the Ubuntu operating system.

After the initial setup of the Wi-Fi connection, you can follow the same instructions to choose a different Wi-Fi network.

> **Note**
>
> AWS DeepRacer does not support Wi-Fi network that requires active captcha verification for user sign-in.

Topics

Get ready to setup Wi-Fi connection for your AWS DeepRacer vehicle

To setup your vehicle's Wi-Fi connection, connect your a laptop or desktop computer to your vehicle's compute module using the included USB-to-USB C cable.

To connect your computer to your vehicle's compute module, follow the steps below.

1. Make sure your computer is disconnected from Wi-Fi before connecting your device.
2. Insert the USB end of the USB-to-USB C cable into your computer's USB port.
3. Insert the cable's USB C end into your vehicle's USB C port.

You're now ready to proceed to setting up your vehicle's Wi-Fi connection.

Setup Wi-Fi connection and update your AWS DeepRacer vehicle's software

Before you follow the steps here to setup the Wi-Fi connection, be sure you complete the steps in Get ready to setup Wi-Fi connection for your AWS DeepRacer vehicle.

1. Look at the bottom of your vehicle and make note of the password printed under Hostname. You'll need it to login to the device control console to perform the setup.
2. On your computer, go to link:https://deepracer.aws to launch the device control console of your vehicle.
3. When prompted with a message that the connection is not private or secure, do one of the following.
   a. In Chrome, choose **Advanced** and then choose **Proceed to <device_console_ip_address> (unsafe)**.
   b. In Safari, choose **Details**, follow the **visit this website** link, and the choose **Visit Websites**. If prompted for your password to update the certificate trust settings, type the password and then choose **Update settings**.
   c. In Opera, choose **Continue Anyway** when warned of an invalid certificate.
   d. In Edge, choose **Details** and then choose **Go on to the webpage (Not recommended)**.
   e. In Firefox, choose **Advanced**, choose **Add Exception**, and then choose **Confirm Security Exception**.
4. Under **Unlock your AWS DeepRacer vehicle**, enter the password noted in Step 1 and then choose **Access vehicle**.
5. On the **Connect your vehicle to your Wi-Fi network** pane, choose your Wi-Fi network name from the **Wi-Fi network name (SSID)** drop-down menu, type the password of your Wi-Fi network under **Wi-Fi password**, and choose **Connect**.
6. Wait until the Wi-Fi connection status changes from **Connecting to Wi-Fi network...** to **Connected**. Then, choose **Next**.
7. On the **Software update** pane, if a software update is required, turn on the vehicle's compute module, with the included power cord and power adapter, and then choose **Install software update**.

   Powering the vehicle with an external power source helps avoid interruption of the software update if the compute module's power bank become discharged.

8. Wait until the software update status changes from **Installing software update** to **Software update installed successfully**.
9. Note the IP address shown under **Wi-Fi network details**. You'll need it to open the vehicle's device control console after the initial setup and any subsequent modification of the Wi-Fi network settings.

Launch device console

After you set up the vehicle's Wi-Fi connection and install required software updates, you should open the device console to verify if the vehicle's network connection is working. Subsequently, you can launch the device console to inspect, calibrate and manage the vehicle's other settings. The process involves signing into your vehicle's device console using the IP address of your vehicle.

The device control console is hosted on the vehicle and is accessed with the IP address you obtained at the end of the Wi-Fi setup section.

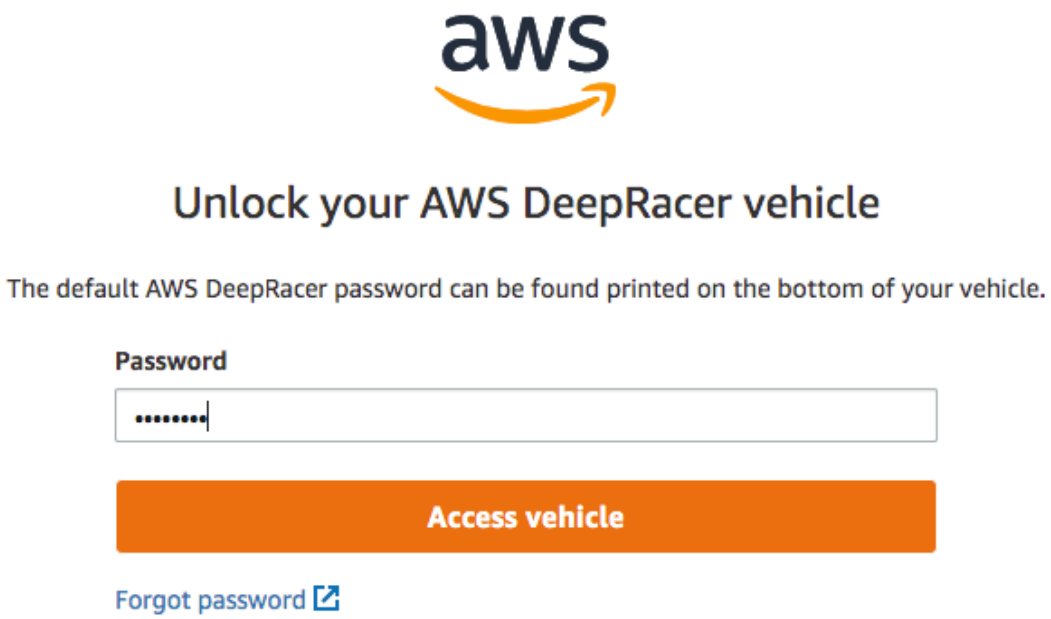
To access the device console of your AWS DeepRacer vehicle through the Wi-Fi connection

1. To access the device console of your vehicle, open a web browser on your computer, tablet or a smart phone and type your vehicle's IP address into the address bar.

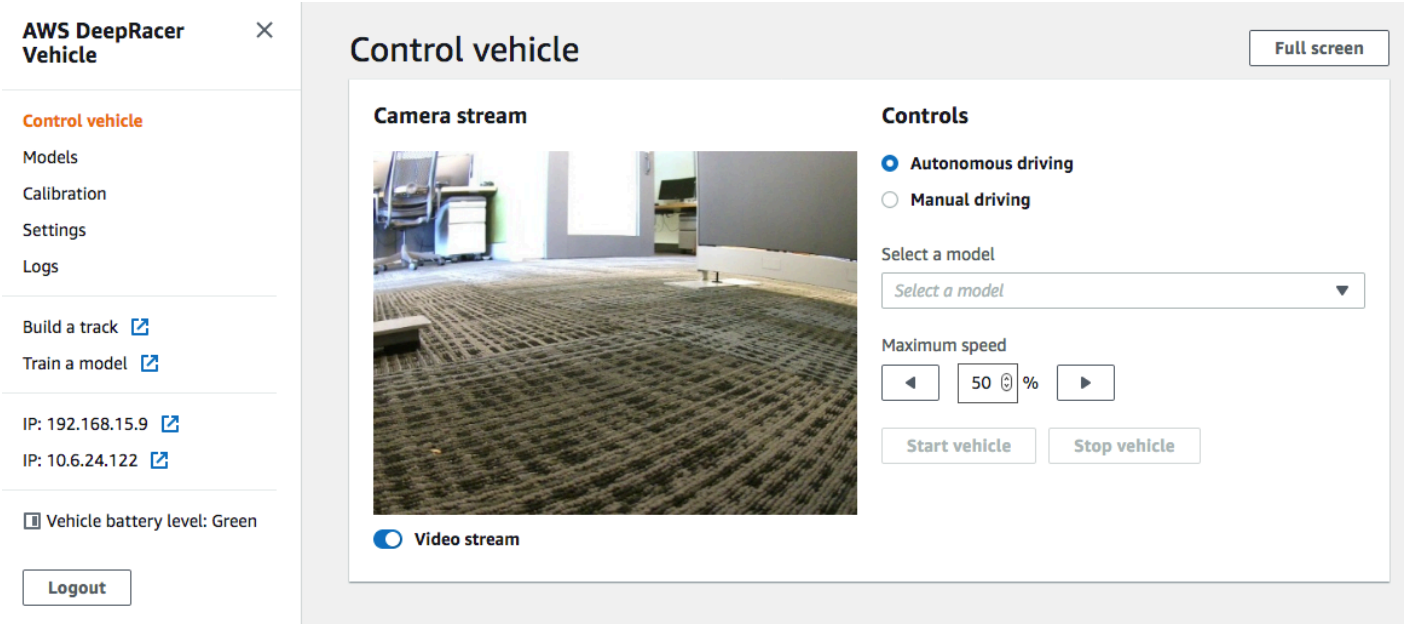
   You can get this IP address when setting up the vehicle's Wi-Fi connection. For illustration, we use 10.92.206.61 as an example.

   If you are prompted with a warning that the connection is not private or secure, ignore the message and continue to connect to the device console.
2. Under **Unlock your AWS DeepRacer vehicle**, type the device console's password in **Password** and then choose **Access vehicle**.

   You can find the default password printed on the bottom of your vehicle (under HostName).



3. When you are successfully signed in, you see the device console's home page as follows.



You're now ready to calibrate and operate your vehicle. If this is your first time operating the vehicle, proceed to calibrating the vehicle now.

Calibrate your vehicle

To achieve the best performance, it's essential that you calibrate some physical parts of your AWS DeepRacer vehicle. If you use an uncalibrated vehicle, it can add uncertainty when testing your model. If the vehicle's performance is not optimal, you might be tempted to only adjust the deep learning model code. However, you won't be able to improve the vehicle performance if the root cause is mechanical. Adjust the mechanics by calibration.

To calibrate your AWS DeepRacer vehicle, set the duty cycle range for the vehicle's electronic control system (ECS) and its servo mechanism (servo), respectively. Both the servo and ECS accept pulse-width modulation (PWM)) signals as control input from the vehicle's compute module. The compute module adjusts both of the vehicle's speed and steering angle by changing the duty cycles of the PWM signals.

The maximum speed and steering angle defines the span of the action space. You can specify the maximum speed and maximum steering angle during training in simulation. When deploying the trained model to your AWS DeepRacer vehicle for driving on a real-world track, the maximum speed and steering angle of the vehicle must be calibrated to match those used in the simulation training.

To ensure that the real-world experiences match the simulated experiences, you should calibrate your vehicle to match the maximum speed and maximum steering angles between the simulation and the real world. In general, there are two ways to do this calibration:

- Define the action space in training and calibrate the physical vehicle to match the settings.
- Measure the actual performance of your vehicle and change the settings of the action space in the simulation.

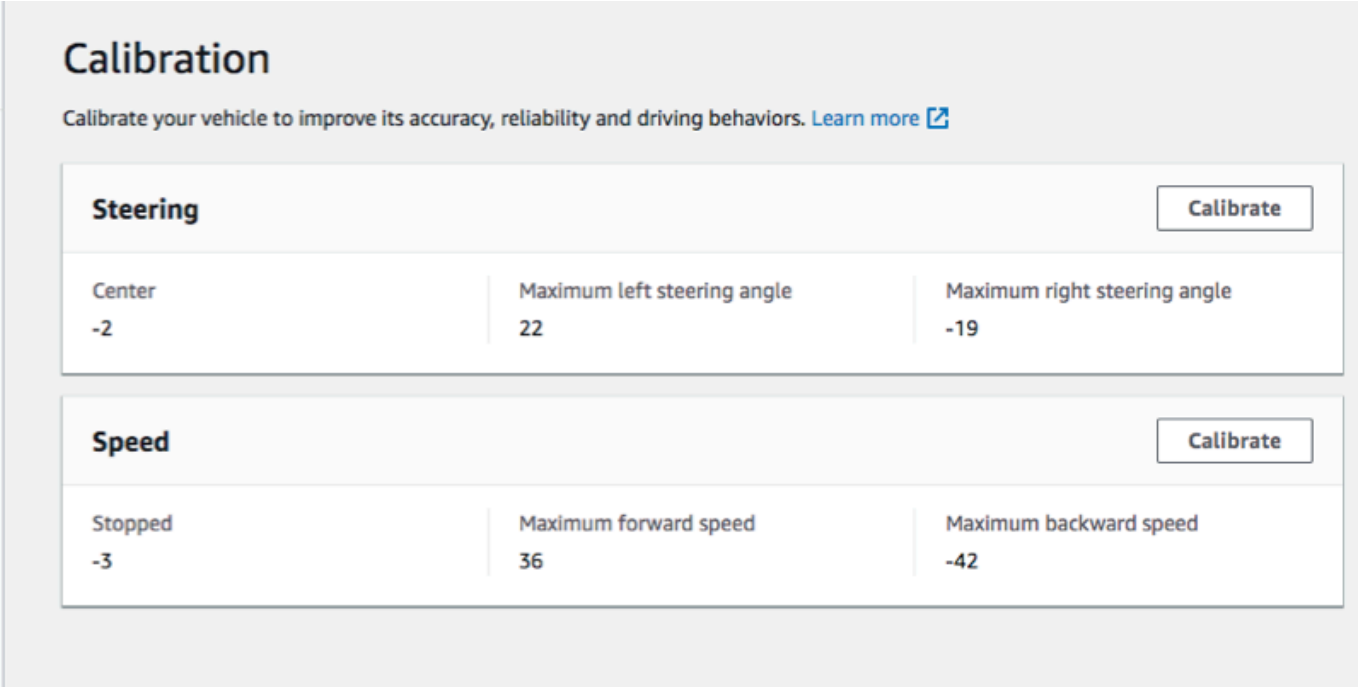
A robust model can handle certain differences between the simulation and the real world. However, you should experiment with either approach and iterate to find the best results.

Before starting the calibration, turn on the compute module. After it's started and the power LED has turned solid blue, turn on the vehicle battery. After you hear two short beeps and one long beep, you're ready to proceed with the calibration.

To calibrate your AWS DeepRacer vehicle to match the training settings:

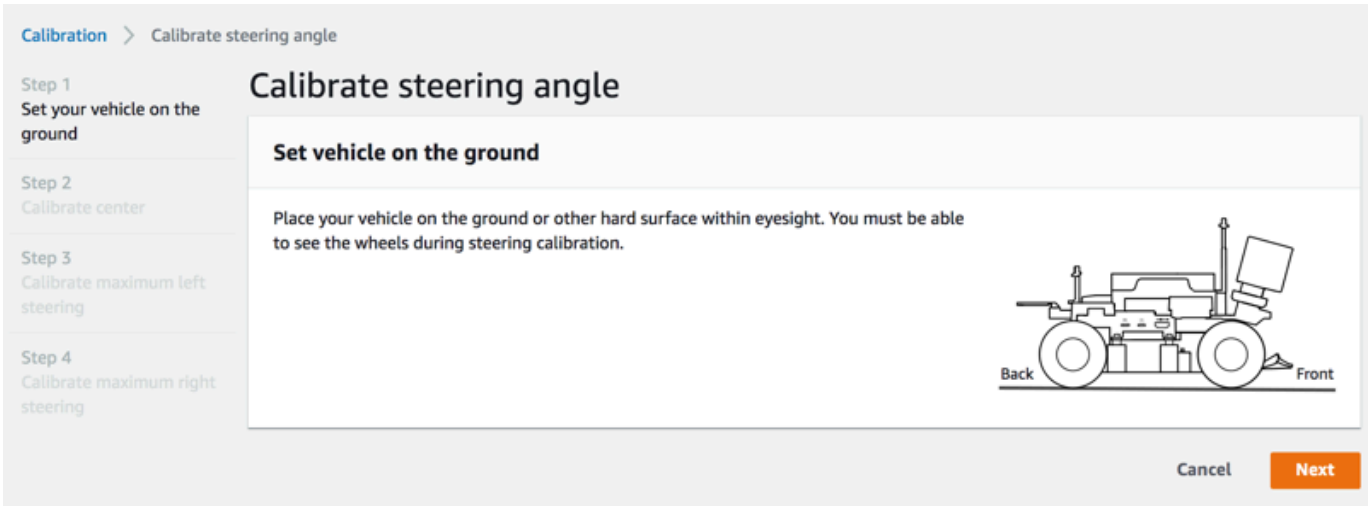
1. Follow the instructions to access your vehicle and open the device control console.

2. Choose **Calibration** from the main navigation pane.
3. On the **Calibration** page, choose **Calibrate** in **Steering** and then follow the steps below to calibrate the vehicle's maximum steering angles.
   a. Set the vehicle on the ground or another hard surface where you can see the wheels during the steering calibration. Choose **Next**.



   Steering a vehicle on a track requires the much smaller steering angles than turning wheels in the air. To measure the actual steering angles of the wheels, it's important that you place the vehicle down onto the track surface.

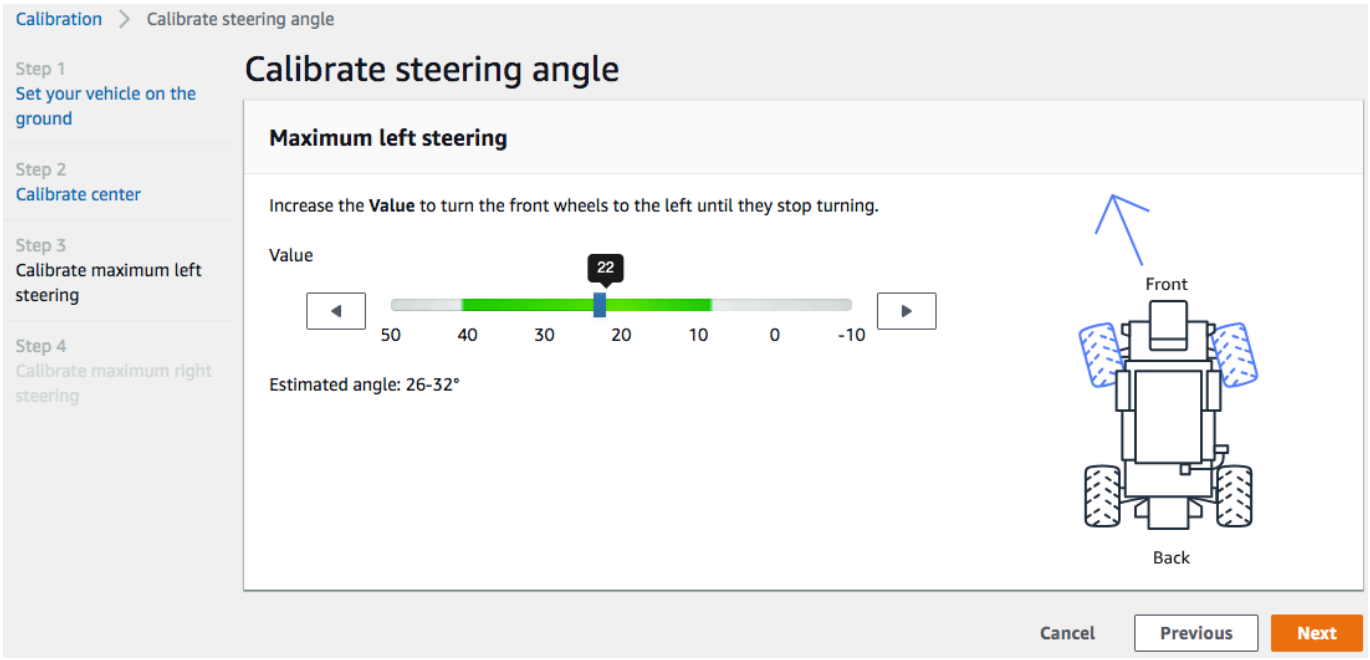
   b. Under **Center steering**, gradually move the slider or press the left or right arrow to the position where at least one of the front wheels is aligned with the rear wheel on the same side. Choose **Next**.



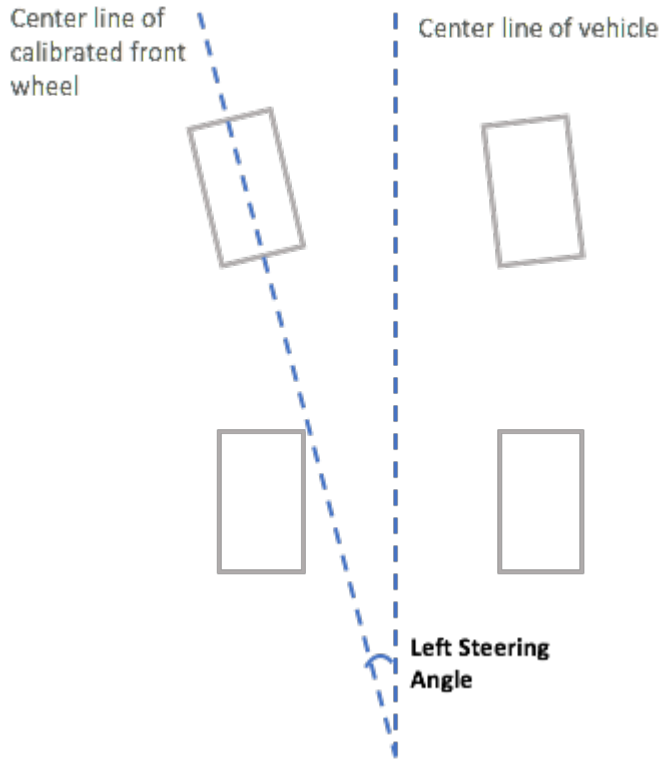
AWS DeepRacer uses Ackermann front-wheel steering to turn wheels on the inside and outside of a turn. This means that the left and right front wheels generally turn at different angles. In AWS DeepRacer, the calibration is done on the center value. Therefore, you need to adjust the wheels on the selected side to be aligned in a straight line.

> **Note**
>
> Make sure to calibrate your AWS DeepRacer vehicle well so that it can maintain center steering as straight as possible. You can test this by manually pushing the vehicle to verify it follows a straight path.

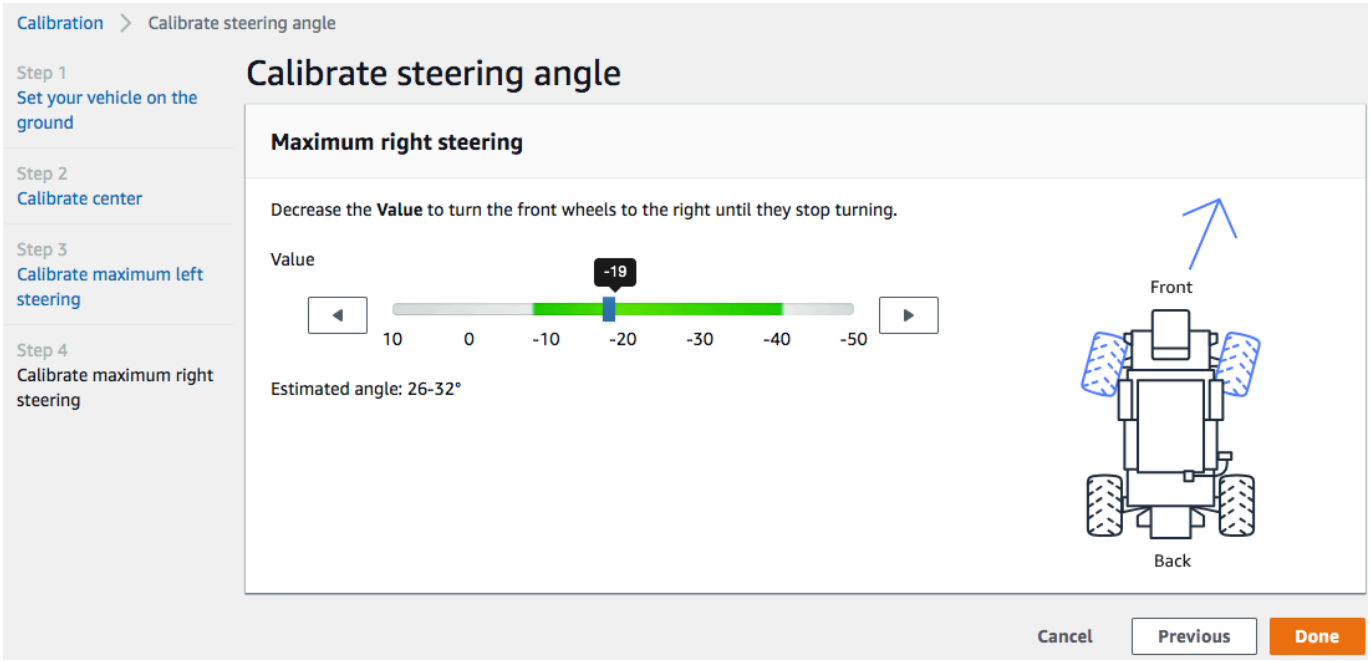
c. Under **Maximum left steering**, gradually move the slider to the left or press the left arrow until the vehicle front wheels stops turning left. There will be a quiet noise. If you hear a loud noise, you have gone too far. The position corresponds to the maximum left steering angle. If you have limited your steering angle in the simulated action space, match the corresponding value here. Choose **Next**.



To measure the actual maximum left steering angle, draw a center line for the vehicle, mark the two edge points of the selected front wheel for calibration, and draw the center line of this front wheel until it crosses over the center line of the vehicle. Use a protractor to measure the angle. See the figure below. If you want to match the actual angle in your training, you can set the same value in the action space in your next training job.



d. Under **Maximum right steering**, gradually move the slider to the right until the selected front wheels stops turning right. There will be a quiet noise. If you hear a loud noise, you have gone too far. The position corresponds to the maximum right steering angle. If you have limited your steering angle in the simulated action space, match the corresponding value here. Choose **Done**.

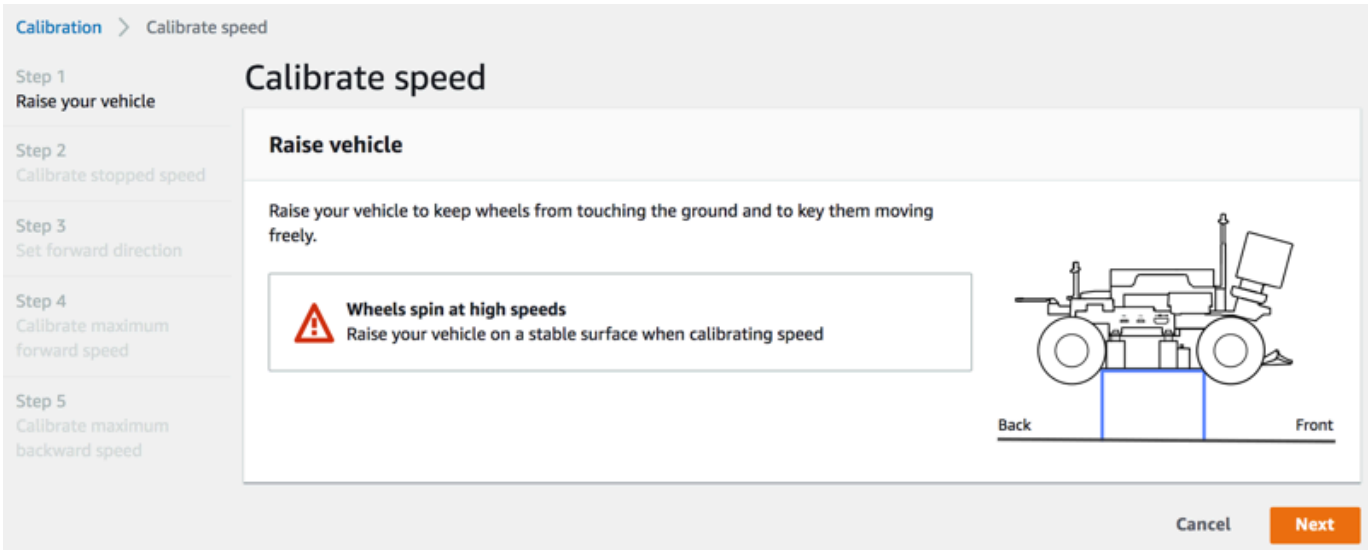


To measure the actual maximum right steering angle, follow the steps similar to those used to measure the maximum left steering angle.

This concludes the steering calibration for your AWS DeepRacer vehicle.

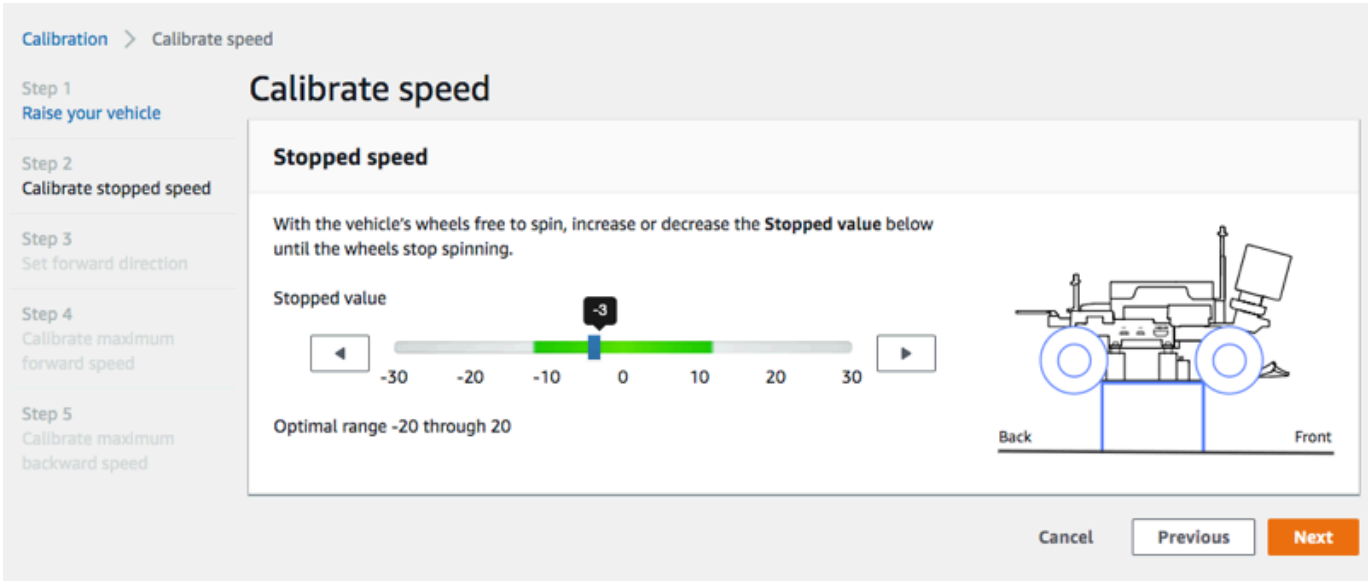
4. To calibrate the vehicle's maximum speed, choose **Calibrate** in **Speed** on the **Calibration** page and then follow the steps below.

   a. Raise the vehicle so that the wheels are free to turn. Choose **Next** on the device control console.



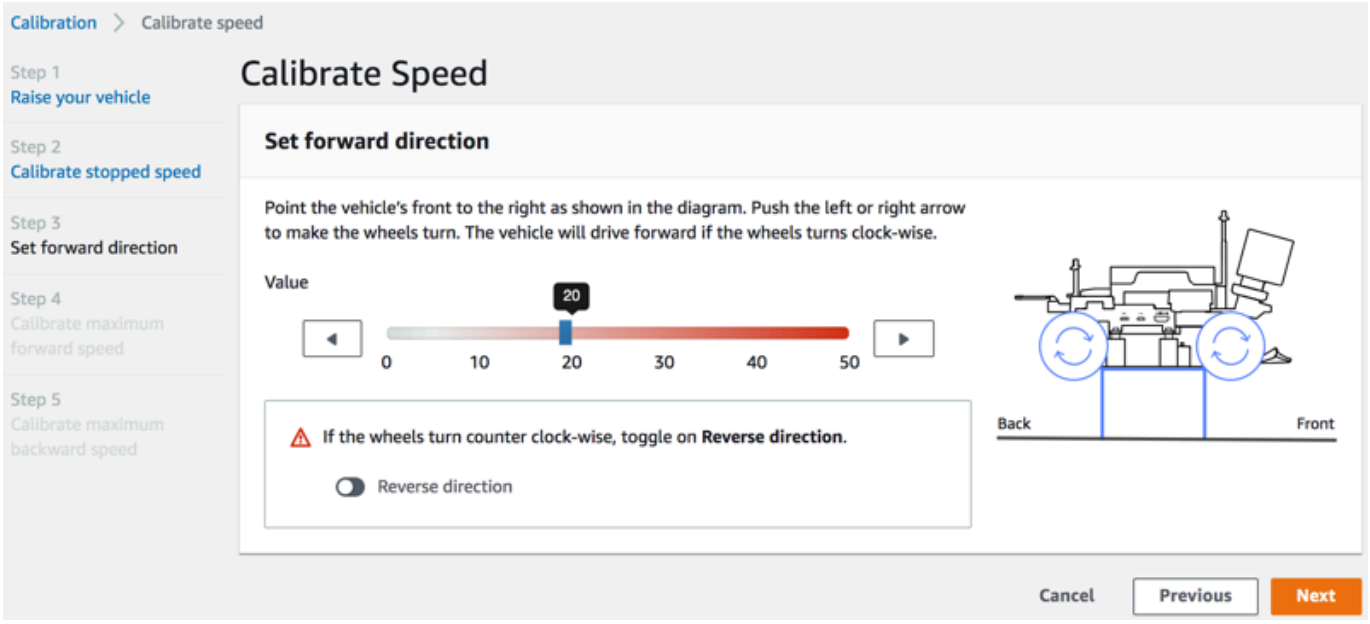
> **Note**
>
> If the vehicle's speed has been set too high, it may run too fast during calibration and cause damage to the environment, the vehicle, or others nearby. You should raise the vehicle, as instructed here, but not hold it in your hands.

b. To calibrate the stopped speed, press the left or right arrow to gradually change **Stopped** value under **Stopped speed** on the device control console until the wheels stop turning. Choose **Next**.



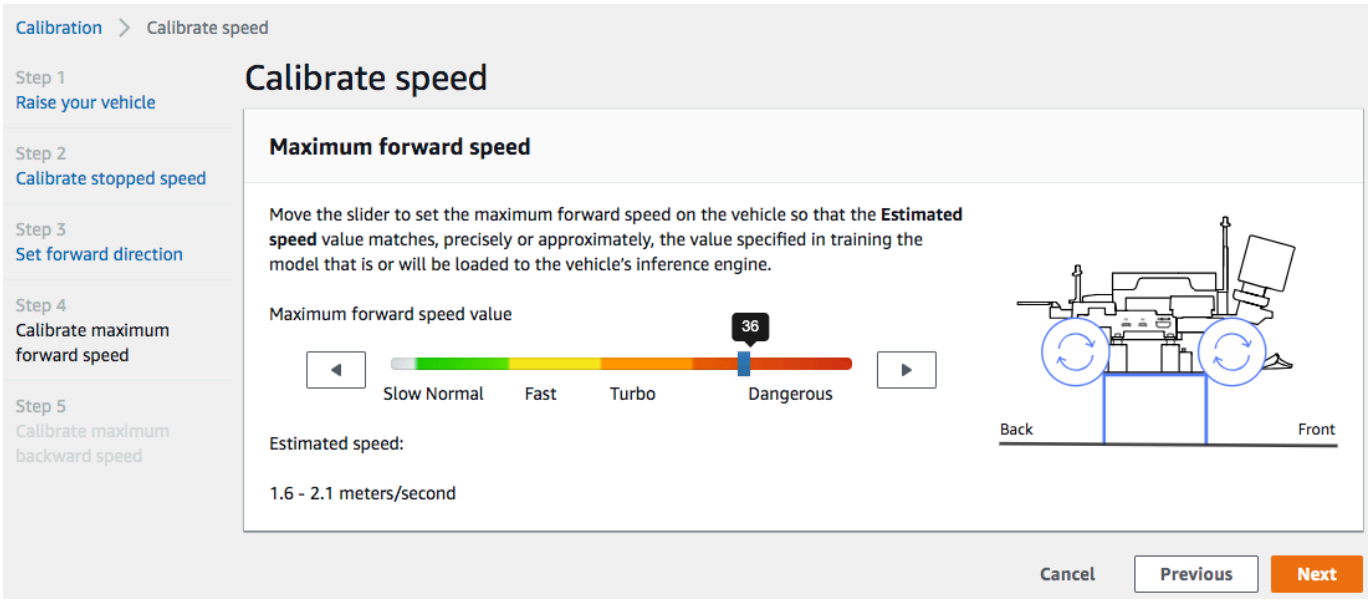
> **Note**
>
> When pressing the **Stopped** value further left or further right to the value when you start hearing noises, the wheels are about to move. The ideal zero-throttle point is the middle of the two values. For example, if you start hearing a noise at 16 on the left and at -4 on the right, the optimal stopped value should be 10.

c. To set the vehicle's forward direction, place the vehicle as shown on the screen and the image here, and then press the left or right arrow to make the wheels turn. If the wheels turn clock-wise, the forward direction is set. If not, toggle **Reverse direction**. Choose **Next**.



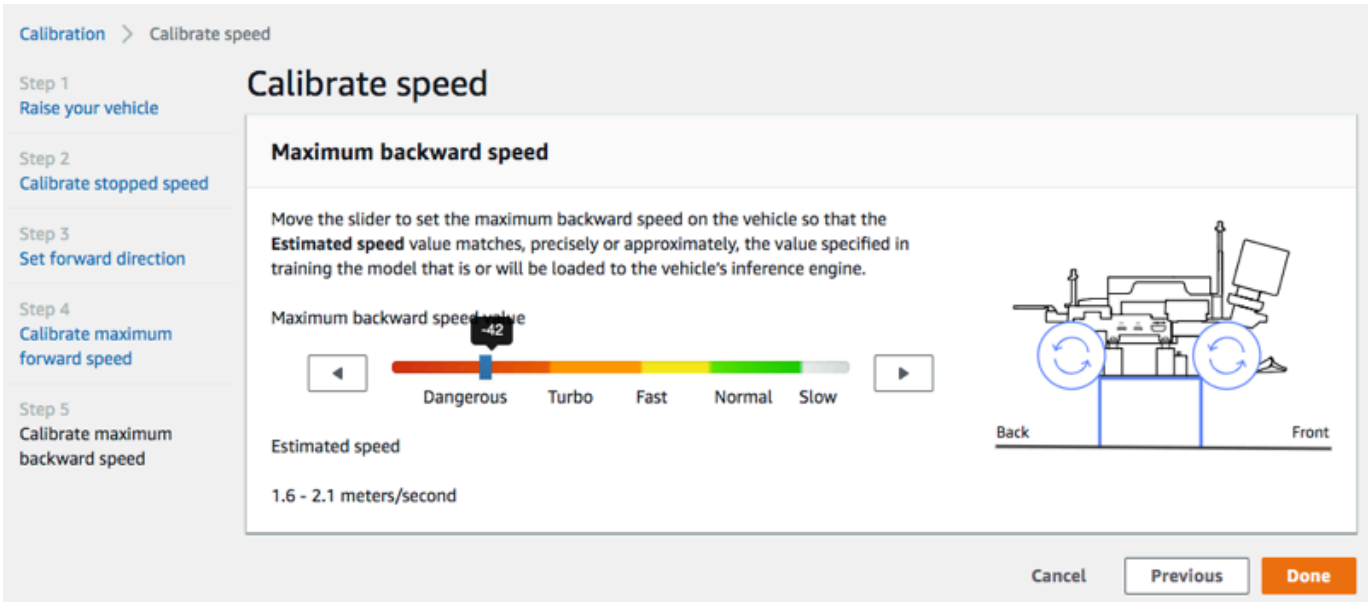
> **Note**
>
> Vehicles distributed at AWS re:Invent 2018 might have their forward direction set in reverse. In such a case, make sure to toggle **Reverse direction**.

d. To calibrate the maximum forward speed, under **Maximum forward speed**, gently move the slider left or right to adjust the **Maximum forward speed** value number gradually to such a positive value that the **Estimated speed** value is equal or similar to the maximum speed specified in the simulation. Choose **Next**.



> **Note**
>
> The actual maximum speed your vehicle depends on the friction of the track surface as well as the vehicle battery level. To make it flexible, you can set the vehicle's throttle limit to be 20-30 percent higher than the maximum speed specified for training in the simulation. Generally speaking, you should set the maximum speed value within the green area. Above that, your vehicle is likely to drive too fast at increased risk of breaking. Additionally, the action space for training doesn't support the maximum speed of more than 2 m/s.

e. To calibrate the maximum backward speed, under **Maximum backward speed**, gently move the slider left or right to adjust the **Maximum backward speed** value number gradually to such a negative value that the **Estimated speed** value is equal or similar to the maximum speed specified in the simulation. Choose **Done**.



> **Note**
>
> The AWS DeepRacer vehicle doesn't use backward speed in the autonomous driving mode. You can set the backward speed to any value with which you can comfortably control the vehicle's manual driving mode.

This concludes calibrating your AWS DeepRacer vehicle's maximum speed.

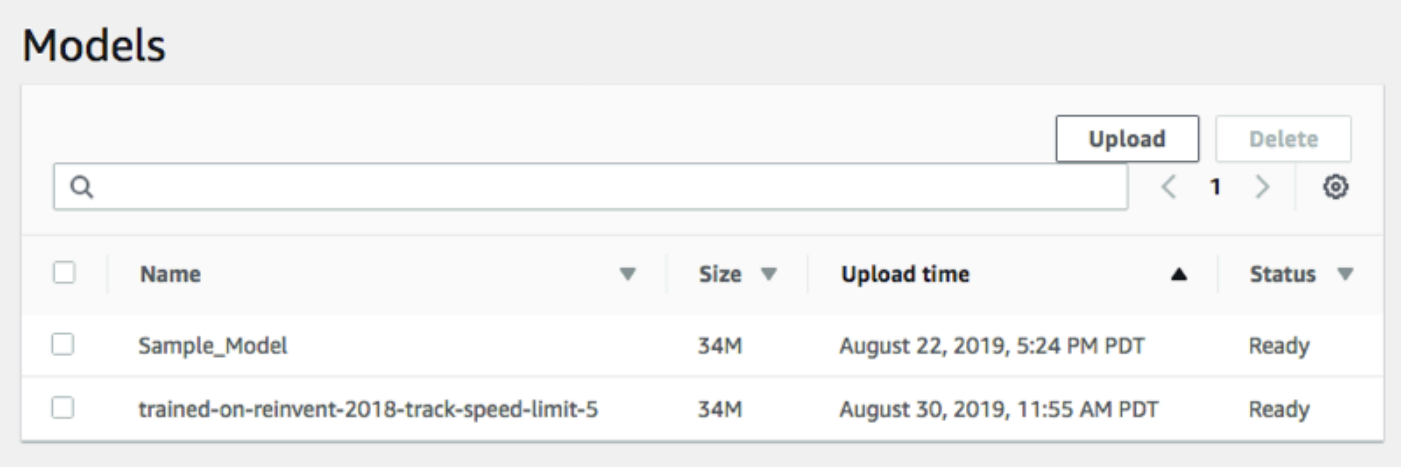
Upload your model

To start your AWS DeepRacer vehicle on autonomous driving, you must have uploaded at least one AWS DeepRacer model to your AWS DeepRacer vehicle.

To upload a model, you must have trained and evaluated the model. You can train the model using the AWS DeepRacer console. After that, you need to download the model artifacts from its Amazon S3 storage to a (local or network) drive that can be accessed by your computer.

To upload a trained model to your vehicle

1. Choose **Models** from the device console's main navigation pane.



2. On the **Models** page, choose **Upload** above the **Models** list.
3. From the file picker, navigate to the drive or share where you've downloaded your model artifacts and choose the the compressed model file (of the `*.tar.gz` extension) to upload.

Only a successfully uploaded model will be added to the **Models** list and can be available for you to load it into the vehicle's inference engine in the autonomous driving mode. For the instructions on how to load a model into your vehicle's inference engine, see Drive your AWS DeepRacer vehicle autonomously.

Drive your vehicle

After setting up your AWS DeepRacer vehicle, you can start to drive your vehicle manually or let it drive autonomously, using the vehicle's device console.

For autonomous driving, you must have trained an AWS DeepRacer model and have the trained model artifacts deployed to the vehicle. In the autonomous racing mode, the model running in the inference engine controls the vehicle's driving directions and speed. Without a trained model downloaded to the vehicle, you can use the vehicle's device console to drive the vehicle manually.

Many factors affect the vehicle's performance in autonomous driving. They include the trained model, vehicle calibration, track conditions, such as surface frictions, color contrasts and light reflections, etc. For your vehicle to achieve an optimal performance, you must make sure that the model transfer from the simulation to the real world is as accurate, relevant and meaningful.

Topics

- Drive your AWS DeepRacer vehicle manually

- Drive your AWS DeepRacer vehicle autonomously

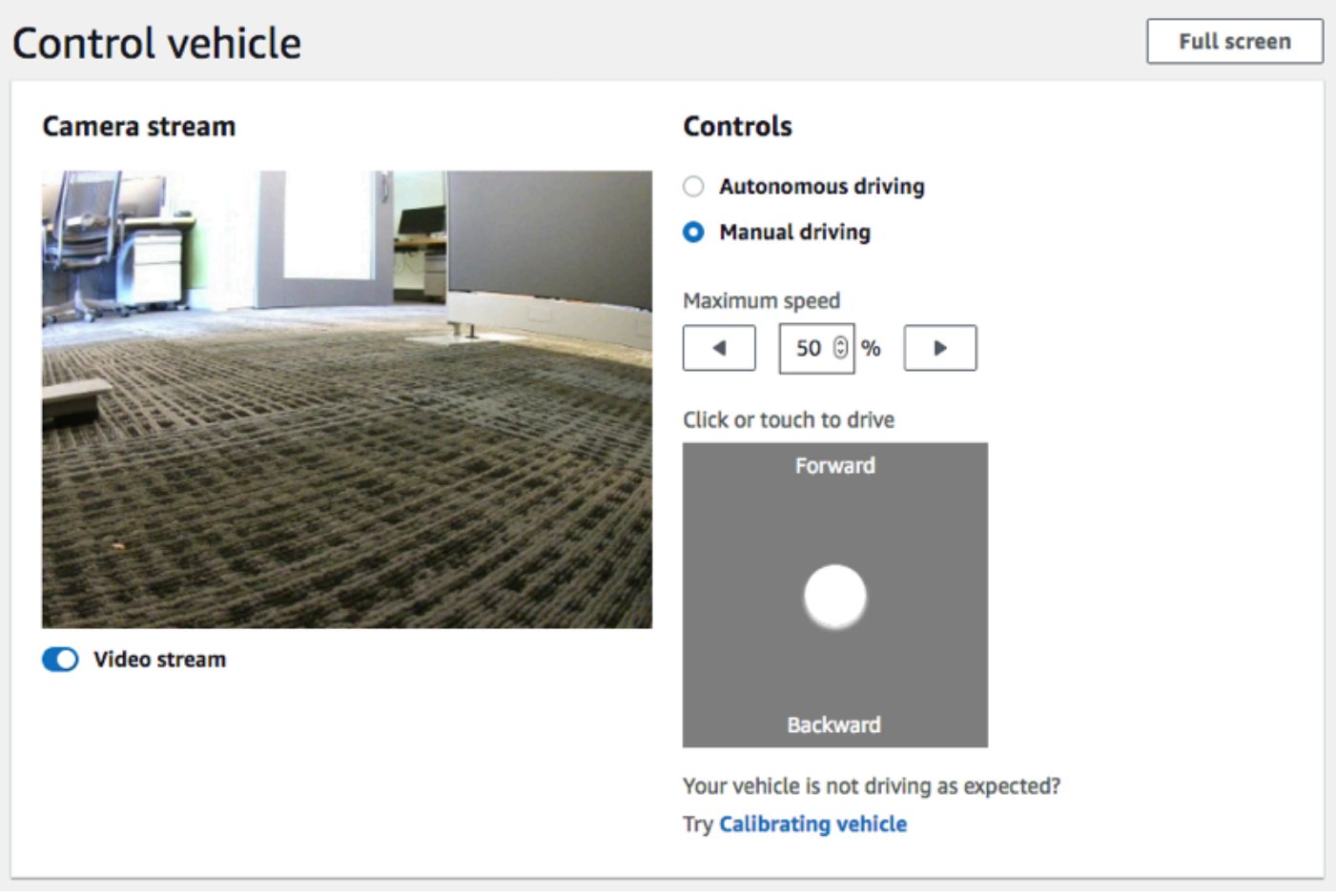
Drive your AWS DeepRacer vehicle manually

If you have not trained any model or have not deployed any trained model to your AWS DeepRacer vehicle, you can't let it drive itself. But you can drive it manually.

To drive a AWS DeepRacer vehicle manually, follow the steps below.

To drive your AWS DeepRacer vehicle manually

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow the instructions to sign into the vehicle's device control console.
2. On the **Control vehicle** page, choose **Manual driving** under **Controls**.



3. Under **Click or touch to drive**, click or touch a position within the driving pad to drive the vehicle. Images captured from the vehicle's front camera are displayed in the video player under **Camera stream**.

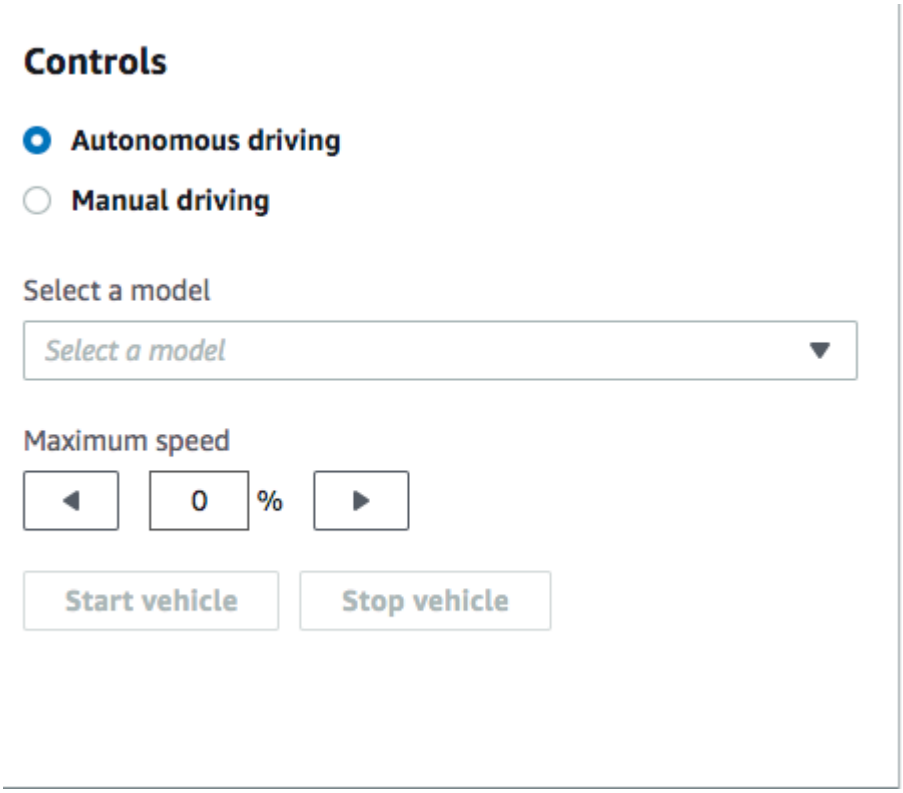
4. To turn video stream on or off on the device console while you drive the vehicle, toggle the **Video stream** option under the **Camera stream** display.
5. Repeat from Step 3 to drive the vehicle to different locations.

Drive your AWS DeepRacer vehicle autonomously

To start autonomous driving, place the vehicle on a physical track and do the following:

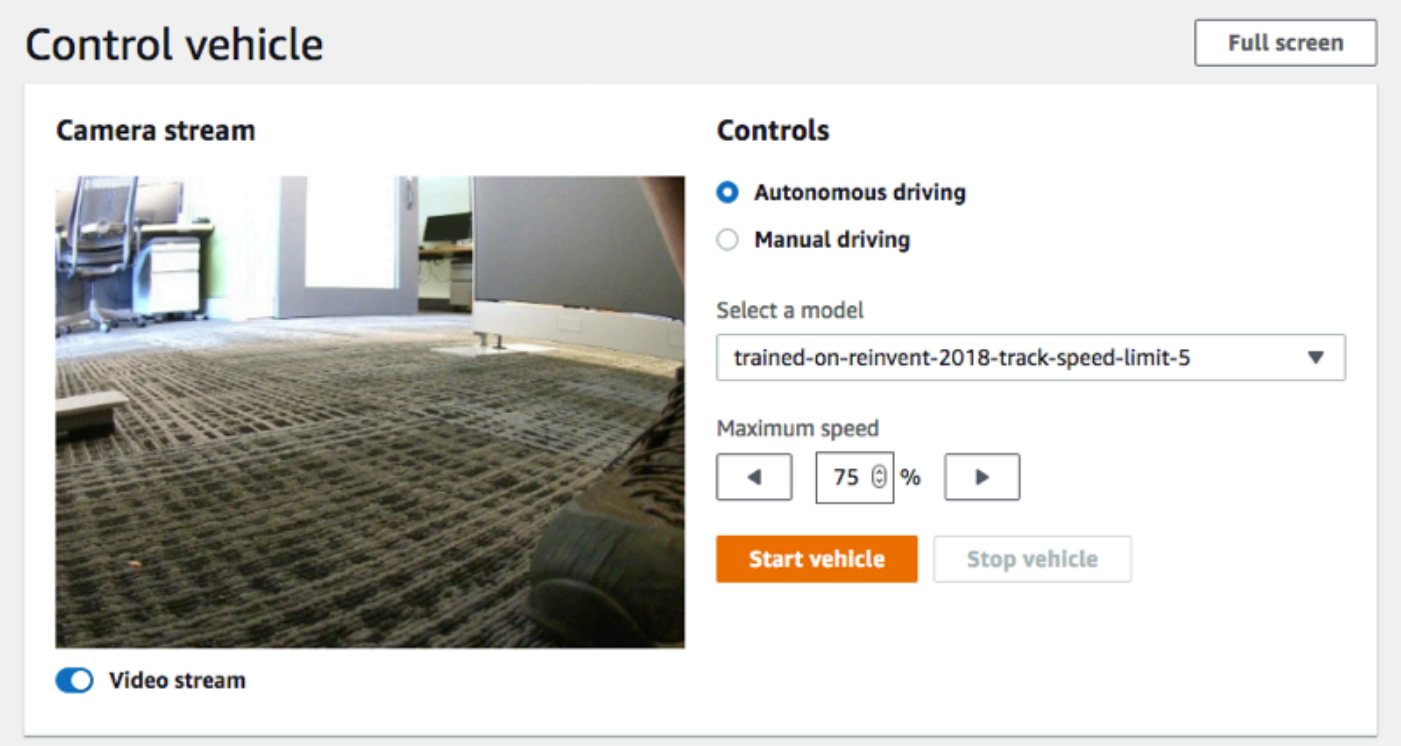
To drive your AWS DeepRacer vehicle autonomously

1. Follow the instructions to sign into the vehicle's device console, and then do the following for autonomous driving:
2. On the **Control vehicle** page, choose **Autonomous driving** under **Controls**.



3. From the **Select a model** drop-down list, choose an uploaded model. Then choose **Load model**. This will start loading the model into the inference engine. The process takes about 10 seconds to complete.
4. Adjust the **Maximum speed** setting of the vehicle to be a percentage of the maximum speed used in training the model.

Certain factors, such as surface friction of the real track, can reduce the maximum speed of the vehicle from the maximum speed used in the training. You'll need to experiment to find the optimal setting.



5. Choose **Start vehicle** to set the vehicle to drive autonomously.
6. To turn video stream on or off on the device console while you drive the vehicle, toggle the **Video stream** option under the **Camera stream** display.
7. Watch the vehicle drive on the physical track or the streaming video player on the device console.
8. To stop the vehicle, choose **Stop vehicle**.

Repeat from Step 3 for another run with the same or a different model.

Inspect and manage vehicle settings

After the initial setup, you can use the AWS DeepRacer device control console to manage your vehicle's settings. The tasks include the following:

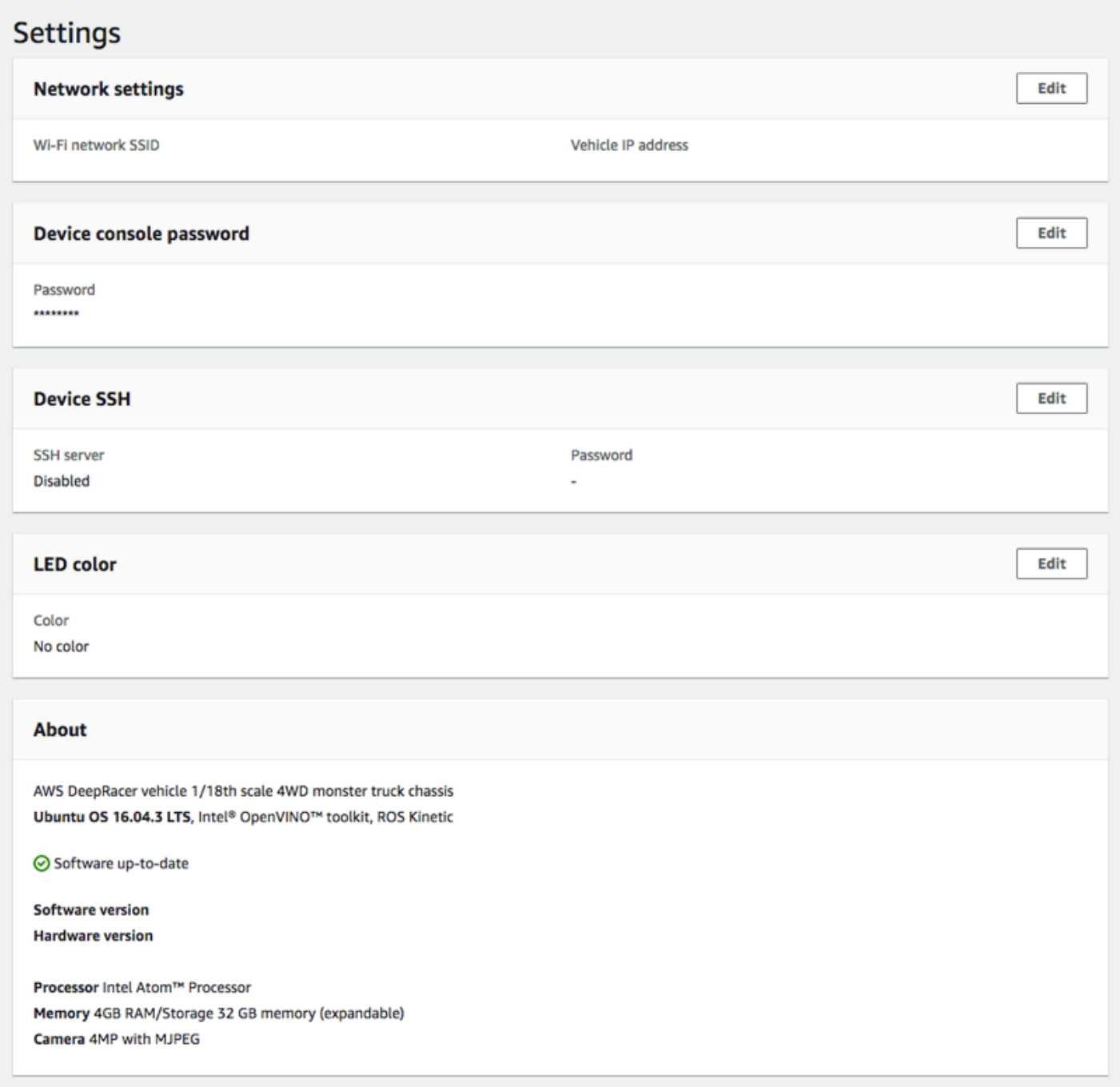
- choosing another Wi-Fi network,
- resetting the device console password,

- enabling or disabling the device SSH settings,
- configuring the vehicle's trail light LED color,
- inspecting the device software and hardware versions,
- checking the vehicle battery level.

The procedure below walks you through these tasks.

To inspect and manage your vehicle's settings

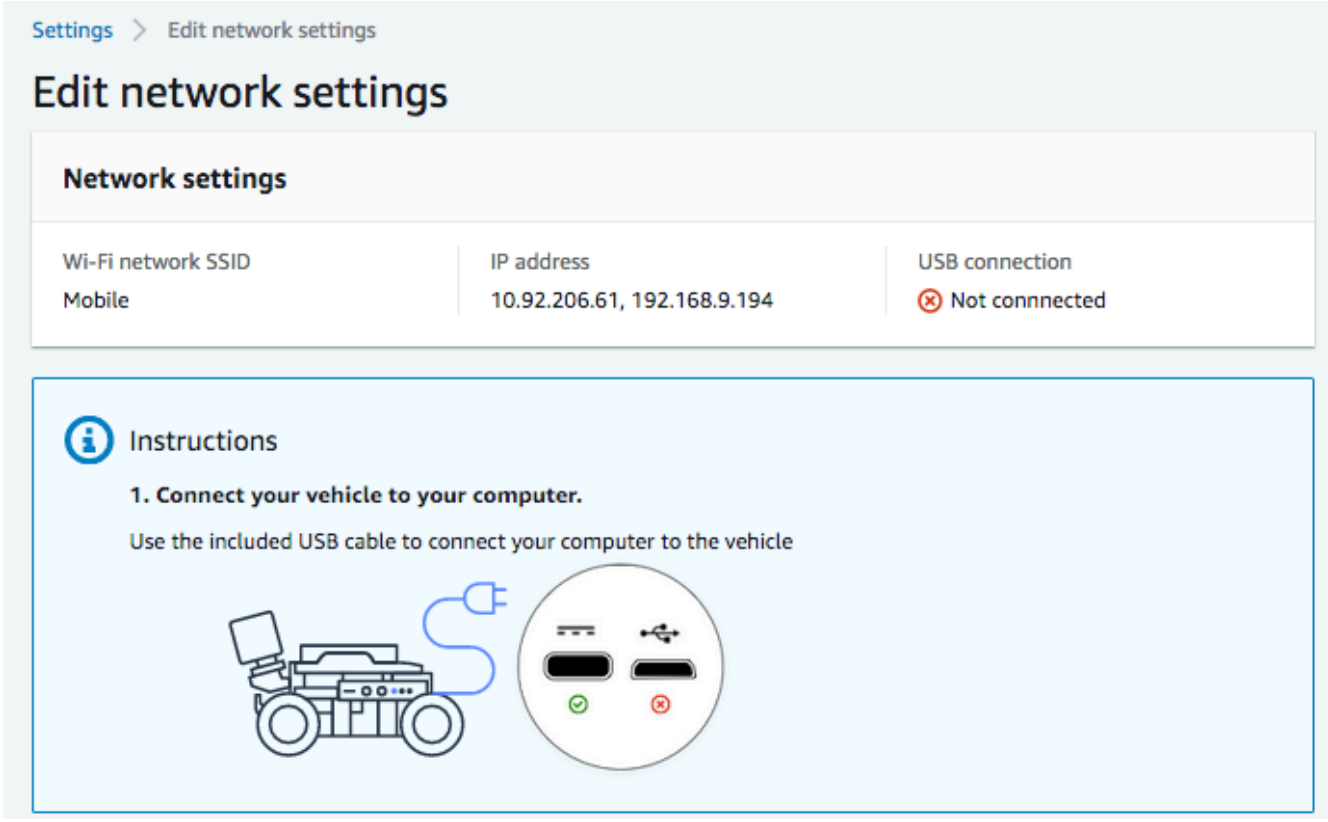
1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow the instructions to sign into the vehicle's device control console.
2. Choose **Settings** from the main navigation pane.



3. On the **Settings** page, perform one or more of the following tasks of your choosing.

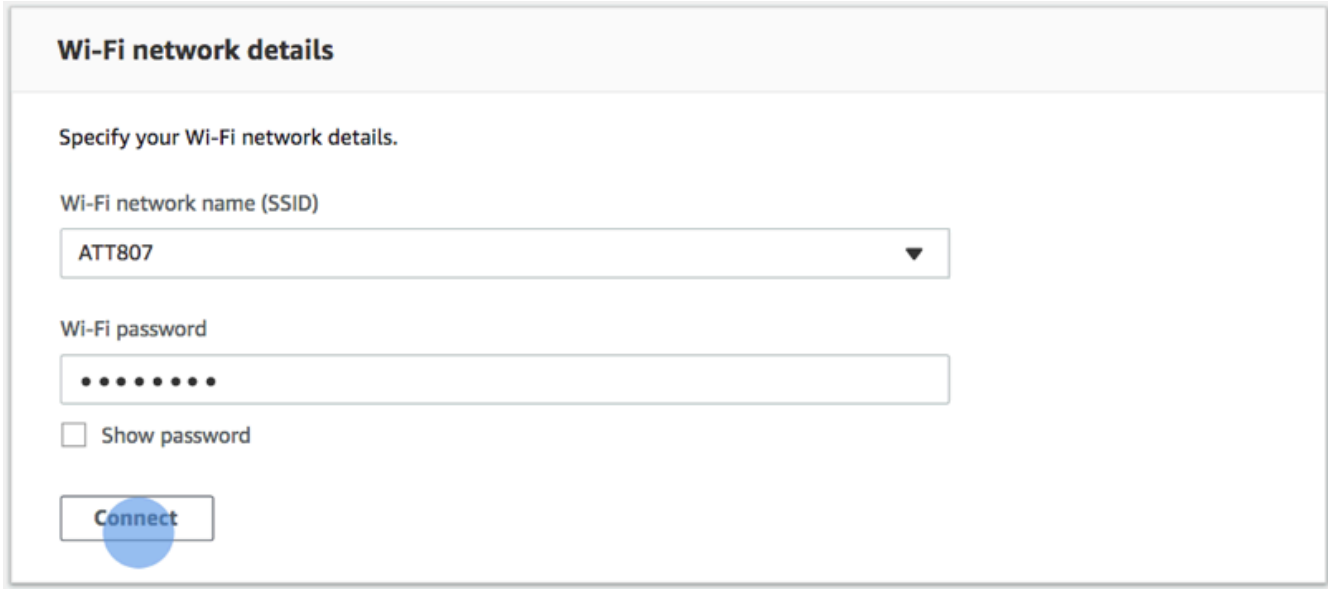
   a. To choose another Wi-Fi network, choose **Edit** for **Network settings** and then follow the steps below.

      i. Follow the instructions, shown on **Edit network settings**, to connect your vehicle to your computer using the USB-to-USB-C cable. After the USB connection status becomes **Connected**, choose the **Go to deepracer.aws** button to open the device console login page.



ii. On the device console login page, type the password printed on the bottom of your vehicle and then choose **Access vehicle**.

iii. Under **Wi-Fi network details**, choose a Wi-Fi network from the drop-down list, type the password of the chosen network, and then choose **Connect**.



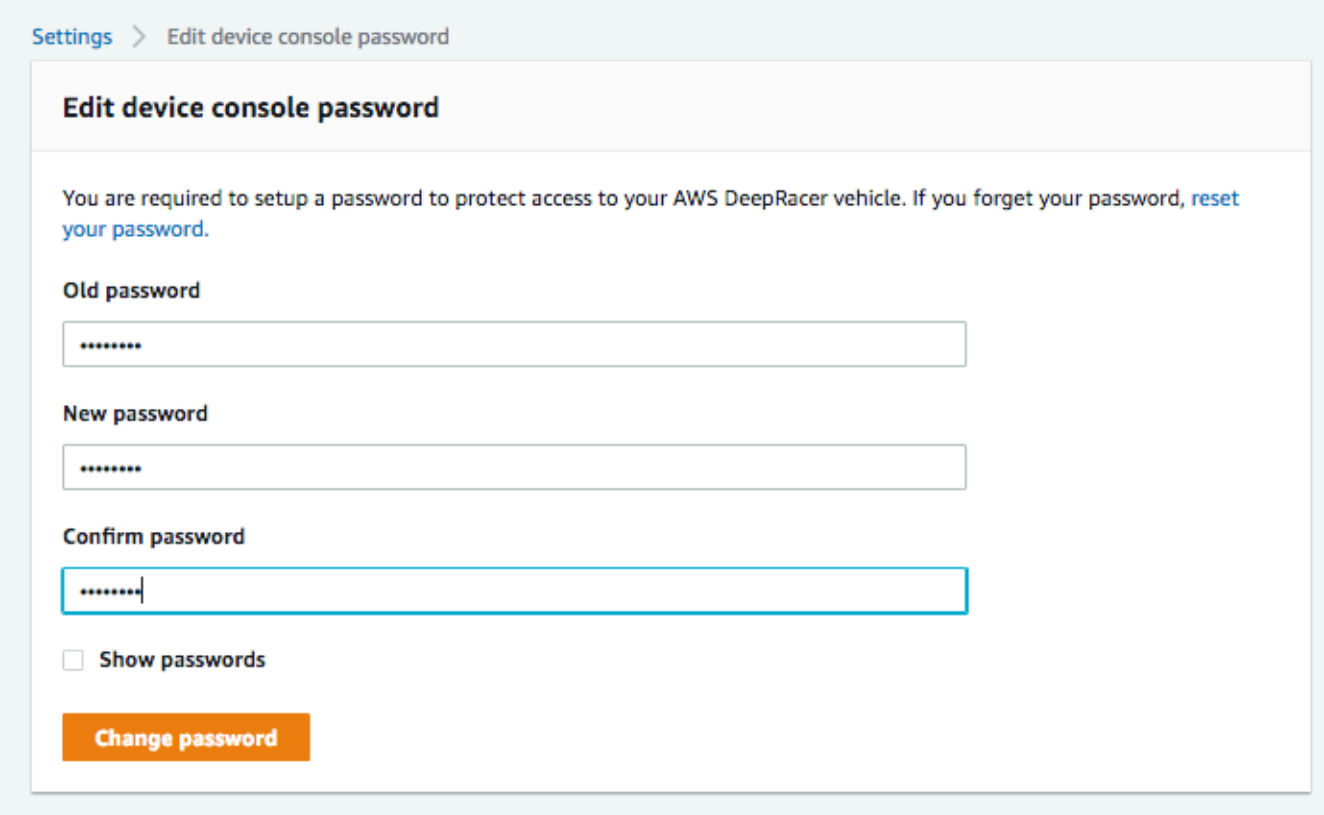
iv. After the **Vehicle status** for the Wi-Fi connection becomes **Connected**, choose **Next** to return to the **Settings** page of the device console, where you'll see a new IP address of the vehicle.

b. To reset the password for signing into the device console, choose **Edit** for **Device console password** and then follow the steps below.

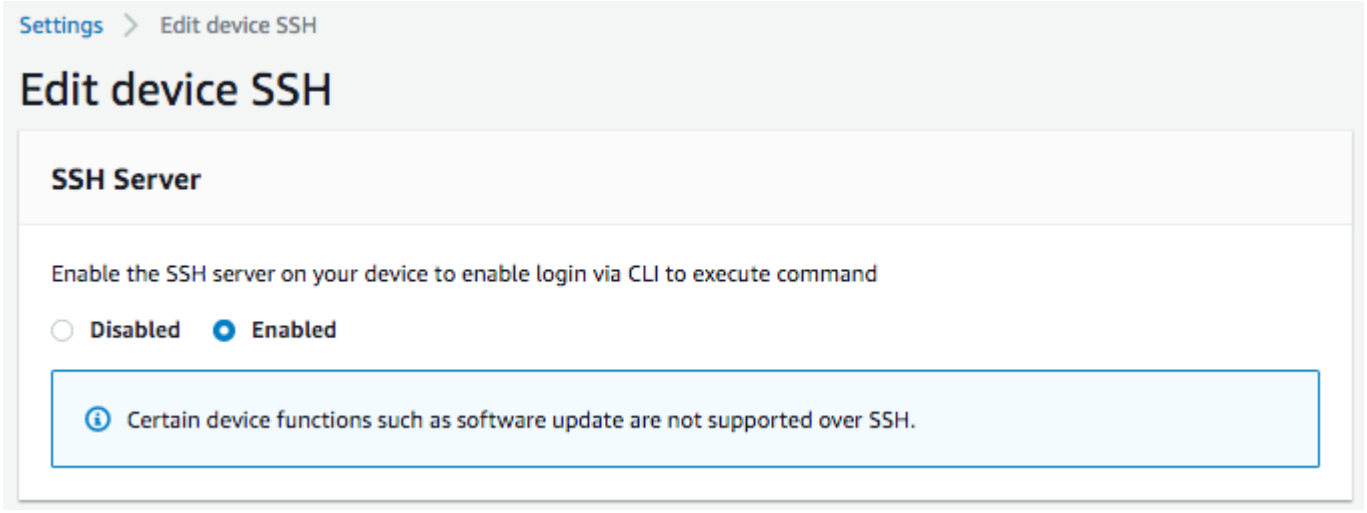
i. On **Edit device console password** page, type a new password in **New password**.

ii. Retype the new password in **Confirm password** to confirm your intention for the change. The password value must be the same before you can move on.

iii. Choose **Change password** to complete the task. This option is activated only if you have entered and confirmed a valid password value in the steps above.



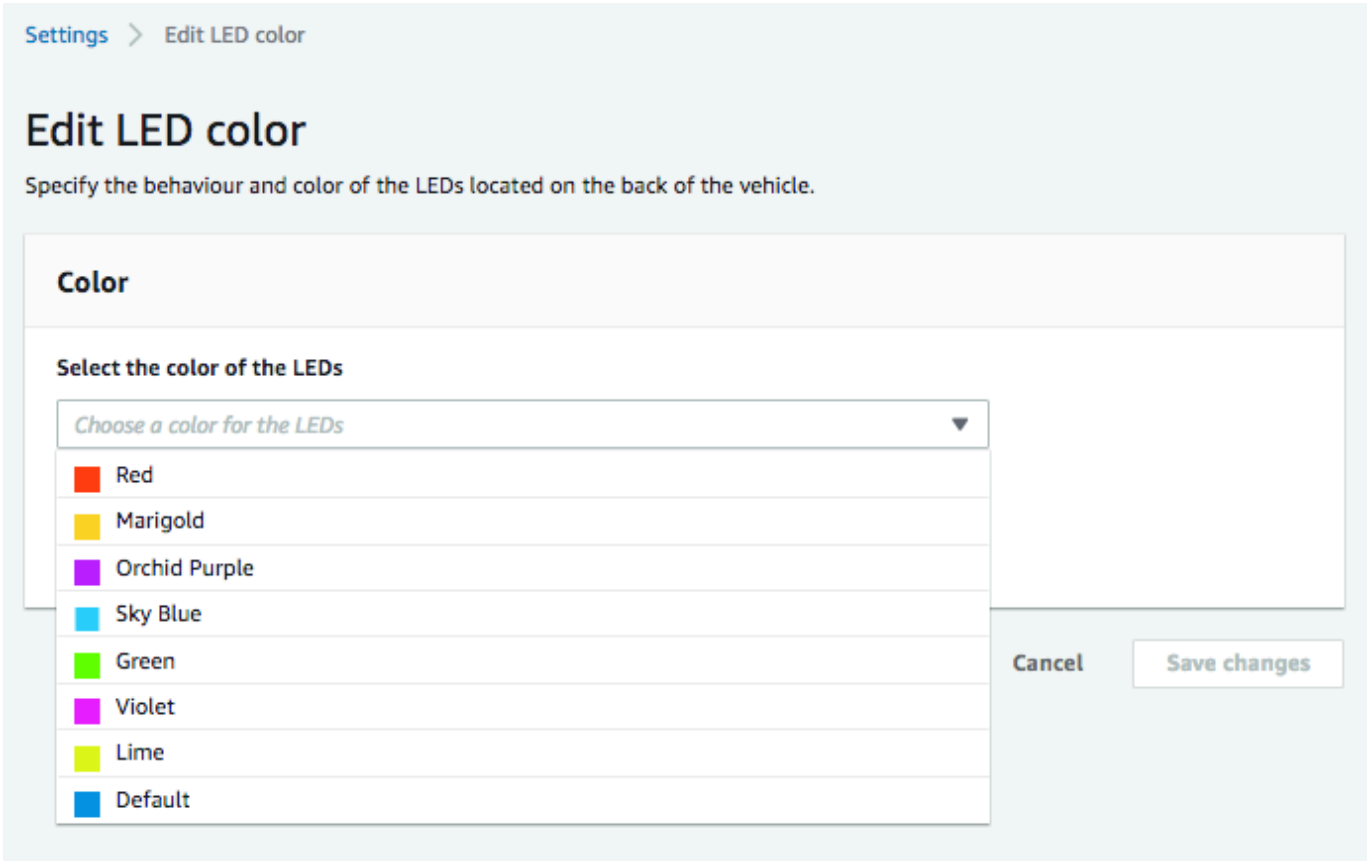
c. To enable or disable SSH connection to the vehicle, choose **Edit** for **Device SSH** and then choose **Enable** or **Disable**.



4. To change the vehicle's trail light LED color to distinguish your vehicle on a track, choose **Edit** for **LED color** on the **Settings** page and do the following.

   a. Choose an available color from the **Select the color of the LEDs** drop-down list on the **Edit LED color** page.

   You should choose a color that can help identify your vehicle from other vehicles sharing the track at the same time.



b. Choose **Save changes** to complete the task.

The **Save changes** functionality becomes active only after you have chosen a color.

5. To inspect the device software and hardware versions and to find out the system and camera configurations, check the **About** section under **Settings**.
6. To inspect the vehicle battery's charge level, check the lower part of the primary navigation pane.

View vehicle logs

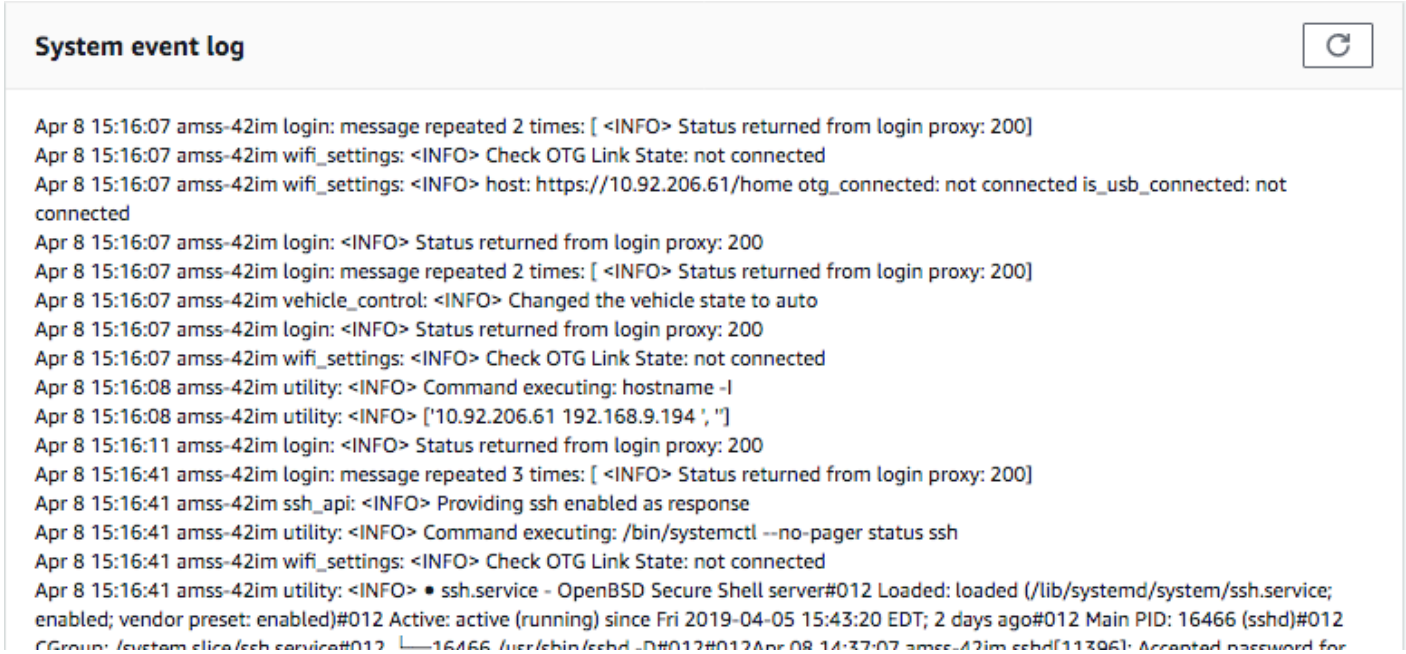
Your AWS DeepRacer vehicle logs operational events that can be helpful for troubleshooting issues encountered in running your vehicle. There are two types of AWS DeepRacer vehicle logs:

- The system event log keeps track of operations taking place in the vehicle's computer operating system, such as process managing, Wi-Fi connecting or password reset events.
- The robot operating system logs record statuses of operations taking place in the vehicle's operating system node for robotic operations, including vehicle driving, video streaming and policy inferencing operations.

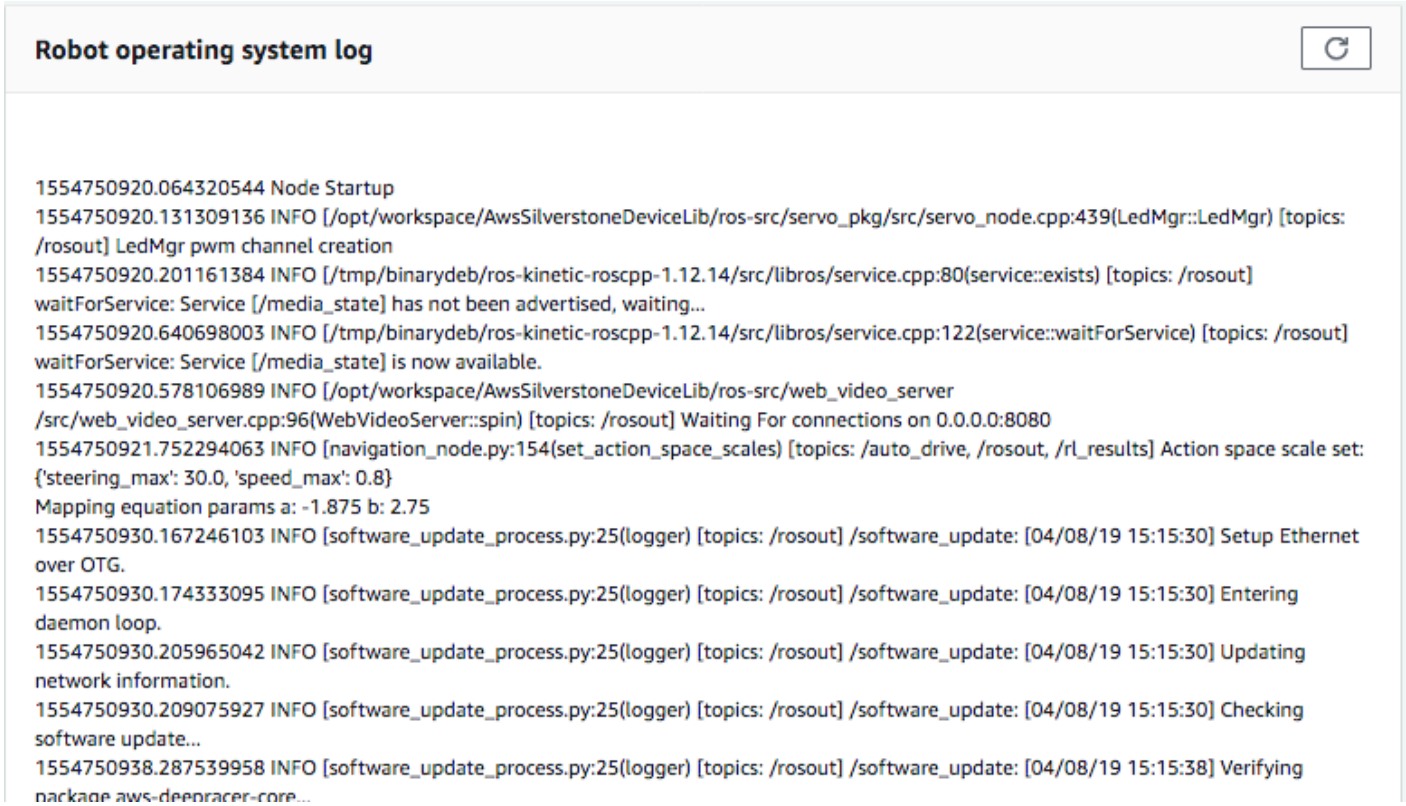
To view the device logs, follow the steps below.

To view your AWS DeepRacer vehicle logs

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow the instructions to sign into the vehicle's device control console.
2. Choose **Logs** from the device console's main navigation pane.
3. To view the system events, scroll down the event list under **System event log**.



4. To view the robot operating system events, scroll down the event list under **Robot operating system log**.



Update your vehicle

Update your AWS DeepRacer device to the latest software stack including Ubuntu 20.04 Focal Fossa, Intel® OpenVINO™ toolkit 2021.1.110, ROS2 Foxy Fitzroy, and Python 3.8. This update is required to run AWS DeepRacer open-source projects but is otherwise optional. AWS DeepRacer only supports Ubuntu 20.04 Focal Fossa and ROS2 Foxy Fitzroy.

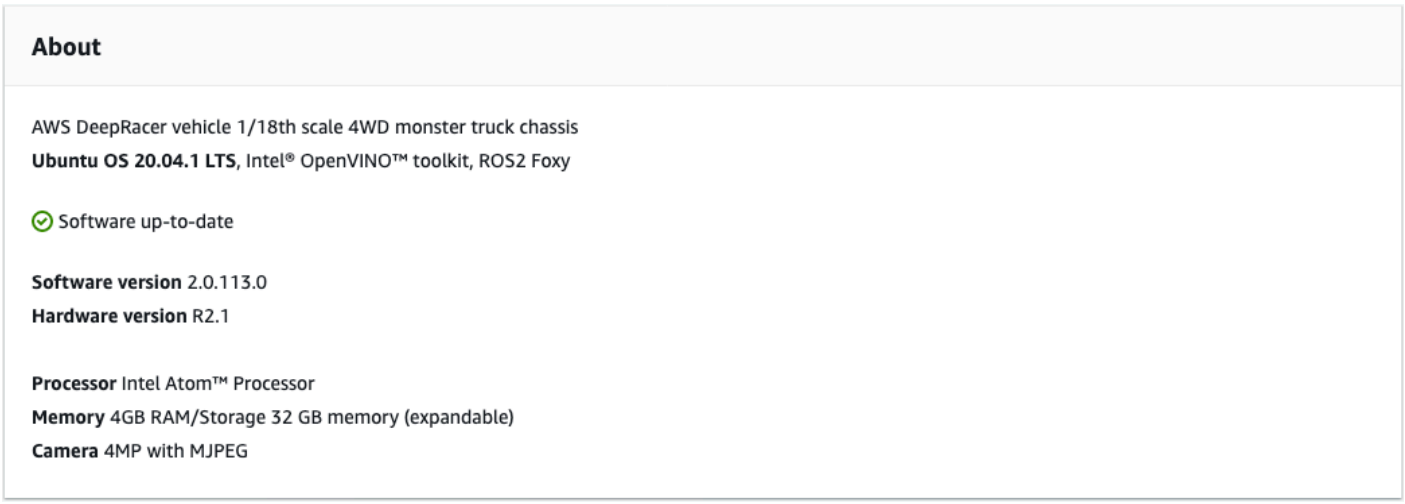
> **Important**
>
> Updating to the new AWS DeepRacer software stack will wipe all data on your AWS DeepRacer device.

Topics

Check which software version your AWS DeepRacer device is currently running

To check which software version your AWS DeepRacer device is currently running

1. Login to the AWS DeepRacer device console. To learn how, follow the steps in the section called "Launch device console".
2. Choose **Settings** on the navigation pane.
3. Check the **About** section to verify which software version your AWS DeepRacer Vehicle is currently running.



Prepare to update your AWS DeepRacer device to the Ubuntu 20.04 software stack

This topic walks you through the process to create the AWS DeepRacer Ubuntu installation media. Preparing the bootable USB drive requires additional hardware.

Prerequisites

Before you get started, make sure you have the following items ready:

- An AWS DeepRacer device
- A USB flash drive (32GB or larger)
- A custom AWS DeepRacer Ubuntu ISO image.
- The latest AWS DeepRacer software update package.
- A copy of UNetbootin compatible with your operating system.
- A computer running Ubuntu, Windows, or macOS to prepare the USB installation media. You can also use the compute module on your AWS DeepRacer device as a Linux computer by connecting a mouse, keyboard, and monitor with an HDMI type A cable.

Preparation

To prepare the AWS DeepRacer update media, you will perform the following tasks:

- Format the USB drive into the following two partitions:

- A 4GB, FAT32 boot partition
- An NTFS data partition of at least 18GB

- Make the USB drive bootable to start the update on reboot:
  - Burn the required custom Ubuntu ISO image to the boot partition
  - Copy the required update files to the data partition of the USB drive

Prepare a bootable USB drive

Follow these instructions to prepare your AWS DeepRacer update media on Ubuntu (Linux), Windows, or macOS. Depending on the computer you use, specific tasks may differ from one operating system to another. Choose the tab corresponding to your operating system.

Topics

Ubuntu

Follow the instructions here to use an Ubuntu computer, including your AWS DeepRacer device's compute module, to prepare the update media for your AWS DeepRacer device. If you are using a different Linux distribution, replace the `apt-get *` commands with those compatible with your operating system's package manager.

To erase and partition the USB drive

1. Run the following commands to install and launch GParted.

```
sudo apt-get update; sudo apt-get install gparted
sudo gparted
```

2. To erase your USB drive, you will need its device path. To find it on the GParted console and erase the USB drive, do the following:
   a. On the menu bar, choose **View**, then choose **Device Information**. A sidebar showing the selected disk's **Model**, **Size**, and **Path** will appear.

b. Select your USB drive by going to **GParted** on the menu bar, then **Devices**, finally, select your USB drive from the list. Match the **Size** and **Model** shown in the Device Description with your USB drive.

c. Once you are sure that you've selected the correct disk, delete all its existing partitions. If the partitions are locked, open the context (right-click) menu and choose **unmount**.

3. To create the FAT32 boot partition with a 4GB capacity, select the file icon on the top-left, set the following parameters, and choose **Add**.

```
Free space preceding: 1
New size: 4096
Free space following: <remaining size>
Align to: MiB
Create as: Primary Partition
Partition name:
Filesystem: fat32
Label: BOOT
```

4. To create the NTFS data partition with a minimum 18GB capacity, select the file icon, set the following parameters, and choose **Add**.

```
Free space preceding: 0
New size: <remaining size>
Free space following: 0
Align to: MiB
Create as: Primary Partition
Partition name:
Filesystem: ntfs
Label: Data
```

5. On the menu bar, choose **Edit**, then **Apply All Operations**. A warning prompt will appear asking if you want to apply the changes. Choose **Apply**.

6. After the FAT32 and NTFS partitions are created, the USB drive's partition information will appear in the GParted console. Make note of the BOOT partition's drive path, you will need it to complete the next step.

To make the USB drive bootable from the FAT32 partition

1. Make sure you downloaded the custom Ubuntu ISO image from the pre-requisites section.

2. If you're using Ubuntu 20.04, you need to run UNetbootin using its binary file. To do this:

a. Download the latest UNetbootin binary file to your Downloads folder. In our example, we use `unetbootin-linux64-702.bin`.

b. Press `Ctrl+Alt+T` to open a new terminal window. Alternatively, choose **Activities** on the menu bar, enter `terminal` in the search bar, then select the Terminal icon.

c. Use the following commands to navigate to the binary file location, give the file execute permission, and run UNetbootin. Make sure to adjust the file name in the commands if the version doesn't match the one on your downloaded binary file.

```
cd Downloads
sudo chmod +x ./unetbootin-linux64-702.bin
sudo ./unetbootin-linux64-702.bin
```

If you're using an older version of Ubuntu, install UNetbootin from its repository by running the following commands:

```
sudo add-apt-repository ppa:gezakovacs/ppa
sudo apt-get update; sudo apt-get install unetbootin
sudo unetbootin
```

3. On the UNetbootin console, do the following:

a. Select the **Diskimage** radio button.

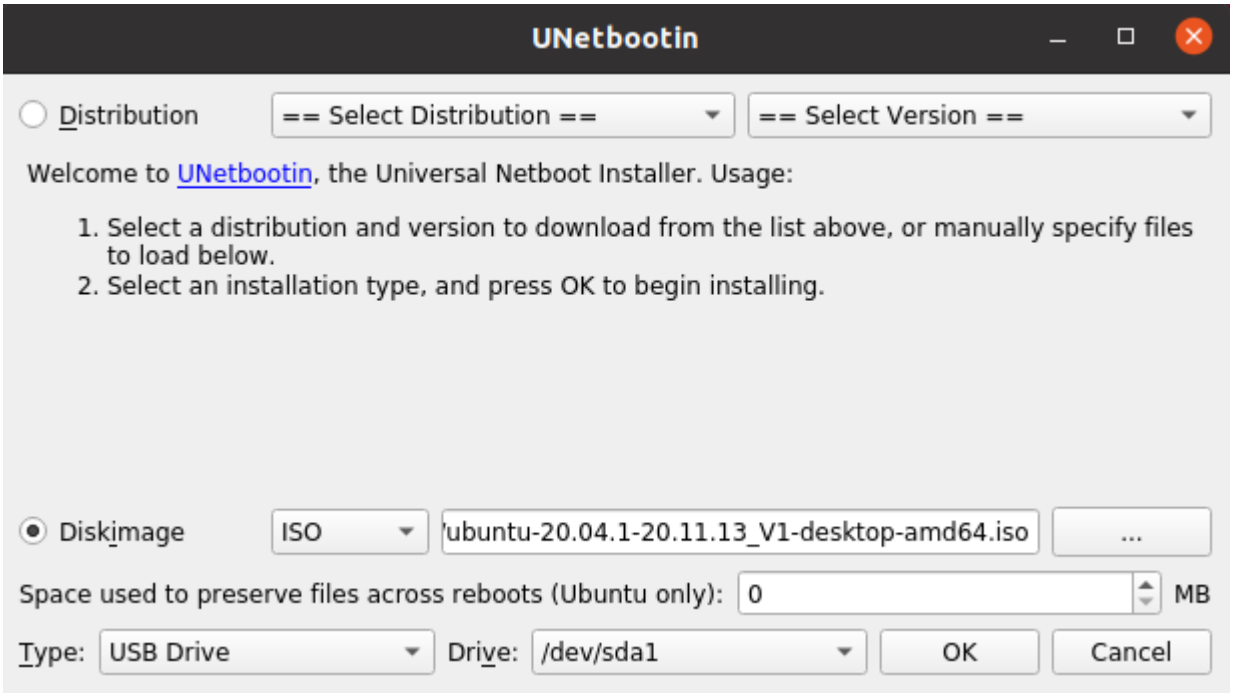
b. For the disk image type, choose **ISO** from the drop-down list.

c. Open the file selector and choose the Ubuntu ISO provided in the pre-requisites section.

d. For **Type**, choose **USB Drive**.

e. For **Drive**, choose the drive path for your BOOT partition, in our case `/dev/sda1`.

f. Choose **OK**.



Tip

If you get a `/dev/sda1 not mounted` alert message, choose **OK** to close the message, unplug the USB drive, plug in the drive again, and then follow the preceding steps to create the Ubuntu ISO image.

To extract the AWS DeepRacer update files to the NTFS partition

1. Unzip the software update package you downloaded from the prerequisites section.
2. Extract the contents of the update package to the root of your USB drive's Data (NTFS) partition.

Windows

Follow the instructions here to use a Windows computer to prepare the update media for your AWS DeepRacer device.

To erase the USB drive

1. Open the Windows command prompt, enter `diskpart`, and choose **OK** to launch Windows DiskPart.

2. Once the terminal for Microsoft DiskPart opens, list the available disks to find the USB drive you want to clean by entering `list disk` after the `DISKPART>` prompt.
3. Select the disk corresponding to your USB drive. For example, we entered `select Disk 2` after the `DISKPART>` prompt. Read the output carefully to verify that you have chosen the disk you want to clean because the next step is irreversible.
4. Once you are sure that you've selected the correct disk, enter `Clean` after the `DISKPART>` prompt.
5. Enter `list disk` after the `DISKPART>` prompt again. Find the disk you cleaned on the table and compare the disk size to the free disk space. If the two values match, the cleaning was successful.
6. Exit the Windows DiskPart console by entering `Exit` after the `DISKPART>` prompt.

To partition the USB drive

1. Open the Windows command prompt, enter `diskmgmt.msc`, and choose **OK** to launch the Disk Management console.
2. From the Disk Management console, select your USB drive.
3. To create the FAT32 partition with a 4GB capacity, open the context (right-click) menu on your USB drive's **Unallocated** space and choose **New Simple Volume**. The **New Simple Volume Wizard** will appear.
4. Once the New Simple Volume Wizard appears, do the following:
   a. On the **Specify Volume Size** page, set the following parameter and then choose **Next**.

   ```
   Simple volume size in MB: 4096
   ```

   b. On the **Assign Drive Letter or Path** page, check the **Assign the following drive letter:** radio button and select a drive letter from the dropdown list, then choose **Next**. Make note of the assigned drive letter, you will need it later to make the FAT32 partition bootable.
   c. On the **Format Partition** page, check the **Format this volume with the following settings** radio button and set the following parameters, then choose **Next**.

   ```
   File system: FAT32
   Allocation unit size: Default
   Volume label: BOOT
   ```

Leave **Perform a quick format** checked.

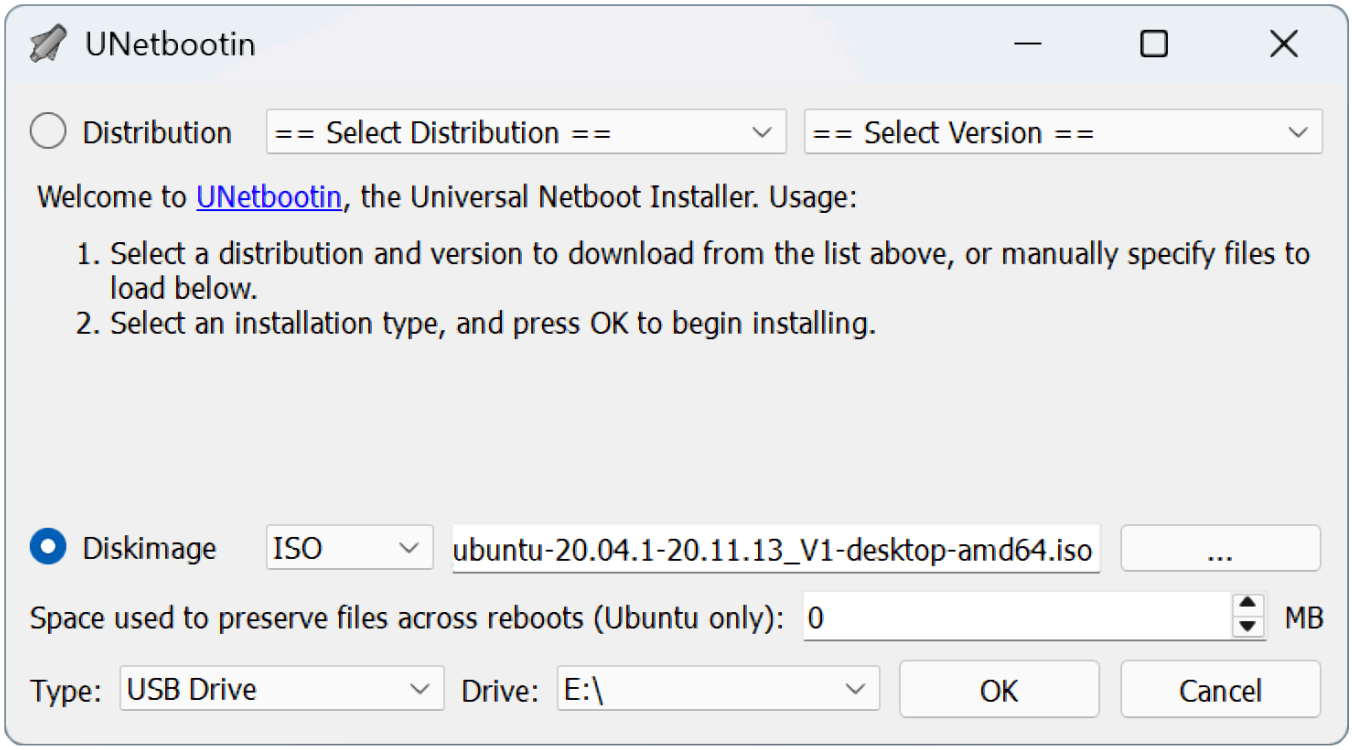
5. To create the NTFS partition with the remaining disk capacity, open the context (right-click) menu on your USB drive's remaining **Unallocated** space and choose **New Simple Volume**. The **New Simple Volume Wizard** will appear.
6. Once the New Simple Volume Wizard appears, do the following:
   a. On the **Specify Volume Size** page, set the **Simple volume size in MB** to match the **Maximum disk space in MB**, then choose **Next**.
   b. On the **Assign Drive Letter or Path** page, check the **Assign the following drive letter:** radio button and select a drive letter from the dropdown list, then choose **Next**.
   c. On the **Format Partition** page, check the **Format this volume with the following settings** radio button and set the following parameters, then choose **Next**.

```
File system: NTFS
Allocation unit size: Default
Volume label: Data
```

   Leave **Perform a quick format** checked.

To make the USB drive bootable from the FAT32 partition

1. Make sure you've downloaded the customized Ubuntu ISO image from the prerequisites section.
2. After downloading UNetbootin, start the UNetbootin console.
3. On the UNetbootin console, do the following:
   a. Check the **Diskimage** radio button.
   b. For disk image, choose **ISO** from the drop-down list.
   c. Open the file picker and choose the custom Ubuntu ISO file.
   d. For **Type**, choose **USB Drive**.
   e. For **Drive**, choose the drive letter corresponding to the FAT32 partition you created. In our case, it's E**:**\.
   f. Choose **OK**.



To extract the AWS DeepRacer update files to the NTFS partition

1. Unzip the software update package you downloaded from the prerequisites section.

> **Tip**
>
> If your favorite tool can't unzip the file successfully, try using the PowerShell `Expand-Archive` command.

2. Extract the contents of the update package to the root of your USB drive's Data (NTFS) partition.

macOS

Follow the instructions here to use a Mac to prepare the update media for your AWS DeepRacer device.

To erase and partition the USB drive

1. Plug in the USB drive to your Mac.

2. Press `Command+Spacebar` to open the Spotlight search field, then enter `Disk Utility`. Alternatively, you can choose **Finder > Applications > Utilities > Disk Utility** to open Disk Utility.
3. On the menu bar, choose **View**, then **Show All Devices**.
4. In the sidebar, under **External**, select the USB drive that you want to format and then choose **Erase**.
5. A new window will ask you to confirm that you want to erase your USB drive and will allow you to change its **Name**, **Format**, and **Partition Scheme**. You don't need to change the name yet, for **Format** and **Scheme**, select the following options and choose **Erase**.
   - **Format**: Mac OS Extended (Journaled)
   - **Scheme**: GUID Partition Map

   Once the erase process is complete, choose **Done** on the dialog window.
6. On the main Disk Utility window, select your USB drive from the sidebar, choose **Partition** from the toolbar on the top. A window titled **Partition device "YOUR-USB-DRIVE"?** will popup. Select the add (+) button to create a new partition.
7. Once you create the new partition, under **Partition Information**, choose and enter the following:
   - **Name**: BOOT
   - **Format**: MS-DOS (FAT)
   - **Size**: 4GB

   > **Tip**
   >
   > If the **Size** input box is grayed out after choosing MS-DOS (FAT) as the format, you can drag the resize control on the partition graph until the BOOT partition is 4GB.

   Do not choose **Apply** yet.
8. Select the other **Untitled** partition, choose and enter the following options under **Partition Information**:
   - **Name**: Data
   - **Format**: ExFAT
   - **Size**: the remaining space of the USB drive (in GB)

Choose **Apply**.

9. A new window will popup and show you the changes that will be made to the USB Drive. Verify that these changes are correct. To confirm and begin the creation of the new partitions, choose **Partition**.
10. On the Disk Utility console, choose the BOOT partition from the Sidebar, then select **Info** from the Toolbar. Make note of the **BSD device node** value, it might be different from the one used in this tutorial. In our case, the value assigned is `disk4s2`. You need to supply this path when making the USB drive bootable from the FAT32 partition.

To make the USB drive bootable from the FAT32 partition

1. Make sure you've downloaded the customized Ubuntu ISO image from the prerequisites section.
2. After downloading UNetbootin, select **open** from the context (right-click) menu. A security prompt will appear asking if you want to open the application, select **open** to start the UNetbootin console.

   If you are using a Mac with Apple Silicon, and the UNetbootin console does not show after selecting open, make sure that Rosetta 2 is installed by following these steps:

   a. Open a terminal window by choosing **Finder > Applications > Utilities > Terminal**.

   b. Enter the following command to install Rosetta 2:

   ```
   softwareupdate --install-rosetta
   ```

   c. Retry opening UNetbootin.
3. On the UNetbootin console, do the following:

   a. Check the **Diskimage** radio button.

   b. For disk image, choose **ISO** from the drop-down list.

   c. Open the file picker and choose the custom Ubuntu ISO file.

   d. For **Type**, choose **USB Drive**.

   e. For **Drive**, choose the BSD device node for your BOOT partition, in our case, `/dev/disk4s2`.

   f. Choose **OK**.

Tip

If you get a `/dev/disk4s2 not mounted` alert message, choose **OK** to close the message, unplug the USB drive, replug the drive, and then follow the steps above create the Ubuntu ISO image.

To extract the AWS DeepRacer update files to the ExFAT partition

1. Unzip the software update package you downloaded from the prerequisites section.
2. Extract the contents of the update package to the root of your USB drive's Data (ExFAT) partition.

Update your AWS DeepRacer device to the Ubuntu 20.04 software stack

Once you create the USB update media as described in the previous steps, you can update your AWS DeepRacer device to the latest software stack including Ubuntu 20.04 Focal Fossa, Intel® OpenVINO™ toolkit 2021.1.110, ROS2 Foxy Fitzroy, and Python 3.8.

Important

Updating to the new AWS DeepRacer software stack will wipe all data on your AWS DeepRacer device.

To update your AWS DeepRacer device software to the Ubuntu 20.04 stack

1. Connect your AWS DeepRacer device to a monitor. You'll need an HDMI-to-HDMI, HDMI-to-DVI, or similar cable. Insert the HDMI end of the cable into the compute module's HDMI port and plug the other end into a compatible port on the monitor.
2. Connect a USB keyboard and mouse. The AWS DeepRacer device's compute module has three USB ports in the front of the vehicle, one either side of and including the port into which the camera is plugged. A fourth USB port is found at the back of the vehicle, in the space between the compute battery and the LED taillight.
3. Insert the USB update media into an available USB port on your compute module. Turn on the power or reset your AWS DeepRacer device and repeatedly press the ESC key to enter the BIOS.

4. From the BIOS window, choose **Boot From File**, then select the option with your boot partition's name, in our case it's named BOOT, then select **<EFI>**, then **<BOOT>**, and finally **BOOTx64.EFI**.
5. After the compute module has booted, a terminal window will appear on the desktop to display the progress. The AWS DeepRacer device will automatically begin the update process after ten seconds. You don't need to provide any input at this stage.

   If an error occurs and the update fails, restart the procedure from Step 1. For detailed error messages, see the `result.log` file generated on the USB drive's data partition.
6. Wait for the update to complete. When the factory reset is complete the terminal window will close automatically.
7. After the device software is updated, disconnect the USB drive from the compute module. You can now reboot or shutdown your AWS DeepRacer device.
8. The AWS DeepRacer device defaults to the following user credentials after update. You will be prompted to change your password on your first login.

   ```
   User: Deepracer
   Password: deepracer
   ```

Build a race track

This section describes how you can build a physical track for an AWS DeepRacer model. To drive your AWS DeepRacer autonomously and to test your reinforcement learning model in a physical environment, you need a physical track. Your track resembles the simulated track used in training and replicates the environment used to train the deployed AWS DeepRacer model.

Topics

Materials and tools

Materials you may need

To build a track, you will need the following materials:

- **For track borders:**

  You can create a track with tape that is about 2-inches wide and white or off-white color against the dark-colored track surface. For a dark surface, use a white or off-white tape. For example, 1.88 inch width, pearl white duct tape or 1.88 inch (less sticky) masking tape.

- **For track surface:**

  You can create a track on a dark-colored hard floor such as hardwood, carpet, concrete, or asphalt felt. The latter mimics the real-world road surface with minimal reflection. Interlocked foam or rubber pads are also good options.

Tools you may need

The following tools are either required or helpful to design and build your track:

- **Tape measure and scissors**

A good tape measure and a pair of scissors are essential for building your track. If you don't already have one, you can order a tape measure here or scissors here.

- **Optional design tools**

  To design your own track, you might need a protractor, a ruler, a pencil, a knife and a compass.

Lay your track

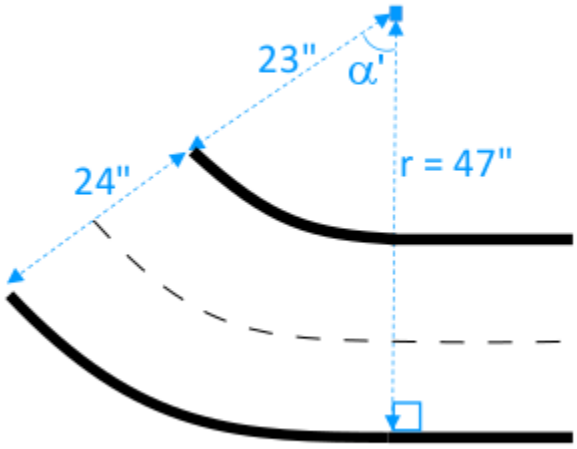
When you build your track, it's a good practice to start with a simple design, such as a straight or single-turn track. Next you can move on to looped tracks. Here, we use a single-turn track as an example to walk you through the steps to construct your own track. First let's review dimensional requirements of a track.

Dimensional Requirements

You can build a track of any shape as long as it meets the following requirements:

- **Minimum turning radius:**

  On a curved track, the turning radius (r) measures from the circle center to the outside border, as illustrated below.



The minimum turning radius (r_{min}) depends on the track turning angle (α) at a corner and should comply to the following limits:

- If the track's turning angle is α ≤ 90 degrees, r_{min} ≥ 25 inches

  We recommend 30 inches.

- If the track's turning angle is α > 90 degrees, r_{min} ≥ 30 inches.

  We recommend 35 inches.

- **Track width**

  The track width (w_{track}) should comply to the following limit:

  w_{track} ≥ 24 ± 3 inches.

- **Track surface:**

  The track surface should be smooth and of a uniform dark color. The minimum enclosing area should be 30 inches x 60 inches in size.

  Carpeted and wood floors work well. Interlocked foam or rubber pads match the simulated environment better than wood, but this is not required. Concrete floors can be problematic due to light reflection on the surface.

- **Track barrier**

  Though not required, we recommend that you encircle the track with uniform-colored barriers that are at least 2.5 feet tall and 2 feet away from the track at all points.

Considerations for model performance

How you build a track can affect the reliability and performance of a trained model. The following are factors you should consider when building your own tracks.

1. Do not place any white objects on or near your track. If necessary, remove any white object from the track or its vicinity. This is because training in the simulated environment assumes that only the track borders are white.
2. Use clean and continuous tape to mark the track borders. Broken or creased track borders can affect the trained model performance.

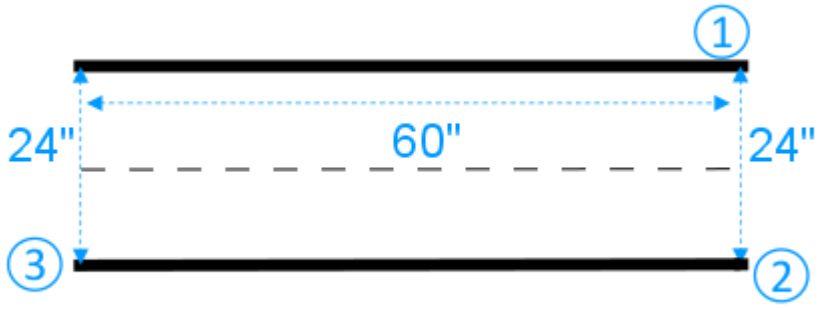
3. Avoid using a reflective surface as the track floor. Reduce glare from bright lights. The glare from straight edges can be misinterpreted as objects or borders.
4. Do not use a track floor with line markings other than the track lines. The model might interpret the non-track lines as part of the track.
5. Place barriers around the track to help reduce distractions from background objects.

Steps to build the track

As an illustration, we use the most basic single-turn track. You can modify the instructions to create a more complex track such as an S-curve, a loop, or the AWS re:invent 2018 track.

To build an AWS DeepRacer single-turn track

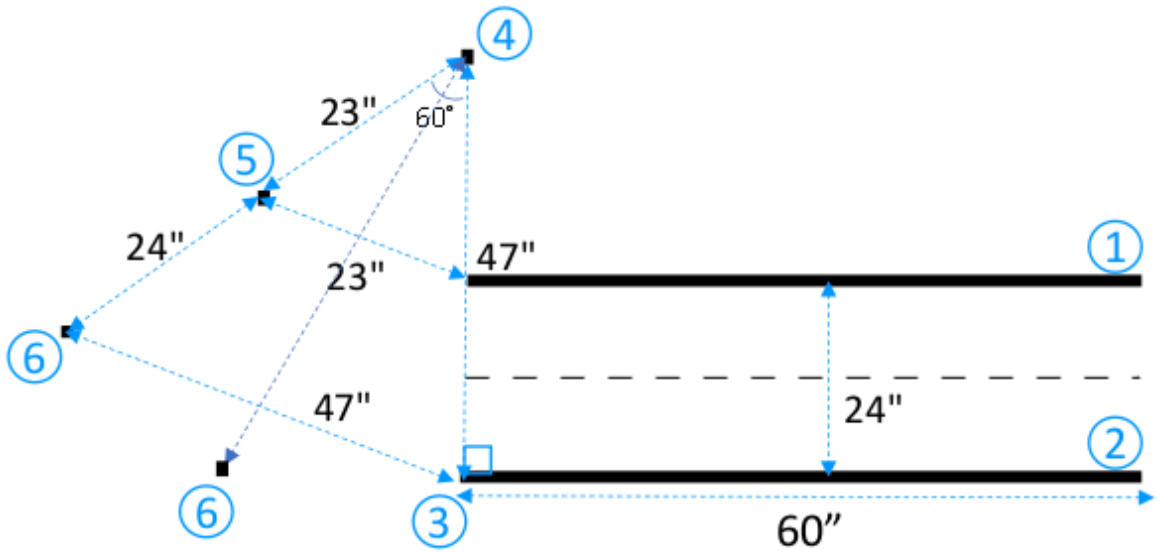
1. To construct the straight portion of the track, follow the steps below and refer to the diagram.
   a. Put a 60-inch long piece of tape on the floor to lay down the first border in a straight line (1).
   b. Use a tape measure to locate the second border's two end points, (2) and (3). Put them 24 inches apart from the first border's two ends.
   c. Put another 60-inch long piece of tape on the floor to lay down the second border to connect the two end points (2) and (3).



   We assume the straight track segment is 60-inches long and 24-inches wide. You can adjust the length and width to fit to your space, provided that the dimensional requirements are met.
2. To make the track to turn at a 60-degree angle, do the following and refer to the diagram:

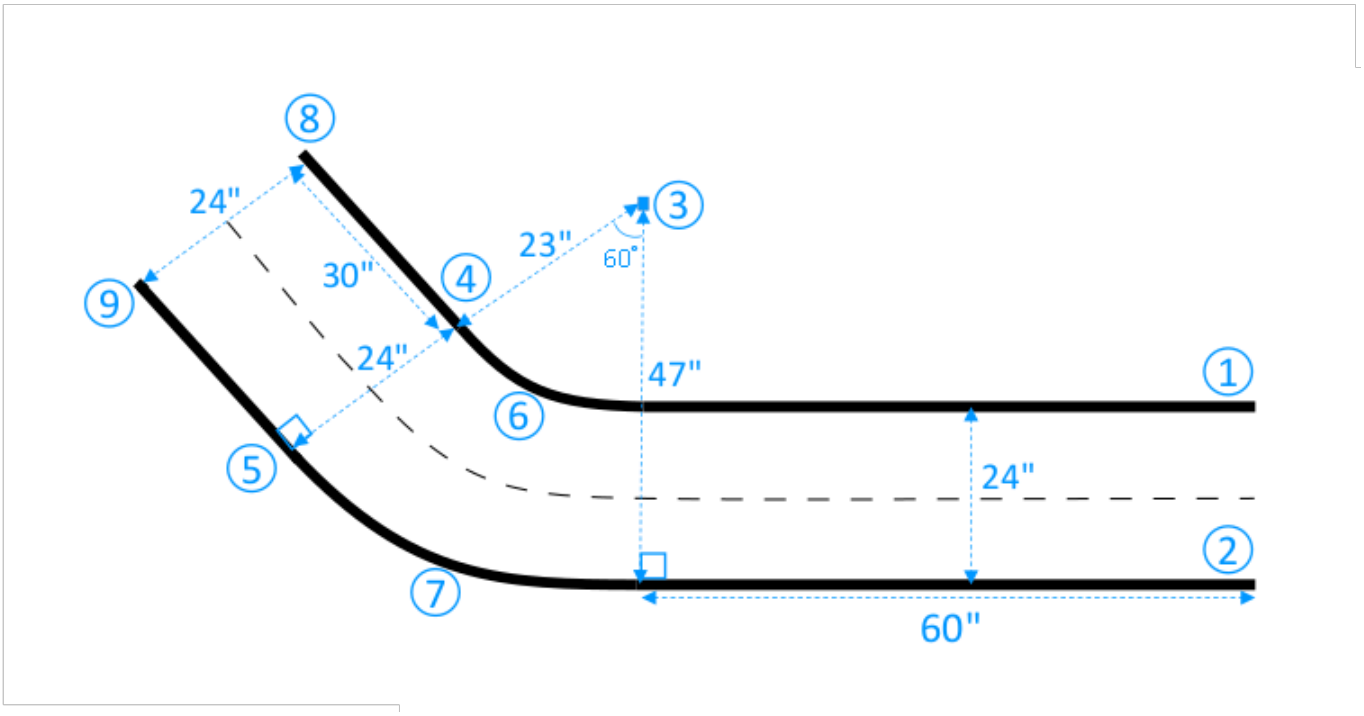
a. Use the tape measure to locate the center (4) of the turning radius (4-3 or 4-6). Mark the center with a piece of tape.

b. Draw an equilateral triangle. The three sides are (3-4), (4-6), and (6-3).



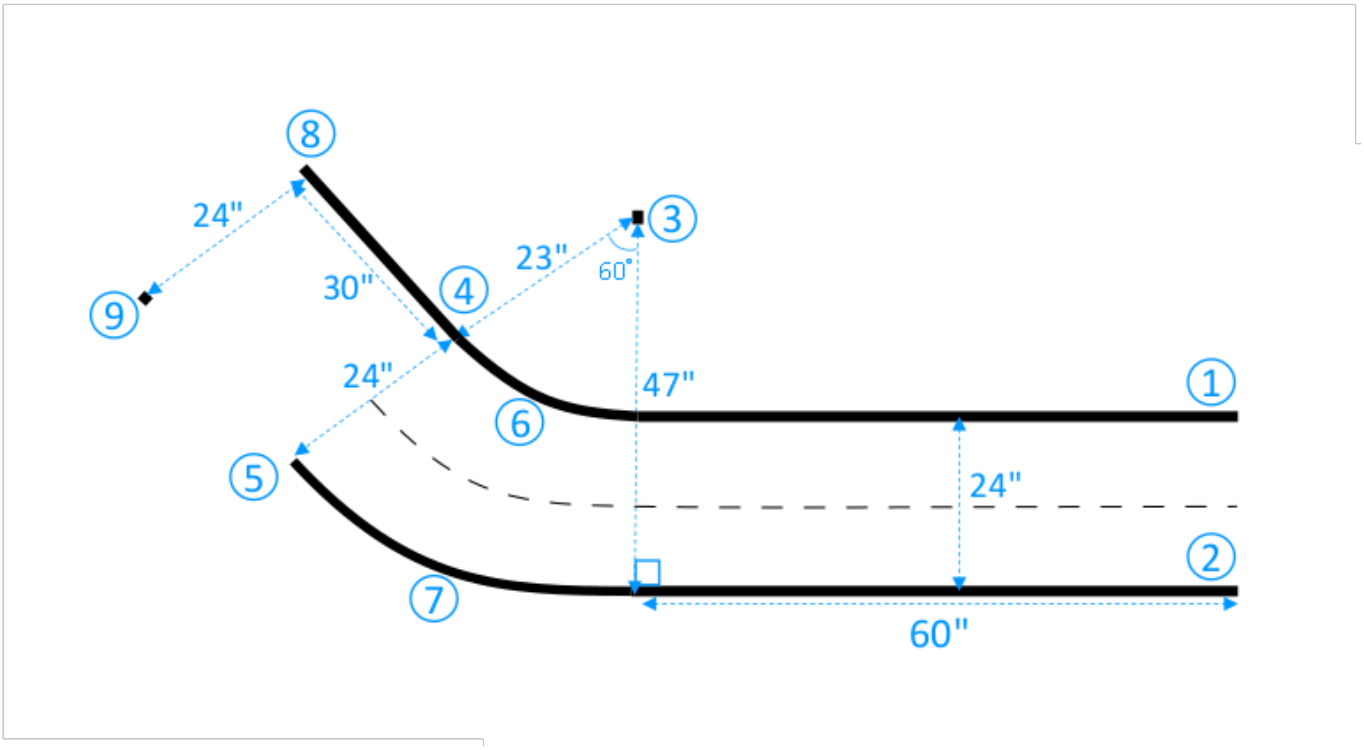
To make a 60-degree turn along the track, use the equilateral triangle (3-4-6) to determine the locations of the two final end points (5) and (6) for the curved track segment. For turns at a different angle, you can use a protractor (or a protractor app) to locate the two final ends (5) and (6) of the curved track segment. Turning radius variations are acceptable as long as the minimum turning radius requirement in Step 2 is met.

c. Put small tape segments, e.g. 4-inches each, on the floor to lay the curved border segments (7) and (8) and connect them with the straight-line borders. The two curved borders don’t need to be parallel.



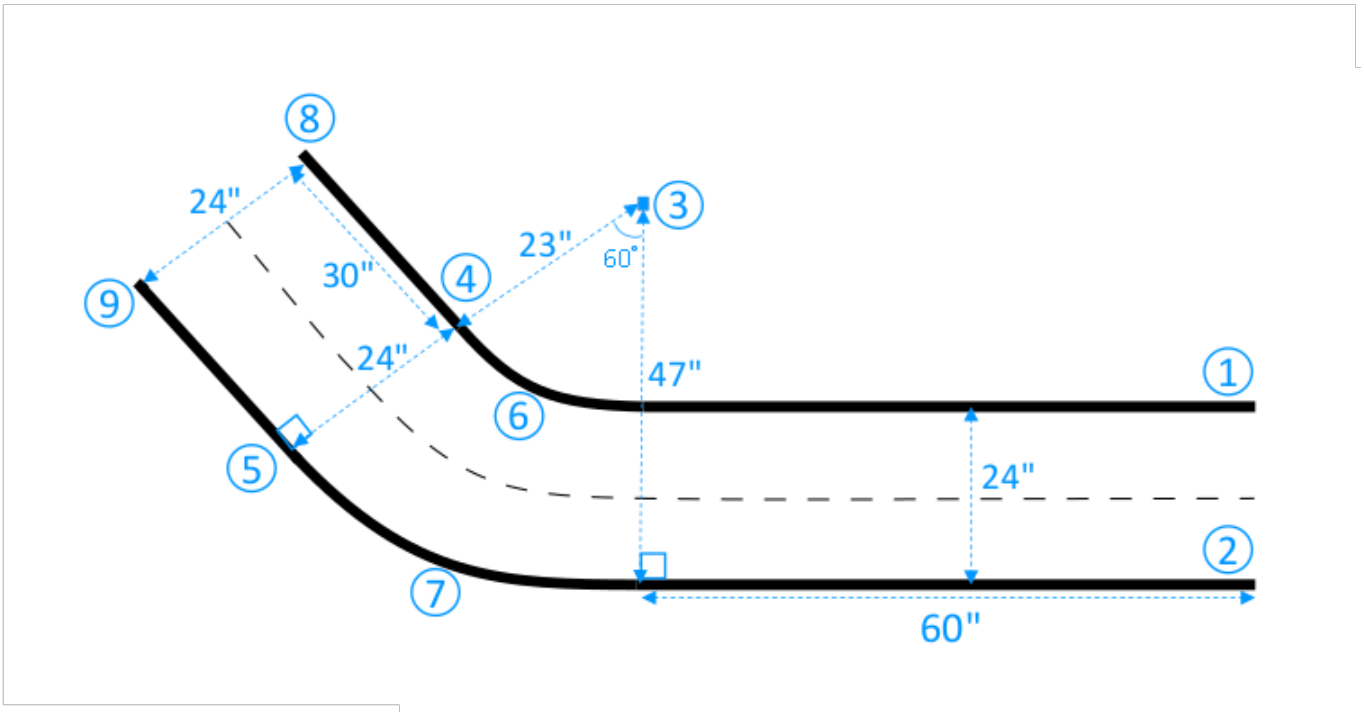
3. To extend the track with the next straight segment of 30 inches long and 24 inches wide, do the following:

   a. Put a 30-inch long piece of tape on the floor to lay down the first border (4-8) perpendicular to the edge (3-5).



b. Use the tape measure to locate the ending point of the second border (9). You can customize the length of the straight lines to fit to the space you have.

c. Put another 30-inch long piece of tape on the floor to lay down the second border (5-9) perpendicular to the edge (3-5).



We assume the second straight track segment is 30 inches long and 24 inches wide. You can adjust the length and width to fit to your space, provided that the dimensional requirements are met and the dimensions are consistent with other track segments.

4. Optionally, cut tape segments of 4 inches long and then place the tape segments 2 inches apart along the track center to lay the dashed center line.

You've now finished building the single-turn track. To help your vehicle to better distinguish the drivable surfaces from non-drivable surfaces, you should paint the off-track surface a color of sufficient contrast with respect to the on-track surface color. To ensure safety, you could encircle the track with uniform-colored barriers that are at least 2.5 feet tall and 2 feet away from the track at all points.

You can apply the instructions to extend the track to more complex shapes.

Track design templates

The following track design templates show AWS DeepRacer tracks that you can build by following the instructions presented in this section.

For all tracks, to reproduce the same color production, use the following specifications:

- Green: PMS 3395 C
- Orange: PMS 137 C
- Black: PMS 432 C
- White: CMYK 0-0-2-9

These tracks were tested with the following materials for their surfaces:

- **Vinyl**

  The tracks were printed on 13-ounce scrim vinyl with a matte finish to reduce glare. Vinyl is typically cheaper than carpet and provides good performance. Vinyl is not as durable as carpet.
- **Carpet**

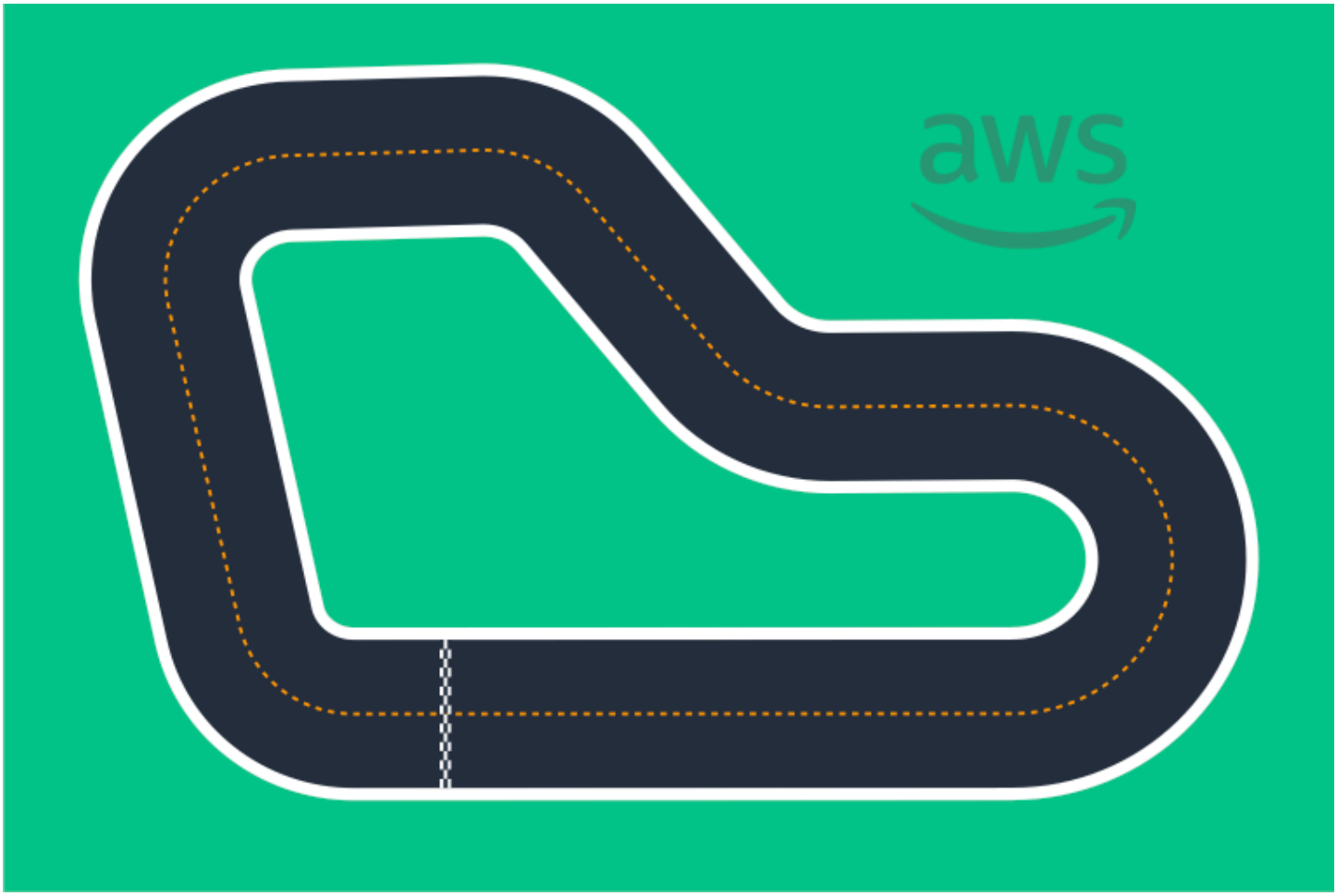
  The tracks were printed on 8-ounce, dye-sublimated, polyester-faced carpet with latex rubberized backing. Carpet is durable and provides great performance, but is expensive.

Due to their large size, the tracks cannot be easily printed on a single piece of material. Align track lines well when connecting pieces together.

Topics

AWS DeepRacer A to Z Speedway (Basic) track template

The AWS DeepRacer A to Z Speedway (Basic) track is the most popular physical competition track in AWS DeepRacer history. It was originally released at AWS re:Invent 2018 and has the smallest footprint of all the AWS DeepRacer physical competition tracks.

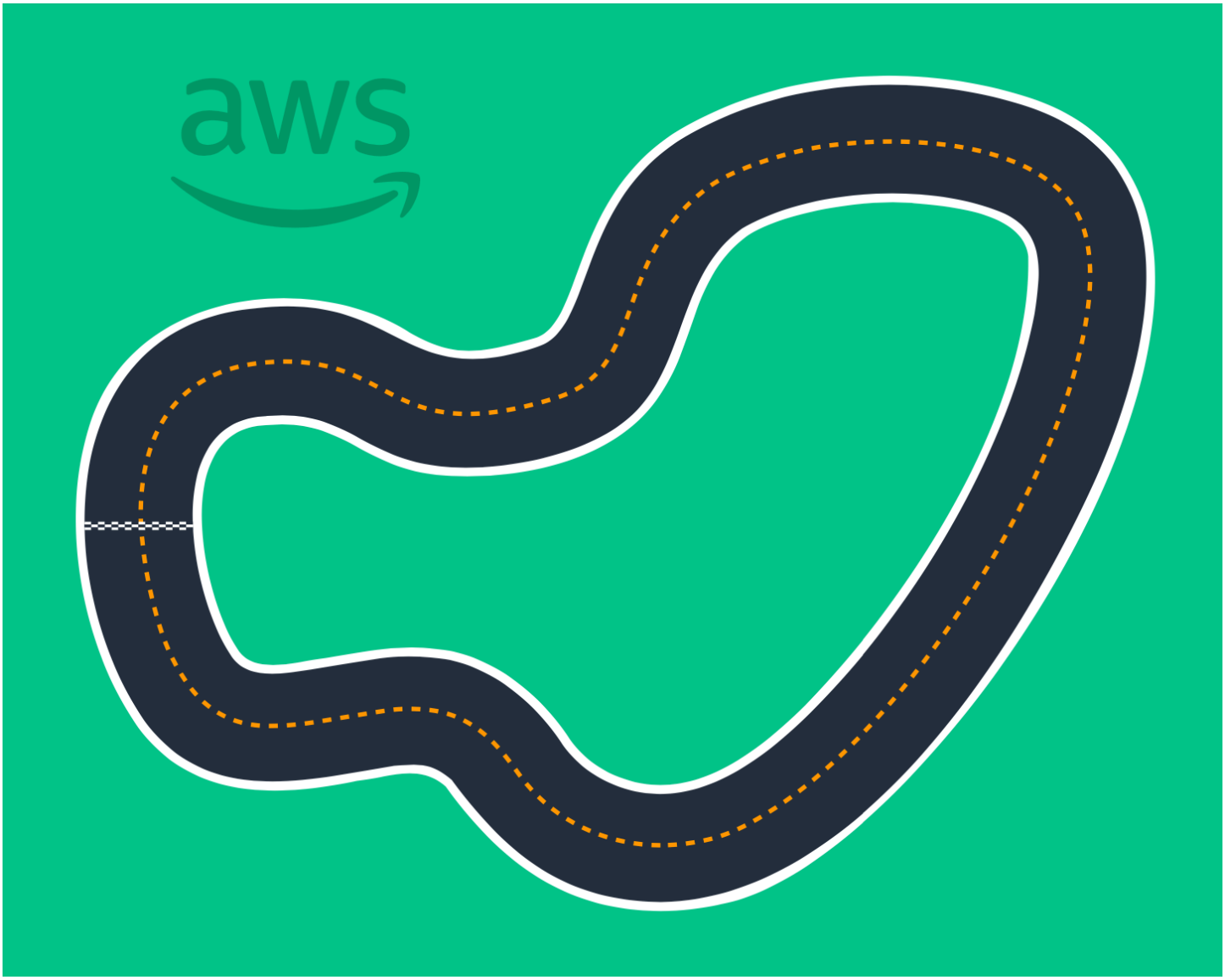


We recommend this track for beginner events and first-time racers. With a variety of runs and straightaways, it offers a compelling challenge for both first-time and experienced racers. The AWS DeepRacer A to Z Speedway (Basic) track is a 1:1 physical reproduction of the virtual track available in the console. It provides racers the opportunity to train a model in a virtual environment and then deploy the model to a physical AWS DeepRacer device for autonomous racing on a physical track.

To print or create your own A to Z Speedway (Basic) track, download the AWS DeepRacer A to Z Speedway (Basic) track file.

AWS DeepRacer Smile Speedway (Intermediate) track template

The AWS DeepRacer Smile Speedway track was originally released as the AWS DeepRacer Championship 2019 track.

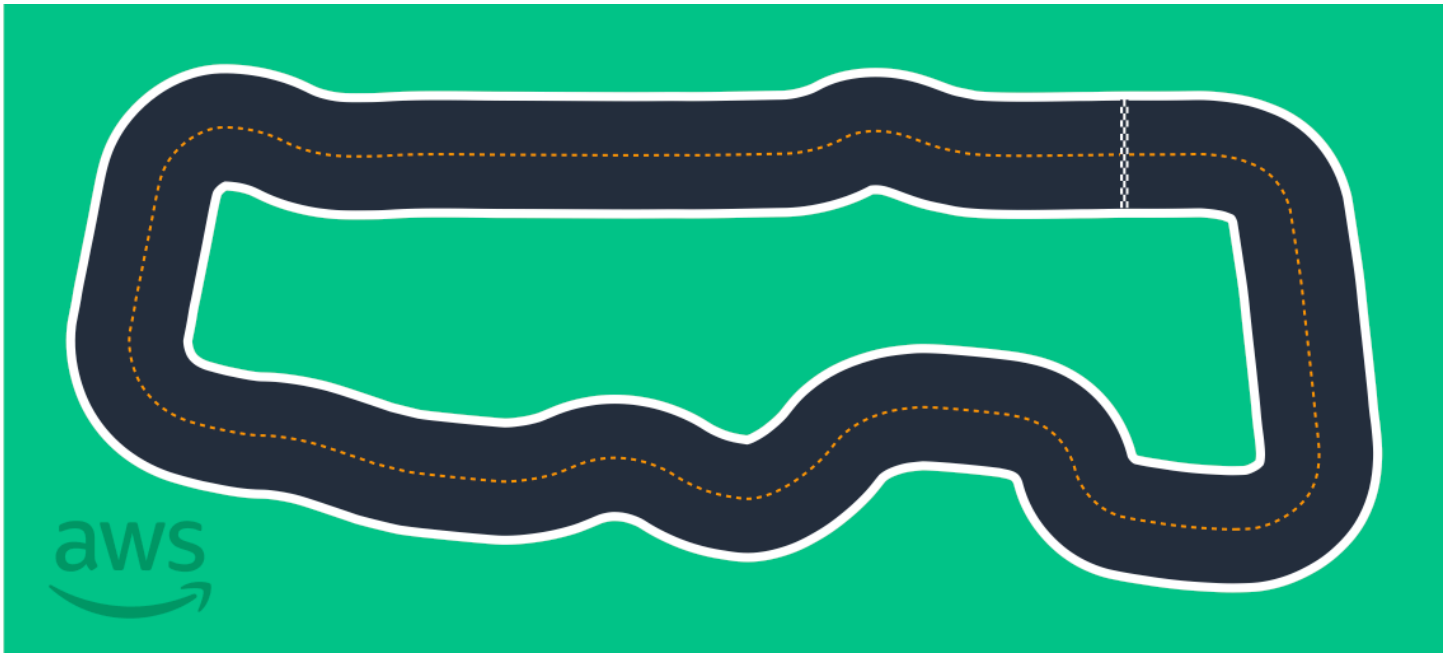


We recommend this intermediate track for events with experienced racers and larger physical spaces. It's a 1:1 physical reproduction of the virtual track available in the console. It provides racers the opportunity to train a model in a virtual environment and then deploy the model to a physical AWS DeepRacer device for autonomous racing on a physical track.

To print or create your own AWS DeepRacer Smile Speedway (Intermediate) track, download the AWS DeepRacer Smile Speedway (Intermediate) track file.

AWS DeepRacer RL Speedway (Advanced) track template

The AWS DeepRacer RL Speedway (Advanced) track (aka AWS DeepRacer Summit Speedway) was originally released for AWS DeepRacer summits in 2022 and is the longest physical track in AWS DeepRacer history.

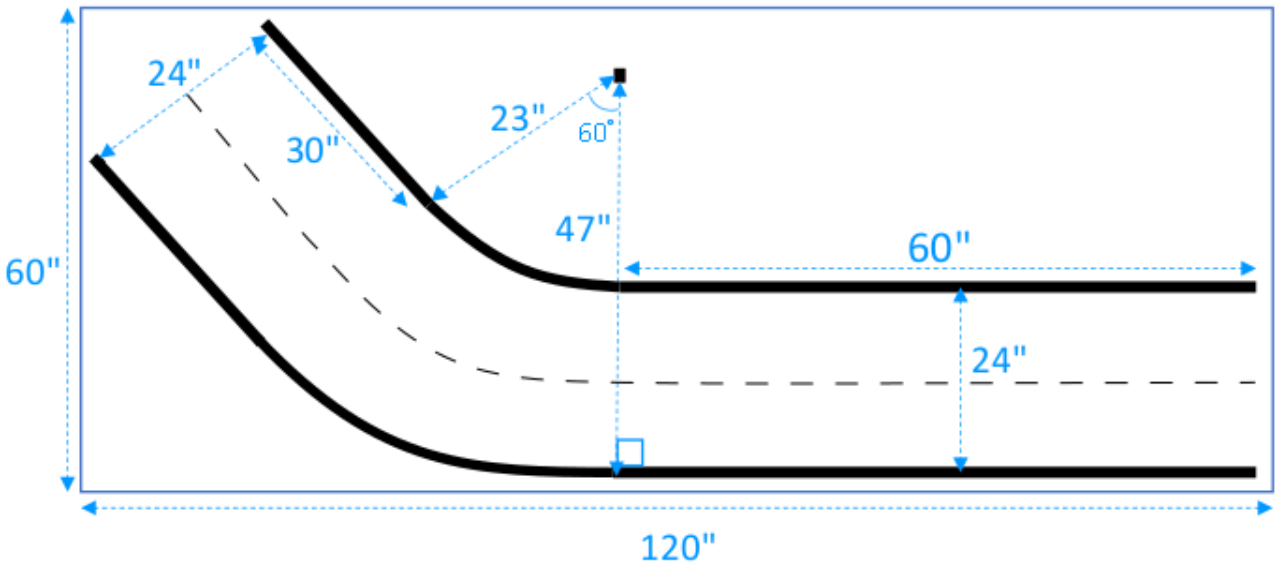


We recommend the AWS DeepRacer RL Speedway (Advanced) track for events with experienced racers. It offers a compelling challenge for racers who enjoy going fast on straightaways. The AWS DeepRacer RL Speedway (Advanced) track is a 1:1 physical reproduction of the virtual track available in the console. It provides the opportunity for racers to train a model in a virtual environment and then deploy the model to a physical AWS DeepRacer device for autonomous racing on a physical track.

To print or create your own AWS RL Speedway (Advanced) track, download the AWS DeepRacer RL Speedway (Advanced) track file.

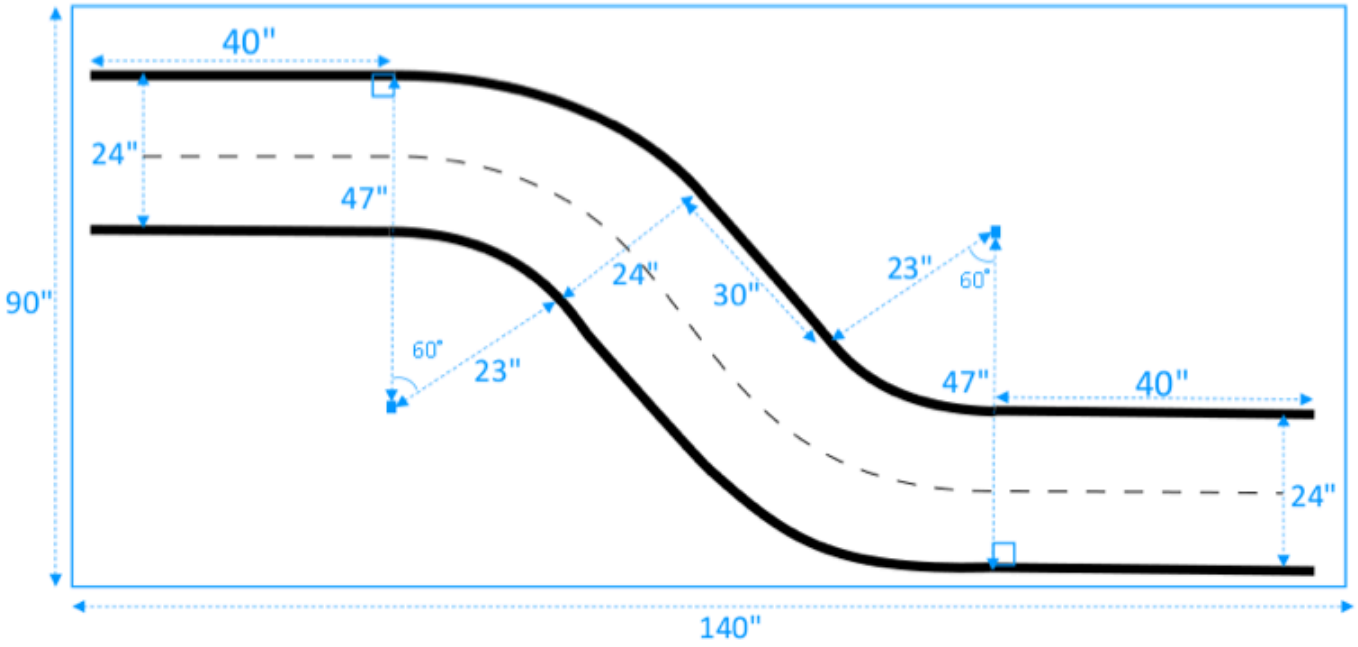
AWS DeepRacer Single-turn track template

This basic track template consists of two straight track segments connected by a curved track segment. Models trained with this track should make your AWS DeepRacer vehicle drive in straight line or make turns in one direction.



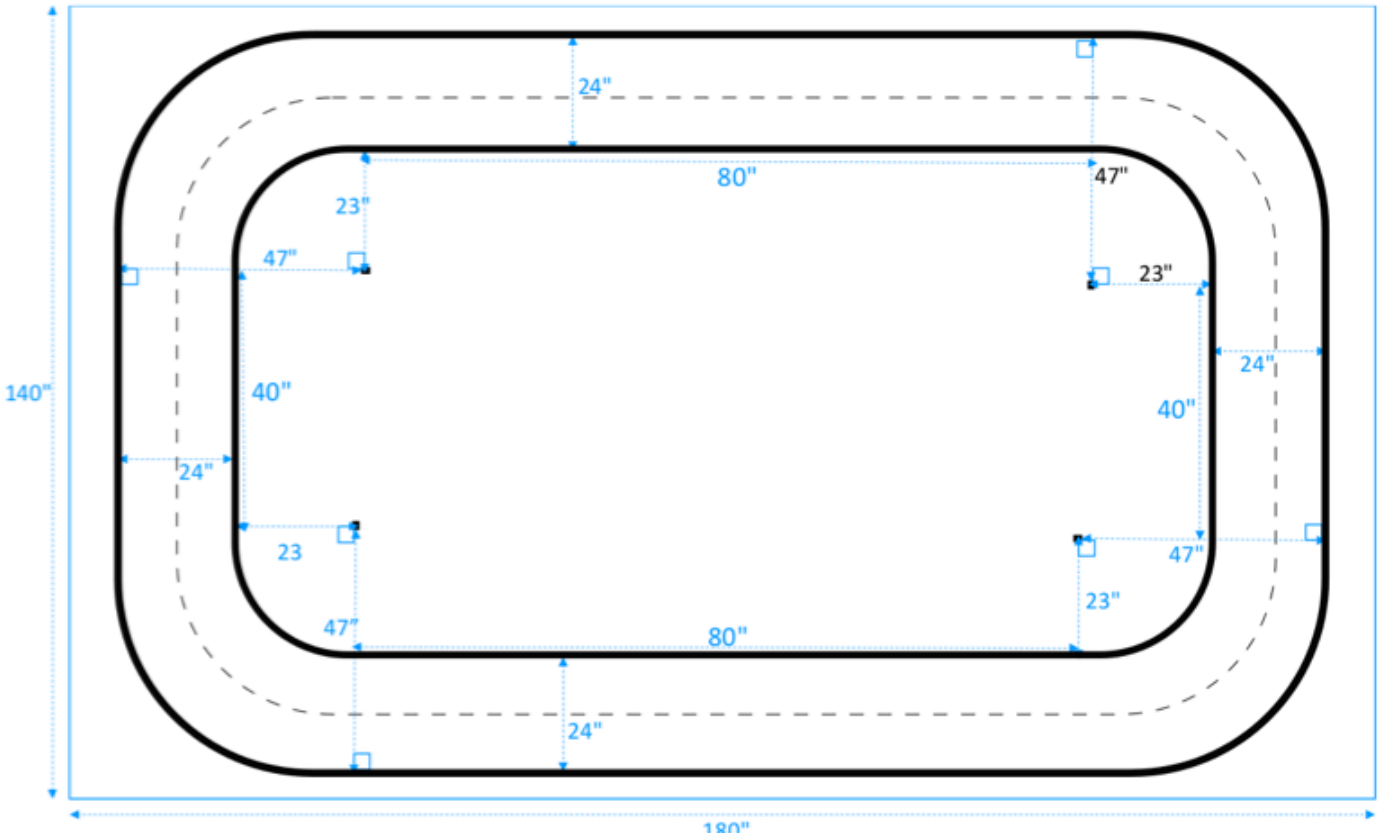
AWS DeepRacer S-curve track template

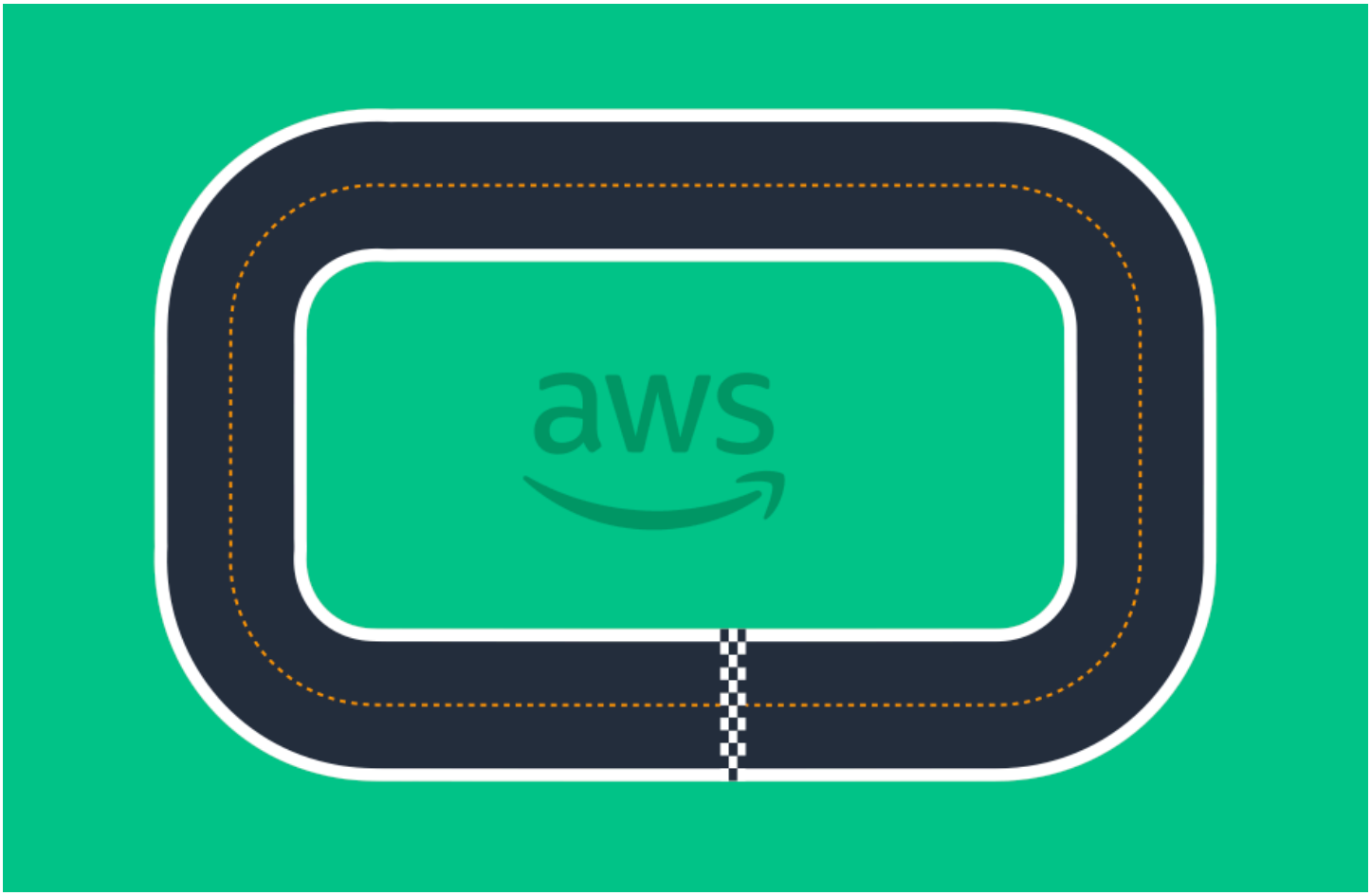
The track is more complex than the single-turn track because the model needs to learn to make turns in two directions. You can easily extend the single-turn track construction instructions to this track by turning it in the opposite direction after the first turn.



AWS DeepRacer Loop track template

This regular loop track is a repeating, 90-degree, single-turn track. It requires a larger enclosing area for laying the entire track.

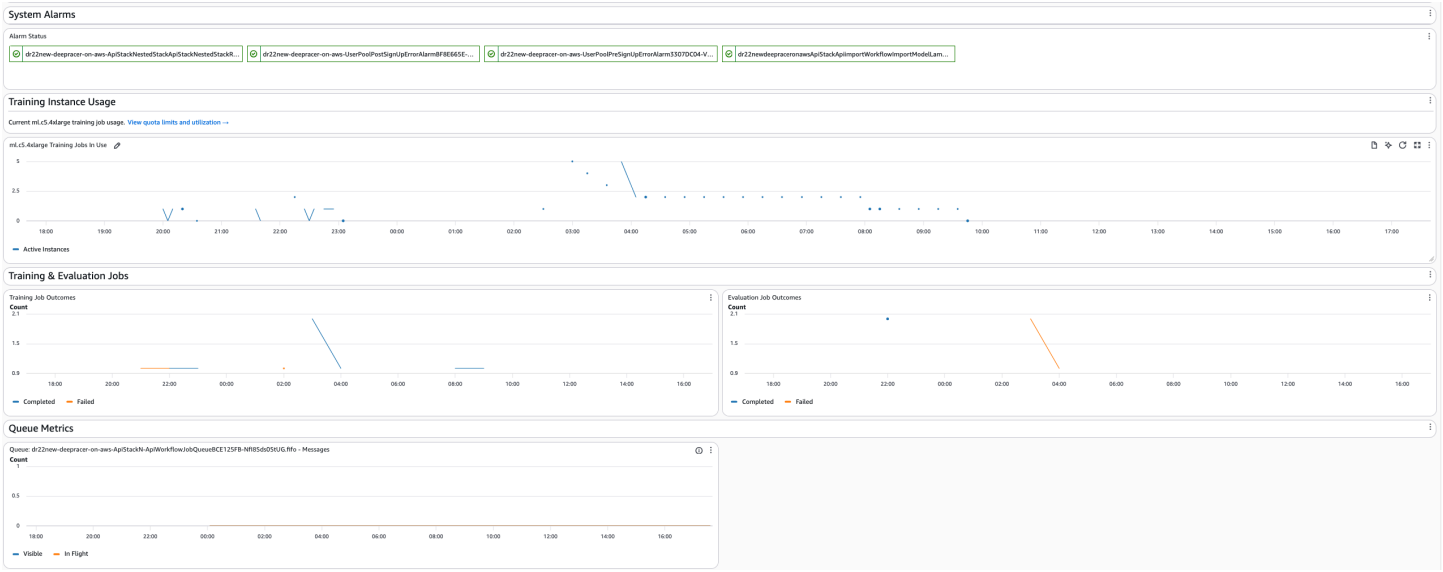

24"
80"
47"
23"
47"
23"
140"
40"
24"
40"
24"
23
47"
23"
47"
80"
24"
180"


aws

Monitoring

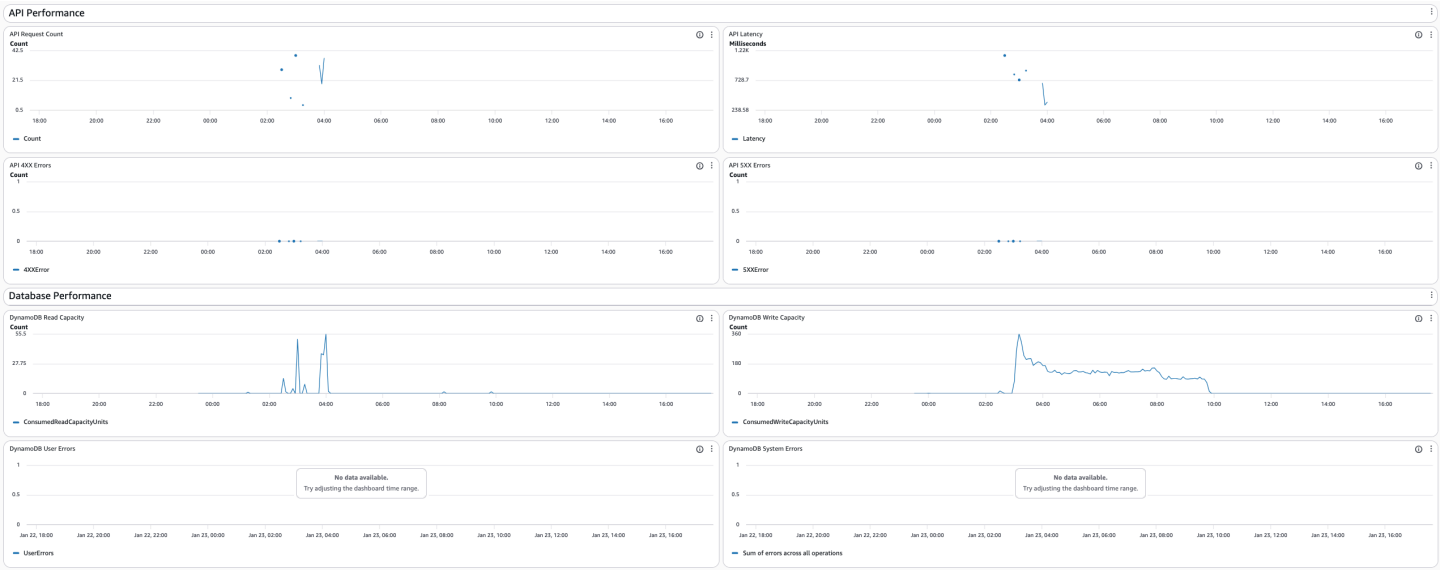
Dashboard

DeepRacer on AWS automatically provisions an Amazon CloudWatch dashboard which surfaces important graphs, metrics, and alarms that are relevant to operating the solution and can help with identifying potential issues. This dashboard can be accessed through the AWS Management Console by going to the **Amazon CloudWatch** console and clicking **Dashboards** in the left sidebar.



The upper-half of this dashboard shows system alarm states (see Alarms), followed by:

- **Training instance usage** metrics, indicating the number of training jobs that are currently in use. This graph is helpful for identifying usage patterns for training and evaluation jobs, and can visually indicate cases such as when your current service quota is being reached.
- **Training job outcomes**, indicating the number of training and evaluation jobs that have completed and/or failed.
- **Queue metrics**, indicating the number of training and evaluation jobs that have been waiting in the queue over time.



The bottom-half of this dashboard shows additional graphs related to:

- **API performance**, indicating the number of requests and latency, as well as the number of 4XX and 5XX errors over time.
- **Database performance**, indicating the number of read and write capacity units being consumed, and the number of user and system errors over time.

Alarms

DeepRacer on AWS automatically provisions CloudWatch alarms to monitor critical system components. These alarms help detect issues early and maintain the health of the deployment.

The deployment includes 9 CloudWatch alarms organized into three monitoring categories:

- **API import workflow monitoring** - 6 alarms that monitor model import processes
- **User authentication monitoring** - 2 alarms that monitor user signup functions
- **Asset processing monitoring** - 1 alarm that monitors asset packaging workflows

API import workflow alarms

These alarms monitor the various stages of importing models into the DeepRacer environment:

Alarm Name	Purpose	Threshold
`ApiimportWorkflowCompletionErrorAlarm`	Monitors completion errors in the API import workflow	≥ 10 errors in 5 minutes
`ApiimportWorkflowRewardValidationErrorAlarm`	Monitors reward validation errors during import processing	≥ 10 errors in 5 minutes
`ApiimportWorkflowDlqProcessorErrorAlarm`	Monitors dead letter queue (DLQ) processor errors	≥ 10 errors in 5 minutes
`ApiimportWorkflowImportAssetsErrorAlarm`	Monitors asset import errors within the workflow	≥ 10 errors in 5 minutes
`ApiimportWorkflowModelValidationErrorAlarm`	Monitors model validation errors during import	≥ 10 errors in 5 minutes
`ApiimportWorkflowImportModelLambdaErrorsAlarm`	Composite alarm for Lambda function errors in the import workflow	Any associated alarm in ALARM state

User authentication alarms

These alarms monitor the Cognito User Pool operations and are more sensitive than workflow alarms:

Alarm Name	Purpose	Threshold
`UserPoolPreSignUpErrorAlarm`	Monitors errors in the pre-signup Lambda trigger function	≥ 1 error in 1 minute

Alarm Name	Purpose	Threshold
`UserPoolPostSignUpErrorAlarm`	Monitors errors in the post-signup Lambda trigger function	≥ 1 error in 1 minute

> **Important**
>
> User authentication alarms have very sensitive thresholds (1 error) and should be prioritized for notification setup.

Asset processing alarms

This alarm monitors the asset packaging workflow:

Alarm Name	Purpose	Threshold
`ApiAssetPackagingDLQAlarm`	Monitors dead letter queue for asset packaging operations	≥ 5 messages visible in 5 minutes

Configuring alarm actions

By default, the solution creates alarms without automated actions. To receive notifications when alarms trigger:

1. Go to the CloudWatch console
2. Select **Alarms** from the left navigation
3. Choose an alarm from the list
4. Select **Actions** → **Edit**
5. Add notification actions such as:
   - SNS topic for email alerts
   - Auto Scaling actions
   - EC2 actions

Viewing alarm status

To check status of an alarm:

1. Open the CloudWatch console
2. Select **Alarms** → **All alarms**
3. Filter by the alarm name prefix `deepracer-on-aws-`
4. Review the **State** column for any alarms in ALARM status

When an alarm enters the ALARM state, investigate the associated service logs and metrics to identify the root cause.

Troubleshooting

This section contains information and procedures for identifying and mitigating issues while operating DeepRacer on AWS. It also contains information on how to contact AWS Support.

Topics

Troubleshooting the solution

Issues with logging in or creating an account

1. *When logging in with correct credentials, I get an error saying that I've provided an incorrect username or password*

   This can be caused by session data and webpage caching in the browser interfering with the normal login behavior. Try opening the console using a private browsing tab and logging in again. If you continue to experience issues, please contact your admin.

2. *When logging in with correct credentials, I'am getting a 403 error*

   This issue can occur when CloudFront or the WAF (Web Application Firewall) is selective about which client requests it allows through and may block legitimate requests that do not meet its access criteria. To resolve this, try turning off your VPN and logging in again.

Issues with creating a model

1. *When I reach the end of the wizard, I get an error that I've exceeded a usage limit*

   This can be caused by your profile either nearing or reaching a usage limit (either compute usage or model storage) that has been set by your admin. The usage limit that is causing this may either be a deployment-wide usage limit (to control usage across the entire deployment), or an individual usage limit (to control usage for individual users). Contact your admin for further

guidance. To prevent this, keep an eye on the usage metric that's on both the home page as well as your profile page.

For admins, find the corresponding user in the Users table and see whether their usage is nearing or has met one or more limits. If not, check whether either of the global limits are approaching their thresholds. If no limits have been set, contact your cloud admin for additional support.

2. *I'm getting an error that my reward function is invalid*

   Reward functions are written in Python and must be valid in order to be trained. This likely means that your reward function code has a syntax error. Use the Validate button to help pinpoint the error in your code.

Issues with viewing the model list

1. *When trying to view the model list, I'am getting a 403 error*

   This issue can occur when CloudFront or the WAF (Web Application Firewall) is selective about which client requests it allows through and may block legitimate requests that do not meet its access criteria. To resolve this, try turning off your VPN and viewing the model list again.

Issues with training or evaluating a model

1. *When I create a model and submit it for training, the model stays in a Queued state for an extended period of time*

   This can be caused by the underlying AWS account reaching its service quota limit for Amazon SageMaker AI training jobs. This can cause training jobs to be delayed, especially if your deployment often experiences high traffic volumes or burst traffic. *For cloud admins:* Try requesting a service limit increase through the Service Quotas console by selecting Amazon SageMaker and requesting an increase for the *ml.c5.4xlarge for training job usage* quota.

2. *When I create a model and submit it for training, the model enters an Error state*

   This means that an error occurred while attempting to train your model, contact your admin for further assistance.

For cloud admins, try examining the `DeepRacerWorkflow` log group that is associated with your deployment in the Amazon CloudWatch Logs console.

3. *How do I download the logs from my training or evaluation job?*
   - For **training jobs**, you can download the logs by clicking the **Download logs** button at the top of the **Training** section on the model detail page.
   - For **evaluation jobs**, you can download the logs by clicking the **Download logs** button at the top of the **Evaluation details** section on the model detail page, after selecting a model from the **Evaluations selector** and clicking **Load evaluation**.

Issues with participating in a race

1. *After my model finishes a race, I don't see it appear in the leaderboard*

   This means that your model has not met the qualification requirements for the race. Keep training and evaluating your model as mentioned in the guide to improve performance.

Issues with importing or exporting a model

1. *I'm having trouble opening a model that I've exported and downloaded to my Windows computer*

   If you are using a Windows computer, you'll need to download and install a file extractor such as 7-Zip in order to extract the model once it has finished downloading to your computer.

Using CloudWatch Logs to diagnose issues

DeepRacer on AWS provisions a series of log groups to which AWS Lambda functions and other services configured by the solution emit logs. This section contains an inventory of those log groups and what they can be used for.

/aws/lambda/DeepRacerApis

This log group contains the log output from all API handlers throughout the application. It is a useful first-stop for root-causing any form of failed request in the application.

/aws/lambda/DeepRacerEcrImages

This log group contains the log output from all functions that are responsible for downloading and hosting the images that power the solution's simulation capabilities. It can be useful for root-causing persistent errors that are returned from failed training or evaluation jobs.

/aws/lambda/DeepRacerIndy-RewardFunctionValidationFn

This log group contains the log output from the reward function validator. This log group can be helpful in identifying why validation on a given reward function, or multiple reward functions, may have failed.

/aws/lambda/DeepRacerScheduled

This log group contains the log output from all functions that are responsible for managing scheduled operations, such as resetting usage metrics at the turn of every month. It can be useful for diagnosing issues with usage metrics.

/aws/lambda/DeepRacerSystemEvents

This log group contains the log output from all functions that are responsible for managing resource utilization, specifically model storage; as well as other cloud admin functions such as updating the CORS policy, updating web assets during a stack update, and other related functions.

/aws/lambda/DeepRacerUserIdentity

This log group contains the log output from all functions that handle user authentication and management. The information in this log group can be useful for root-causing issues with inviting a user, initial account creation, user group assignment, and related processes.

/aws/lambda/DeepRacerWorkflow

This log group contains the log output from the functions that are responsible for dispatching, initializing, monitoring, and finalizing training and evaluation jobs. It can be useful for diagnosing any errors that are thrown while a model is being trained or evaluated, or going through the various state changes expected as part of this process.

Using CloudWatch Logs Insights queries to diagnose issues

DeepRacer on AWS comes with a set of pre-configured sample queries that can be used either as-is or to create a custom query for logs in CloudWatch log groups. To access these:

1. Go to **Logs Insights** in Amazon CloudWatch, and select the log group(s) you would like to query.

2. Click **Saved and sample queries**, which will display a sidebar on the right-hand side of the screen with **DeepRacerOnAWS Sample Queries**, which can be expanded. Clicking one of the options will populate the sample log query in the query body.
3. When ready, click **Run query**.

Example: The following CloudWatch Logs Insights query filters logs for error messages or exceptions, then displays the 200 most recent matching entries with their timestamps sorted newest first. This can help you get started with visualizing, analyzing, and localizing recent errors emitted by the system.

```
filter @message like /(?i)(Exception|error)/| fields @timestamp, @message | sort
 @timestamp desc | limit 200
```

Using point-in-time recovery to restore data from a backup

DeepRacer on AWS uses an Amazon DynamoDB table that is configured with point-in-time recovery and continuous backups enabled by default. In the event that data in the table is accidentally deleted, lost, or otherwise corrupted, you can use the service's backup recovery features to restore normal operation. See Restoring a DynamoDB table from a backup for directions on how to restore your table to a certain point using the console or AWS CLI.

Troubleshooting the vehicle

How to resolve issues connecting your computer directly to the vehicle using a USB cable

When setting up your vehicle for the first time, you might find it unable to open the device console (also known as the device web server, `https://deepracer.aws`, hosted on the vehicle) after connecting your AWS DeepRacer vehicle to your computer with a micro-USB/USB cable (USB is also referred to as USB-A).

Multiple causes may be behind this. Typically, you can resolve the issue with the following simple remedy. To activate your device's USB-over-Ethernet network

- Turn off Wi-Fi on your computer and unplug any Ethernet cable connected to it.
- Press the **RESET** button on the vehicle to reboot the device.

- Open the device console by navigating to `https://deepracer.aws` from a web browser on your computer.

If the previous procedure doesn't work, you can check your computer's network preferences to verify that they're properly configured to let the computer connect to the device's network, whose network name is `Deepracer`. To do this, follow the steps in the following procedure.

> **Note**
>
> The instructions below assume you're working with a MacOS computer. For other computer systems, consult with the network preferences documentation for the respective operating system and use the below instructions as a general guide.

To activate the device's USB-over-ethernet network on your MacOS computer

1. Choose the network icon (on the top-right corner of the display) to open **Network preferences**. Alternatively, choose **Command+space**, type **Network**, and then choose **Network System Preferences**.
2. Check if **Deepracer** is listed as **Connected**. If **DeepRacer** is listed but not connected, make sure the micro-USB/USB cable is tightly plugged in between the vehicle and your computer.
3. If the **Deepracer** network is not listed there or is listed but not connected when the USB cable is plugged in, choose **Automatic** from the **Location** preference and then choose **Apply**.
4. Verify that the AWS DeepRacer network is up and running as **Connected**.
5. When your computer is connected to the **Deepracer** network, refresh the `https://deepracer.aws` page on the browser, and continue with the rest of **Get Started Guide** instructions of **Connect to Wi-Fi**.
6. If the **Deepracer** network is not connected, disconnect your computer from the AWS DeepRacer vehicle and then reconnect it. When the **Deepracer** network becomes **Connected**, continue with the **Get Started Guide**.
7. If the **Deepracer** network on the device is still not connected, reboot your computer and AWS DeepRacer vehicle and repeat from **Step 1** of this procedure, if necessary.

If the above remedy still doesn't resolve the issue, the device certificate might have been corrupted. Follow the steps below to generate a new certificate for your AWS DeepRacer vehicle to repair the corrupted file.

To generate a new certificate on the AWS DeepRacer vehicle

1. Terminate the USB connection between your computer and your AWS DeepRacer vehicle by unplugging the micro-USB/USB cable.
2. Connect your AWS DeepRacer vehicle to a monitor (with a HDMI-to-HDMI cable) and to USB keyboard and mouse.
3. Log in to the AWS DeepRacer operating system. If this is the first login to the device operating system, use `deepracer` for the password, when asked for, and then proceed to change the password, as required, and use the updated password for subsequent logins.
4. Open a terminal window and type the following Shell command. You can choose the **Terminal** shortcut from **Applications** → **System Tools** on the desktop to open a terminal window. Or you can use the file browser, navigate to the `/usr/bin` folder, and choose **gnome-terminal** to open it.

   ```
   sudo /opt/aws/deepracer/nginx/nginx_install_certs.sh && sudo reboot
   ```

   Enter the password, which you used or updated in the previous step, when prompted.

   The above command installs a new certificate and reboots the device. It also reverts the device console's password to the default value printed at the bottom of the AWS DeepRacer vehicle.
5. Disconnect the monitor, keyboard and mouse from the vehicle and reconnect it to your computer with the micro-USB/USB cable.
6. Follow the second procedure in this topic to verify your computer is indeed connected to the device network before opening the device console (https://deepracer.aws) again and, then, continue with the **Connect to Wi-Fi** instructions in **Get Started Guide**.

How to switch compute module power from battery to power outlet

If the compute module battery level is low when you set up your AWS DeepRacer for the first time, follow the steps below to switch the compute power supply from the battery to a power outlet:

1. Unplug the USB-C cable from the vehicle's compute power port.

2. Attach the AC power cord and the USB-C cable to the computer module power adapter. Plug the power cord to a power outlet and plug the USB-C cable the vehicle's computer module power port.

How to connect your vehicle to a Wi-Fi network using a flash drive

To connect an AWS DeepRacer vehicle to your home or office Wi-Fi network using a USB flash drive, you need the following:

- A USB flash drive
- The name (SSID) and password for the Wi-Fi network that you want to join

> **Note**
>
> AWS DeepRacer does not support Wi-Fi networks that require active captcha verification for user sign-in.

To connect an AWS DeepRacer vehicle to a Wi-Fi network using a USB flash drive

1. Plug the USB flash drive into your computer.
2. Open a web browser on your computer and navigate to https://aws.amazon.com/deepracer/usbwifi. This link opens a text file named wifi-creds.txt hosted on GitHub.
3. Save `wifi-creds.txt` to your USB flash drive. Depending on which web browser you use, the text file might download to your computer and open in your default code editor automatically. If `wifi-creds.txt` doesn't download automatically, open the context (right-click) menu and choose **Save as** to save the text file to your USB flash drive. **Warning**: Do not change the file name.
4. If `wifi-creds.txt` isn't already open, open it in a code editor in plain text mode. Some text editors default to rich text (.rtf) instead of plain text (.txt) when the file type isn't specified, so if you are having trouble editing the file, check your settings. If you are using Windows, you can also try to open the file using the Sublime Text application, which you can download for free, or, if you use a Mac, try the TextEdit application, which is pre-installed on most Mac devices and defaults to plain text.

5. In between the single quotation marks at the bottom of the file, enter the name (SSID) and password of the Wi-Fi network that you want to use. SSID stands for "Service Set Identifier." It is the technical term for the name of your Wi-Fi network.
   - If the network name (SSID) or password contains a space, such as in *Your-Wi-Fi 100*, enter the name exactly, including the space, inside the quotation marks ('). If there is no space, using quotation marks is optional. For example, the Wi-Fi password, *Passwd1234* doesn't contain a space, so using single quotation marks works but isn't necessary. Both SSID and password are case sensitive.
6. Save the file on your USB flash drive.
7. Eject the USB drive from your computer and plug it into the USB-A port on the back of the AWS DeepRacer vehicle between the compute battery power button and the rear stanchion.
8. Ensure that the AWS DeepRacer is powered on.
9. Watch the Wi-Fi LED on the vehicle. If it blinks and then changes from white to blue, the vehicle is connected to the Wi-Fi network. Unplug the USB drive and skip to step 11.
   - If the USB drive was plugged into the vehicle before you attempted to connect the vehicle to a Wi-Fi network, a list of available Wi-Fi networks will be automatically displayed in `wifi-creds.txt` file on your flash drive. Uncomment the one that you want to connect to by removing the pound sign.
10. If the Wi-Fi LED turns red after blinking, unplug the USB drive from the vehicle and plug it back into your computer. Check the Wi-Fi name and password that you entered in the text file for typos, errors in spacing, incorrect sentence casing, or missing or misused single quotation marks. Correct mistakes, and re-save the file, and repeat Steps 7-9.
11. After the vehicle Wi-Fi LED turns blue, unplug the USB drive from the vehicle and plug it into your computer.
12. Open the `wifi-creds.txt` file. Find your vehicle's IP address at the bottom of the text file and copy it.
13. Make sure your computer is in the same network as the vehicle, then paste the IP address into your web browser.
    - If you are using macOS Catalina, use the Firefox web browser. Chrome is not supported.
14. When prompted with a message that the connection is not private or secure, accept the security warning and proceed to the host page.

Your AWS DeepRacer is now connected to Wi-Fi.

How to charge the drive module battery

The AWS DeepRacer drive module battery has two sets of cables with two different color JST connectors, white and red. The white 3-pin connector, at the end of the black, red, and white cables, connects the vehicle module battery to its battery charger. The red 2-pin connector, at the end of the black and red cables, connects the battery to the vehicle drive train.

Follow the steps below to charge your AWS DeepRacer drive module battery:

1. To access the drive module battery if it is connected to the vehicle, lift the compute module, being careful not to loosen the wires connecting it to the drive train.
2. Optionally, to remove the drive module battery from the vehicle, disconnect the red 2-pin battery connector from the black and red drive train connector and unstrap the Velcro strap.
3. Attach the battery to the battery charger by connecting the battery's white 3-pin connector to the charger port.
   - *Red light + green light = not fully charged*
4. Plug the power cord of the battery charger into a power outlet. When only the green light is illuminated, your battery is fully charged.
5. Disconnect the charged vehicle battery's white 3-pin connector from the charge adapter. If you removed the battery to charge it (optional) make sure to reconnect its red 2-pin connector to the vehicle drive train connector and secure the battery to the vehicle with the Velcro strap.
6. Turn on the vehicle drive train by pushing its switch to the "on" position. Listen for the indicator sound (two short beeps) to confirm a successful charge. If you don't hear two beeps, try unlocking your vehicle battery.

How to charge the compute module battery

Follow the steps below to charge your AWS DeepRacer compute module battery:

1. Optionally remove the compute module battery from the vehicle.
2. Attach the compute power charger to the compute module battery.
3. Plug the power cord of the compute battery charger into a power outlet.

My battery is charged but my AWS DeepRacer vehicle doesn't move

Follow these steps if your AWS DeepRacer console is set up, your compute battery is charged, and your Wi-Fi is connected, but your vehicle still doesn't move:

1. Lift the compute module, being careful not to loosen the wires connecting it to the drive train. Make sure the vehicle battery underneath is correctly connected, red 2-pin connector to black and red drive train connector.
2. Turn on the vehicle drive train by pushing the switch to the "on" position. Listen for the indicator sound (two short beeps) to confirm that the vehicle has charge. If the vehicle powers on successfully, skip to step 4.
3. If you do not hear two beeps when you switch on your vehicle battery, ensure that the battery is fully charged. Plug the vehicle battery's white connector cable into its charge adapter, which can be differentiated from the compute module's adapter by its red and green LED indicator lights. Connect the adapter to its charge cable and plug it into a power outlet. When both red and green lights on the vehicle battery charge adapter are lit, it indicates that the battery still needs charging.

   *Red light + green light = **not** fully charged*

   When only the green light is illuminated, your battery is fully charged and ready to use. Disconnect the car battery's white connector from the charge adapter, and reconnect its red connector to the vehicle. If you removed the battery to charge it (optional) make sure to once again secure it to the drive train with the Velcro strap. Turn on the vehicle drive train by pushing its switch to the "on" position. If you still don't hear two beeps, try unlocking your vehicle battery.
4. Connect your vehicle to Wi-Fi and open the AWS DeepRacer console in your browser. Manually drive your vehicle with the touch joystick to confirm that it can move.

> **Note**
>
> To get the most mileage out of your vehicle battery, make sure to switch off the vehicle drive train or disconnect its battery when you are not using your AWS DeepRacer.

How to resolve a battery lockout

Important

This battery is only for use with the DeepRacer Car. This battery must be handled properly to avoid risk of fire, explosion, or other safety concerns. Follow all instructions and heed all warnings included in the AWS DeepRacer Device Safety Guide.

AWS DeepRacer Device Terms, Warranties, and Notices

- AWS DeepRacer Device Terms of Use
- One-Year Limited Warranty for AWS DeepRacer Device
- AWS DeepRacer Device Safety Guide

Preventing a battery lockout

To preserve battery health, the AWS DeepRacer vehicle battery goes into lockout state. When this happens, the battery won't power your vehicle even if it's still partially charged. To prevent your car battery from entering lockout state, do the following:

- When you finish using your AWS DeepRacer, turn off the vehicle to preserve the battery's charge.
- When the device console alerts you that your vehicle battery's power level is low, charge it as soon as possible.
- When you think you won't use AWS DeepRacer for a while, disconnect the battery from the vehicle and fully charge it. We suggest you charge your vehicle battery at least once a year to protect it and prevent lockout.

Note

All lithium polymer (LiPo) batteries slowly discharge over time, even when not in use.

Unlocking a battery after lockout

To unlock your vehicle battery after lockout, use the unlock cable, which is a short cable that has a white, 3-pin connector on one end, and a red, 2-pin connector on the other end.

1. Insert battery connectors into the matching colored cable connectors, red to red and white to white.
2. Disconnect the battery from the cable.
3. Your AWS DeepRacer vehicle battery is immediately ready for use. Reconnect its red 2-pin connector to the vehicle drive train connector and secure the battery to the vehicle with the Velcro strap.
4. Turn on the vehicle drive train by pushing its switch to the "on" position. Listen for the indicator sound (two short beeps) to confirm that the battery has been successfully unlocked.

How to wrap compute module battery connector cable when installing LiDAR sensor

Fitting the Evo shell over a LiDAR sensor connected to an AWS DeepRacer vehicle using the extra-long Dell USB-C to angle USB-C connector cable requires a specific cable wrapping technique.

To watch a video of this process, see AWS DeepRacer: Install LiDAR Sensor and wrap Dell compute battery connector cable on YouTube. The video starts with the installation of the LiDAR sensor on the AWS DeepRacer vehicle. The Dell battery wrapping technique begins at 00:01:27 seconds.

> **Note**
>
> The Dell compute battery connector cable has a barrel, a standard USB-C end, and an angle USB-C end.

To wrap a Dell battery cable around a LiDAR sensor to accommodate the Evo shell

1. Facing the rear of the AWS DeepRacer vehicle, plug the angle end of the compute battery connector cable into the compute battery USB-C port with the connector cable pointing to the left.
2. Turning the vehicle slightly to the left, find the opening to the space in between the LiDAR holder and the compute battery just below the rear stanchions and lace the cable through. Stop pulling the cable through when the barrel is inserted into this space. There should be a loop of slack cable to the left of the USB-C port.
3. Facing the rear of the AWS DeepRacer vehicle, wrap the cable counterclockwise around the base of the LiDAR sensor, using the cable clips to secure the cable to itself to ensure a snug fit.

4. Turn the vehicle slightly to the right and plug the standard USB-C end of the cable into the USB-C port.
5. Place the Evo shell on your AWS DeepRacer vehicle and fasten it with pins to test the fit. When the shell fits correctly, the LiDAR sensor is fully visible through the cutout in the shell, and you have access to the pin holes on the top of the stanchions. Remove the shell and adjust your cable as necessary.

Your LiDAR sensor should now be connected. You are ready to turn on your vehicle, drive, and experiment.

How to maintain your vehicle's Wi-Fi connection

How to troubleshoot Wi-Fi connection if your vehicle's Wi-Fi LED indicator flashes blue, then turns red for two seconds, and finally off

Check the following to verify you have the valid Wi-Fi connection settings.

- Verify that the USB drive has only one disk partition with only one `wifi-creds.txt` file on it. If multiple `wifi-creds.txt` files are found, all of them will be processed in the order they were found, which may lead to unpredictable behavior.
- Verify the Wi-Fi network's SSID and password are correctly specified in the `wifi-creds.txt` file. An example of this file is shown as follows:

```
##################################################################################
#                                    AWS DeepRacer                               #
# File name: wifi-creds.txt                                                      #
#                                                                                #
# ...                                                                            #
##################################################################################

# Provide your SSID and password below
ssid: ' MyHomeWi-Fi'
password: myWiFiPassword
```

- Verify both the field names of `ssid` and `password` in the `wifi-creds.txt` file are in lower case.
- Verify that each of the field name and value is separated by one colon (:). For example. `ssid : ' MyHomeWi-Fi'`

- Verify that the field value containing a space is enclosed by a pair of single quotes. On Mac, TextEdit or some other text editor shows single quotes as of the '…' form, but not of ‘…’. If the field value does not contain spaces, the value can be without single quotes.

What does it mean when the vehicle’s Wi-Fi or power LED indicator flashes blue?

If the USB drive contains the `wifi-creds.txt` file, the Wi-Fi LED indicator flashes blue while the vehicle is attempting to connect to the Wi-Fi network specified in the file.

If the USB drive has the `models` directory, the Power LED flashes blue while the vehicle is attempting to load the model files inside the directory.

If the USB drive has both the `wifi-creds.txt` file and the `models` directory, the vehicle will process the two sequentially, starting with an attempt to connect to Wi-Fi and then loading models.

The Wi-Fi LED might also turn red for two seconds if the Wi-Fi connection attempt fails.

How can I connect to the vehicle’s device console using its hostname?

When connecting to the vehicle’s device console using its hostname, make sure you type: `https://hostname.local` in the browser, where `hostname` (of `AMSS-1234` format) is printed on the bottom of the AWS DeepRacer vehicle.

How to connect to the vehicle’s device console using its IP address

To connect to the device console using IP address as shown in the `device-status.txt` file (found on the USB drive), make sure the following conditions are met.

- Check your laptop or mobile devices are in the same network as the AWS DeepRacer vehicle.
- Check if you have connected to any VPN, if so, disconnect first.
- Try a different Wi-Fi network. For example, turn on personal hotspot on your phone.

How to get the MAC address for your vehicle

Follow the instructions below to get the MAC address of your AWS DeepRacer device:

1. Make sure that your AWS DeepRacer device is only connected to a Wi-Fi network.

2. Connect your AWS DeepRacer device to a monitor. You'll need a HDMI-to-HDMI, HDMI-to-DVI or similar cable and insert one end of the cable into the HDMI port on the vehicle's chassis and plug the other end into a supported display port on the monitor.
3. Connect a USB keyboard to your AWS DeepRacer using the USB port on the device's compute module, after the compute module is booted.
4. Type `deepracer` in the **Username** input field.
5. Type the device SSH password in the **Password** input field.

   If this is your first time logging in to the device, type `deepracer` in the **Password** input field. Reset the password, as required, before moving to the next step. You'll use the new password for future logins. For security reasons, use a complex or strong password phrase for the new password.
6. After logging in, open a Terminal window. You can use the Search button for the Terminal application.
7. Type the following Ubuntu shell command in the Terminal window:

   ```
   ifconfig | grep HWaddr
   ```

   The command produces an output similar to the following:

   ```
   mlan0 Link encap:Ethernet HWaddr 01:2a:34:b5:c6:de
   ```

   The hexadecimal numbers are the device's MAC address.

How to recover your vehicle's default password

Recovering your AWS DeepRacer device console default password involves retrieving or resetting the default password. The default password is printed on the bottom of the device.

Follow the instructions in the following procedure to recover the password for your AWS DeepRacer device web server using an Ubuntu terminal window.

1. Connect your AWS DeepRacer device to a monitor. You'll need a HDMI-to-HDMI, HDMI-to-DVI or similar cable and insert one end of the cable into the HDMI port on the vehicle's chassis and plug the other end into a supported display port on the monitor.
2. Connect a USB keyboard to your AWS DeepRacer using the USB port on the device's compute module, after the compute module is booted.

3. In the **Username**, enter `deepracer`.
4. In **Password**, enter the device SSH password. + If this is your first time to log in to the device, enter `deepracer` in **Password**. Reset the password, as required, before moving to the next step. You'll use the new password for future logins. For security reasons, use a complex or strong password phrase for the new password.
5. After you're logged in, open a terminal window. You can use the search button to find the terminal window application.
6. To get the default device console password, type the following command in the terminal window:

   ```
   $cat /sys/class/dmi/id/chassis_asset_tag
   ```

   The command outputs the default password as its result.
7. To reset the device console password to the default, run the following Python script in the terminal window:

   ```
   sudo python /opt/aws/deepracer/nginx/reset_default_password.py
   ```

How to manually update your vehicle

Recent changes in the AWS DeepRacer service has made certain legacy devices, such as those distributed at AWS re:Invent 2018, unable to update automatically. Follow the steps below to manually update such a device.

To manually update an AWS DeepRacer device

1. Download and unzip this device script.

   The default name of the uncompressed file for this script is `deepracer-device-manual-update.sh`. In this topic, we'll assume you use this default script file name.
2. Copy the downloaded and uncompressed the script file (`deepracer-device-manual-update.sh`) from your computer to a USB drive.
3. Connect the device to a monitor using a HDMI-HDMI cable, to a USB keyboard, and to a USB mouse.
4. Power on the device and sign into the OS after the device is booted up.

   You'll need to set the new OS password, if this is your first sign-in to the device.

5. Plug in the USB drive into the device and copy the script file to a folder (for example, ~/Desktop) on the device.
6. From a terminal on the device, type the following command to go to the script file's folder and to add execution permission to the script file:

```
cd ~/Desktop
chmod +x deepracer-device-manual-update.sh
```

7. Type the following shell command to run the script:

```
sudo -H ./deepracer-device-manual-update.sh
```

8. When done with updating the device, open a web browser on your computer or a mobile device and navigate to the device IP address, e.g., `192.168.1.11` in a home network or `10.56.101.13` in an office network.

   Make sure that your device is connected to your Wi-Fi network and use a browser in the same network without tunneling through a VPN.
9. On the device console, type the password for the device console to sign in. Wait for the update screen to show up. When prompted for further updates, follow the instructions therein.

How to diagnose and resolve common operational issues

As you explore reinforcement learning with your AWS DeepRacer vehicle, the device may become non functional. The following troubleshooting topics help you diagnose the problems and resolve the issues.

Topics

Why doesn't the video player on the device console show the video stream from my vehicle's camera?

After logging into the AWS DeepRacer device console, you don't see any live video streamed from the camera mounted on the AWS DeepRacer vehicle in the video player in **Device Controls**. The following could cause this issue:

- The camera might have a loose connection to the USB port. Unplug the camera module from the vehicle, replug it into the USB port, power off the device, and then power on the device to restart it.
- The camera might be defective. Use a known working camera from another AWS DeepRacer vehicle, if available, to test whether this is the cause.

Why doesn't my AWS DeepRacer vehicle move?

You powered on your AWS DeepRacer vehicle, but you can't get it to move. The following could cause this issue:

- The vehicle's power bank is not turned on or the power bank is not connected to the vehicle. Make sure to connect the provided USB-C-to-USB C cable between the USB-C port on the power bank and the USB-C port on the vehicle chassis. Verify that the LED indicators light up, which indicates the charge levels of the power bank. If not, push the power button on the power bank, and then push the power button on the vehicle's chassis to boot up the device. The device is booted up when its tail lights light up.
- If the power bank is on and the vehicle is booted up, but the vehicle does not move in either manual or autonomous driving mode, check if the vehicle's battery under the vehicle chassis is charged and turned on. If not, recharge the vehicle battery and then turn it on after the battery is fully charged.
- The vehicle battery cable connectors are not fully plugged into the device driving module power cable connector. Make sure the cable connectors are tightly coupled.
- The battery cables are defective. Test this battery on another working vehicle, if possible, to test whether this is the cause.
- The power switch of the vehicle battery is not turned on. Turn on the power switch and make sure you hear two beeps followed by a long beep.

Why don't I see the latest device update? How do I get the latest update?

Why is my AWS DeepRacer vehicle's software outdated?

- No automatic update is performed on the device lately. You may need to perform a manual update.
- The vehicle is not connected to the Internet. Make sure the vehicle is connected to a Wi-Fi or Ethernet network with internet access.

Why isn't my AWS DeepRacer vehicle connected to my Wi-Fi network?

When I check the network status on the vehicle's OS, I don't see the AWS DeepRacer vehicle connected to any Wi-Fi network. This could happen because of the following issues:

- No Wi-Fi has been configured for the AWS DeepRacer vehicle.
- The vehicle is out of the active network signal range. Make sure to operate the vehicle within the chosen Wi-Fi network range.
- The vehicle's pre-configured Wi-Fi network doesn't match the available Wi-Fi network.

Why does the AWS DeepRacer device console page take a long time to load?

When I tried to open the device console of my AWS DeepRacer vehicle, the device console page appears to take a long time to load.

- Your vehicle is down or off. Make sure the vehicle is powered on when the tail lights are on.
- The IP address of your vehicle has been changed, most likely by your network's DHCP server. To find out the vehicle's new IP address, sign in to the device console with the USB-US cable connection between your computer and the vehicle. View the new IP address in Settings. Alternatively, you can examine the list of devices attached to your network to discover the new IP address. If you're not a network administrator, ask the administrator to investigate this for you.

Why does a model fail to perform well when deployed to an AWS DeepRacer vehicle?

After training a model and deploying its artifacts to your AWS DeepRacer vehicle, sometimes the vehicle doesn't perform as expected. What went wrong?

In general, optimizing a trained model for transfer to a physical AWS DeepRacer vehicle is a challenging learning process. It often requires iterations through trial and error.

The following are some likely common factors affecting the model performance in your AWS DeepRacer vehicle:

- Your model has not converged in training. Clone the model to continue the training or retrain the model for a longer period of time. Make sure the agent continuously finishes laps in the simulation (that is, 100% process towards the end of the training).
- Your model was over-trained (that is, over-fitted). It fits too well to the training data, but doesn't generalize to unknown situations. Retrain the model with a more flexible or accommodating reward function or increase the granularities of the action space. You should also evaluate a trained model on different tracks to see if the model generalizes well.
- Your AWS DeepRacer vehicle might not have been calibrated properly. To test whether this is true, switch to manual driving and see if the vehicle drives as expected. If it doesn't, calibrate the vehicle.
- You are running the vehicle autonomously on a track that doesn't meet the requirements. For track requirements, see the Build your track section in this guide.
- There are too many objects close to the physical track, making the track significantly different from the simulated environment. Clear the track surroundings to make the physical track as close to the simulated one as possible.
- Reflection from the track surface or a near-by object can create glare to confuse the camera. Adjust lighting and avoid making the track on smooth-surfaced concrete floors or with other shiny materials.

Contact AWS Support

If you have AWS Business Support+, AWS Enterprise Support, or Unified Operations, you can use the AWS Support Center to get expert assistance with this solution. The following sections provide instructions.

Create case

1. Sign in to Support Center.
2. Choose **Create case**.

How can we help?

1. Choose **Technical**.
2. For **Service**, select **Solutions**.
3. For **Category**, select **Other Solutions**.
4. For **Severity**, select the option that best matches your use case.
5. When you enter the **Service**, **Category**, and **Severity**, the interface populates links to common troubleshooting questions. If you can't resolve your question with these links, choose **Next step: Additional information**.

Additional information

1. For **Subject**, enter text summarizing your question or issue.
2. For **Description**, describe the issue in detail, including the name of this product and the version you are using, such as this example: **DeepRacer on AWS vX.Y.Z**.
3. Choose **Attach files**.
4. Attach the information that AWS Support needs to process the request.

Help us resolve your case faster

1. Enter the requested information.
2. Choose **Next step: Solve now or contact us**.

Solve now or contact us

1. Review the **Solve now** solutions.
2. If you can't resolve your issue with these solutions, choose **Contact us**, enter the requested information, and choose **Submit**.

Update the solution

Updating the solution applies the latest features, security patches, and bug fixes to your deployment. To update to the latest version, refer to the appropriate section based on your original deployment method: AWS Launch Wizard or AWS CloudFormation.

Update using AWS Launch Wizard

The console automatically displays the newest available version of the solution in the **Deployment version** dropdown. If you have previously deployed the solution, follow this procedure to update your deployment to the latest version.

1. Go to Launch Wizard Deployments.
2. Select the deployment you want to update.
3. Choose **Actions**, then **Update deployment version**.
4. Select the latest version from the available **Deployment versions**.
5. Review configuration.
6. Make changes needed on each step.
7. Confirm the update.

Update using AWS CloudFormation

If you have previously deployed the solution, follow this procedure to update the CloudFormation stack to the latest version.

1. Sign in to the AWS CloudFormation console, select your existing DeepRacer on AWS CloudFormation stack, and choose **Update**.
2. Select **Replace current template**.
3. Enter the appropriate Amazon S3 URL:
   - If using the default main template: `https://solutions-reference.s3.amazonaws.com/deepracer-on-aws/latest/deepracer-on-aws-main.template`
4. Under **Parameters**, review the parameters for the template and modify them as necessary.
5. Choose **Next**.

6. On the **Configure stack options** page, choose **Next**.
7. On the **Review** page, review and confirm the settings. Be sure to check the box acknowledging that the template might create AWS Identity and Access Management (IAM) resources.
8. Choose **View change set** and verify the changes.
9. Choose **Update stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a status in approximately 30 minutes.

Uninstall the solution

To uninstall the solution, use the AWS Management Console or the AWS Command Line Interface (AWS CLI).

Using the AWS Management Console

AWS CloudFormation

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack.
3. Choose **Delete**.

AWS Launch Wizard

1. Sign in to the AWS Launch Wizard console.
2. On the **Launch Wizard Deployments** page, select this solution's deployment.
3. Choose **Actions**, then **Delete**.
4. Confirm the deletion.

Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, refer to What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>
```

Deleting retained resources

This solution is configured to retain certain resources after stack deletion to prevent accidental data loss.

Deleting Amazon S3 buckets

By default, the solution will retain any solution-created Amazon S3 buckets (for deploying in an opt-in region) to prevent accidental data loss if you decide to delete the AWS CloudFormation stack. After uninstalling the solution, you can manually delete these S3 buckets if you do not need to retain the data. Follow these steps to delete the Amazon S3 buckets.

1. Sign in to the Amazon S3 console.
2. Choose **Buckets** from the left navigation pane.
3. In the **Find buckets by name** field, enter the name of this solution's stack.
4. Select one of the solution's S3 buckets and choose **Empty**.
5. Enter **permanently delete** in the verification field and choose **Empty**.
6. Select the S3 bucket you just emptied and choose **Delete**.
7. Enter the S3 bucket name in the verification field and choose **Delete bucket**.

Repeat steps 4 through 7 until you delete all the S3 buckets.

To delete the S3 bucket using AWS CLI, run the following command:

```
$ aws s3 rb s3://<bucket-name> --force
```

Deleting Amazon CloudWatch Logs

Amazon CloudWatch Log Groups are also retained following stack deletion, and will remain in your AWS account until they are manually deleted. This configuration is by design to prevent loss of log data and to align with security and operational best practices.

1. Sign in to the Amazon CloudWatch console.
2. Choose **Log Management** from the left navigation pane.
3. In the **Filter log groups or try pattern search** field, enter the name of this solution's stack.
4. Select all log groups that belong to the solution.
5. In the **Actions** menu, click **Delete log group(s)**.
6. Review and confirm your selection by clicking **Delete**.

To delete a log group using AWS CLI, run the following command:

```
$ aws logs delete-log-group --log-group-name <log-group-name>
```

Developer guide

Source code

Visit our GitHub repository to view the source files for this solution, and to share your customizations with others. See the README.md file for more information.

Reference

This section includes information about data collection and a list of individuals who contributed to this solution.

Data collection

This solution sends operational metrics to AWS (the "Data") about the use of this solution. We use this Data to better understand how customers use this solution and related services and products. AWS's collection of this Data is subject to the AWS Privacy Notice.

Contributors

- Ryan Hayes
- Abhishek Patil
- Aijun Peng
- Sanjay Reddy Kandi
- Abe Wubshet
- Advait Dhamdhere
- Patrick Palmer
- Karim Siala
- Su Zhou
- Colin McCoy
- Taylor Rodrigues
- Tae Hong
- Pratik Nichat
- Eric Yang
- Adam Barker
- Nikhil Reddy
- Peter DeVries
- Shu Jackson
- Ken Spokas

- Fabien Houeto
- Celia Ng
- Joan Morgan
- Todd Bevins
- Don Barber
- Seyha Kry

Revisions

Visit the CHANGELOG.md in our GitHub repository to track version-specific improvements and fixes.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers, or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.