User Guide for version 1.17.0

AWS SimSpace Weaver

AWS SimSpace Weaver: User Guide for version 1.17.0

Table of Contents

What is AWS SimSpace Weaver?

AWS SimSpace Weaver is a service that you can use to build and run large-scale spatial simulations in the AWS Cloud. For example, you can create crowd simulations, large real-world environments, and immersive and interactive experiences.

With SimSpace Weaver, you can distribute simulation workloads across multiple Amazon Elastic Compute Cloud (Amazon EC2) instances. SimSpace Weaver deploys the underlying AWS infrastructure for you, and handles the simulation data management and network communication between the Amazon EC2 instances running your simulation.

Key concepts for SimSpace Weaver

A simulation or game is limited by the computer that runs it. As you grow the size and complexity of your virtual world, processing performance begins to degrade. Calculations take longer, systems run out of memory, and client frame rates drop. For simulations that don't need real-time performance, this might only be annoying. Or, it could be a business-critical situation in which increased processing delays result in increased costs. If your simulation or game needs real-time performance, then performance degradation is definitely a problem.

A common solution for a simulation that reaches a performance limit is to simplify the simulation. Online games with many users often address scaling problems by making copies of their virtual world on different servers and spreading users across them.

SimSpace Weaver solves the scaling problem by dividing your virtual world spatially and distributing the pieces across a cluster of compute instances that run in the AWS Cloud. The compute instances work together to process the entire simulation world in parallel. Your simulation world appears as a single integrated space to everything in it, and to all clients that connect to it. You no longer have to simplify a simulation because of a hardware performance limit. You can instead add more compute capacity in the cloud.

Topics

- Simulation clock
- Partitions
- State Fabric
- Entities
- Apps

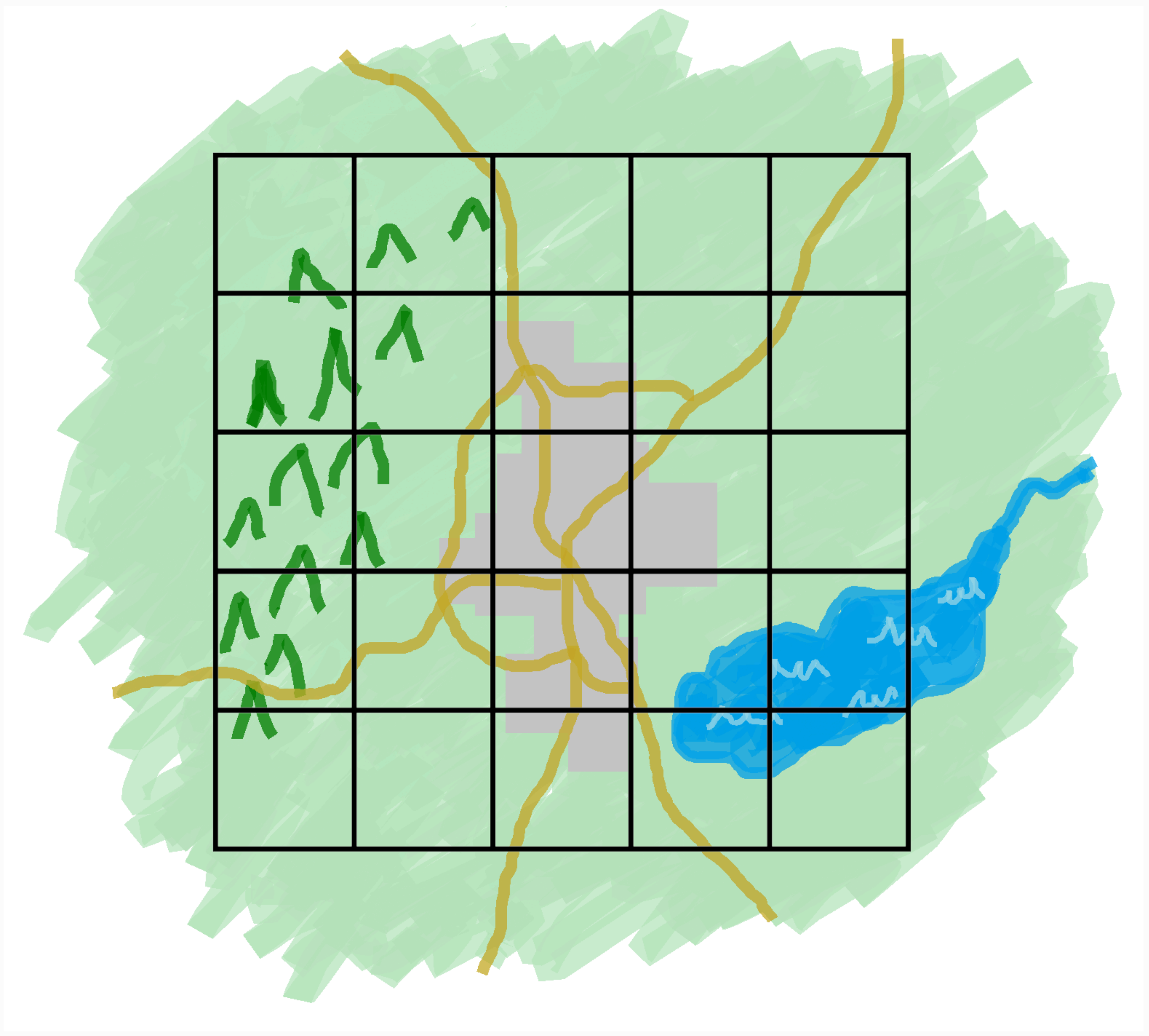
How SimSpace Weaver works

Your simulation consists of a world with objects in it. Some of the objects (such as people and vehicles) move and do things. Other objects (such as trees and buildings) are static. In SimSpace Weaver, an *entity* is an object in your simulation world.

You define the boundaries of your simulation world and divide it into a grid. Instead of creating simulation logic that operates on the entire grid, you create simulation logic that operates on one cell of the grid. In SimSpace Weaver, a *spatial app* is a program that you write that implements the simulation logic for a cell of your grid. This includes the logic for all entities in that cell. A spatial app's *ownership area* is the grid cell that the spatial app controls.

Note

In SimSpace Weaver, the term "app" can refer to the code of an app or a running instance of that code.



Your simulation world divided into a grid

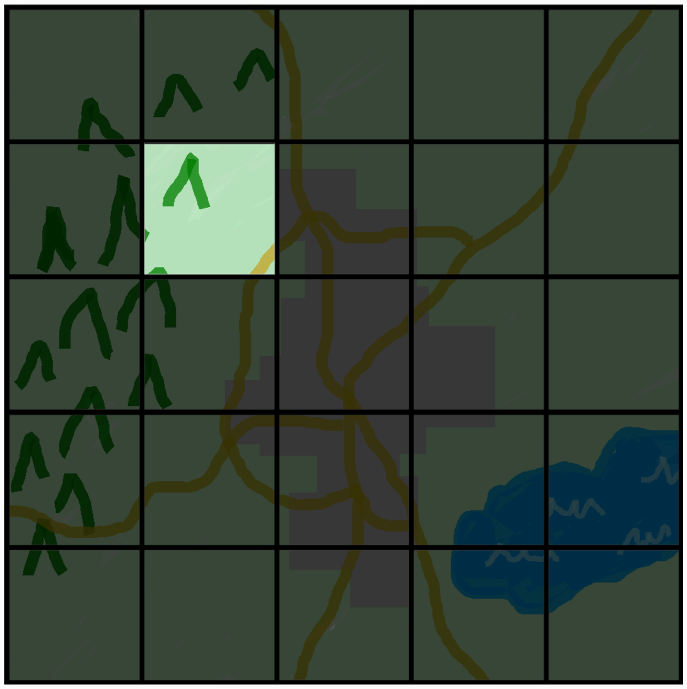
You divide your simulation world into a grid. Each spatial app implements simulation logic for a single cell in that grid.

SimSpace Weaver runs an instance of your spatial app code for each cell of your grid. All of the spatial app instances run in parallel. Essentially, SimSpace Weaver divides your overall simulation into multiple smaller simulations. Each of the smaller simulations handles a part of the overall simulation world. SimSpace Weaver can distribute and run these smaller simulations across multiple Amazon Elastic Compute Cloud (Amazon EC2) instances (called *workers*) in the AWS Cloud. A single worker can run multiple spatial apps.

Entities can move through the simulation world. If an entity enters another spatial app's ownership area (another cell in the grid), then the spatial app owner of the new area takes over control of the entity. If your simulation runs on multiple workers, then an entity could move from the control of a spatial app on one worker to a spatial app on a different worker. When an entity moves to a different worker, SimSpace Weaver handles the underlying network communication.

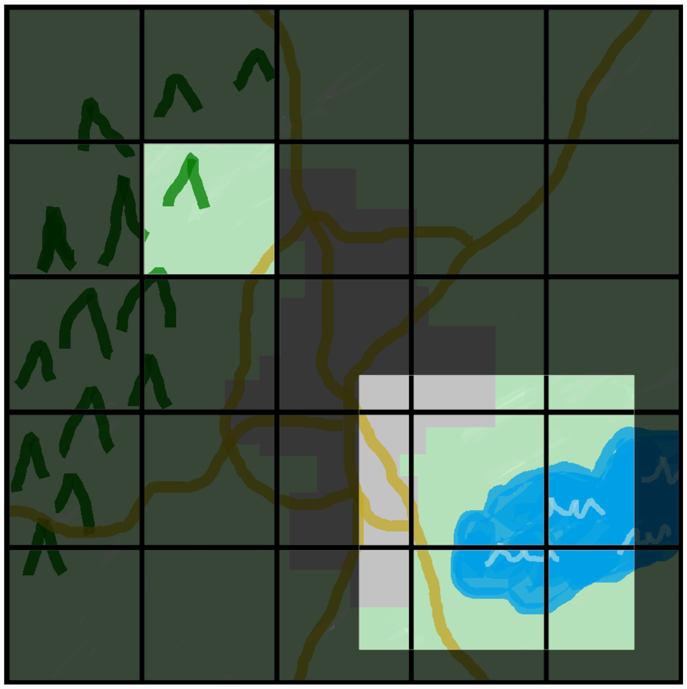
Subscriptions

A spatial app's view of the world is its own ownership area. To find out what's happening in another part of the simulation world, the spatial app creates a *subscription*. The *subscription area* is a subset of the overall simulation world area. A subscription area can include parts of multiple ownership areas, including the spatial app's own ownership area. SimSpace Weaver notifies the spatial app of all entity events (for example, enter, exit, create, update, and delete) that occur within the subscription area.



A spatial app's view of the world

A spatial app's view of the world is its ownership area, which is one cell in the world grid.



A spatial app's view with an added subscription area

A spatial app uses a subscription to find out what's happening in another part of the simulation world. The subscription area can contain multiple grid cells and parts of cells.

For example, an app that simulates entities interacting physically might need to know about entities just beyond the spatial boundaries of its ownership area. To accomplish this, the app can subscribe to areas that border its ownership area. After creating the subscription, the app receives notifications about entity events in those areas and it can read the entities. Another example is an autonomous vehicle that needs to see all entities 200 meters in front of it regardless of what app owns the area. The app for the vehicle can create a subscription with a filter as an axis-aligned bounding box (AABB) that covers the viewable area.

You can create simulation logic that isn't responsible for managing spatial aspects of your simulation. A *custom app* is an executable program that runs on a single worker. You control the lifecycle (start and stop) of a custom app. Simulation clients can connect to a custom app to view or interact with the simulation. You can also create a *service app* that runs on every worker. SimSpace Weaver starts an instance of your service app on every worker that runs your simulation.

Custom apps and service apps create subscriptions to learn about entity events and read entities. These apps don't have ownership areas because they aren't spatial. Using a subscription is the only way that they can find out what is happening in the simulation world.

How you use SimSpace Weaver

When you use SimSpace Weaver, these are the major steps that you follow:

1. Write and build C++ apps that integrate the SimSpace Weaver app SDK.
    a. Your apps make API calls to interact with the simulation state.
2. Write clients that view and interact with your simulation through some apps.
3. Configure your simulation in a text file.
4. Upload your app packages and simulation configuration to the service.
5. Start your simulation.
6. Start and stop your custom apps as needed.
7. Connect clients to your custom or service apps to view or interact with the simulation.
8. Check your simulation logs in Amazon CloudWatch Logs.
9. Stop your simulation.
10. Clean up your simulation.

Simulation schema

The *simulation schema* (or *schema*) is a YAML-formatted text file that contains configuration information for your simulation. SimSpace Weaver uses your schema when it starts a simulation. The SimSpace Weaver app SDK distributable package includes a schema for a sample project. You can use this as a starting point for your own schema. For more information about the simulation schema, see SimSpace Weaver simulation schema reference.

Workers and resource units

A *worker* is an Amazon EC2 instance that runs your simulation. You specify a worker type in your simulation schema. SimSpace Weaver maps your worker type to a specific Amazon EC2 instance type that the service uses. SimSpace Weaver starts and stops your workers for you, and manages network communication between the workers. SimSpace Weaver starts a set of workers for each simulation. Different simulations use different workers.

The available compute (processor and memory) capacity on a worker is divided into logical units called *compute resource units* (or *resource units*). A resource unit represents a fixed amount of processor and memory capacity.

Note

We previously referred to a compute resource unit as a *slot*. You might still see this previous term in our documentation.

Simulation clock

Each simulation has its own clock. You start and stop the clock using API calls or the SimSpace Weaver console. The simulation updates only when the clock is running. All operations in the simulation occur within time segments called *ticks*. The clock announces the start time of each tick to all workers.

The *clock rate* (or *tick rate*) is the number of ticks per second (hertz, or Hz) that the clock announces. The desired clock rate for a simulation is part of the simulation schema. All operations for a tick must complete before the next tick starts. For this reason, the effective clock rate can be lower than the desired clock rate. The effective clock rate won't be higher than the desired clock rate.

Partitions

A *partition* is a segment of the shared memory on a worker. Each partition holds part of the simulation state data.

A partition for a spatial app (also called a *spatial app partition* or *spatial partition*) contains all the entities in a spatial app's ownership area. SimSpace Weaver puts entities in spatial app partitions based on the spatial location of each entity. This means that SimSpace Weaver tries to place entities that are spatially near each other on the same worker. This minimizes the amount of knowledge that an app requires of entities that it doesn't own to simulate the entities that it does own.

State Fabric

The *State Fabric* is the system of shared memory (the collection of all partitions) on all workers. It holds all of the state data for your simulation.

The State Fabric uses a custom binary format that describes an entity as a set of initial data and an update log, for each data field of that entity. With this format, you can access the state of an entity at a previous point in simulation time and map it back to a point in real-world time. The buffer has a finite size, and it's not possible to go back in time beyond what's in the buffer. SimSpace Weaver uses a pointer to the current offset in the update log for each field, and it updates a pointer as part of a field update. SimSpace Weaver maps these update logs into an app’s process space using shared memory.

This object format results in low overhead and no serialization costs. SimSpace Weaver also uses this object format to parse and identify index fields (such as entity position).

Entities

An *entity* is the smallest building block of data in your simulation. Examples of entities include actors (such as people and vehicles) and static objects (such as buildings and obstacles). Entities have properties (such as position and orientation) that you can store as persistent data in SimSpace Weaver. Entities exist within partitions.

Apps

A SimSpace Weaver *app* is software that you write that contains custom logic that runs each simulation tick. The purpose of most apps is to update entities as the simulation runs. Your apps call APIs in the SimSpace Weaver app SDK to perform actions (such as reading and updating) on entities in your simulation.

You package your apps and their required resources (such as libraries) as .zip files and upload them to SimSpace Weaver. An app runs in a Docker container on a worker. SimSpace Weaver allocates each app a fixed number of resource units on the worker.

SimSpace Weaver assigns ownership of one (and only one) partition to each app. An app and its partition are located on the same worker. Each partition has only one app owner. An app can create, read, update, and delete entities in its partition. An app owns all entities in its partition.

There are three types of apps: *spatial apps*, *custom apps*, and *service apps*. They differ by use cases and lifecycles.

Note

In SimSpace Weaver, the term "app" can refer to the code for an app or a running instance of that code.

Spatial apps

Spatial apps update the state of entities that exist spatially in your simulation. For example, you might define a `Physics` app that's responsible for moving and colliding entities for every tick based on their velocity, shape, and size. In this case, SimSpace Weaver runs multiple instances of the `Physics` app in parallel to handle the size of the workload.

SimSpace Weaver manages the lifecycle of spatial apps. You specify an arrangement of spatial app partitions in your simulation schema. When you launch your simulation, SimSpace Weaver starts a spatial app for each spatial app partition. When you stop the simulation, SimSpace Weaver shuts down your spatial apps.

Other types of apps can create entities, but only spatial apps can update entities. Other types of apps must transfer entities that they create to a spatial domain. SimSpace Weaver uses the spatial location of an entity to move the entity to the partition of a spatial app. This transfers ownership of the entity to the spatial app.

Custom apps

You use *custom apps* to interact with your simulation. A custom app reads entity data using subscriptions. A custom app can create entities. However, the app must transfer an entity to a spatial app to include the entity in the simulation and update it. You can have SimSpace Weaver assign a network endpoint to a custom app. Simulation clients can connect to the network endpoint to interact with the simulation. You define your custom apps in your simulation schema, but you are responsible to start and stop them (using SimSpace Weaver API calls). After you start a custom app instance on a worker, SimSpace Weaver doesn't transfer the instance to another worker.

Service apps

You can use a *service app* when you need a read-only process running on every worker. For example, you can use a service app if you have a large simulation and you need a viewing client that moves through the simulation and displays only the visible entities to the user. In this case, a single custom app instance can’t process all the entities in the simulation. You can configure a service app to launch on every worker. Each of these service apps can then filter the entities on its assigned worker and send only the relevant entities to its connected clients. Your viewing client can then connect to different service apps as it moves through the simulation space. You configure service apps in your simulation schema. SimSpace Weaver starts and stops your service apps for you.

App summary

The following table summarizes the characteristics of the different types of SimSpace Weaver apps.

	Spatial apps	**Custom apps**	**Service apps**
Read entities	Yes	Yes	Yes
Update entities	Yes	No	No
Create entities	Yes	Yes*	Yes*
Lifecycle	Managed (SimSpace Weaver controls it.)	Unmanaged (You control it.)	Managed (SimSpace Weaver controls it.)
Start method	SimSpace Weaver starts one app	You start each app instance.	SimSpace Weaver starts one or more

	Spatial apps	Custom apps	Service apps
	instance for each spatial partition, as specified in your schema.		app instances on each worker, as specified in your schema.
Clients can connect	No	Yes	Yes

* When a custom app or service app creates an entity, the app must transfer ownership of the entity to a spatial app so that the spatial app can update the state of the entity.

Domains

A SimSpace Weaver *domain* is a collection of app instances that run the same executable app code and have the same launch options and commands. We refer to domains by the types of apps that they contain: spatial domains, custom domains, and service domains. You configure your apps within domains.

Subscriptions and replication

An app creates a *subscription* to a spatial region to learn entity events (for example, enter, exit, create, update, and delete) in that region. An app processes entity events from a subscription before it reads data for entities in partitions that it doesn't own.

A partition can exist on the same worker as the app (this is called a *local partition*), but another app can own the partition. A partition can also exist on a different worker (this is called a *remote partition*). If the subscription is to a remote partition, the worker creates a local copy of the remote partition through a process called *replication*. The worker then reads the local copy (replicated remote partition). If another app on the worker needs to read from that partition on the same tick, then the worker reads the same local copy.

Example use cases for SimSpace Weaver

You can use SimSpace Weaver for agent-based models and discrete time step simulations with a spatial component.

Create large crowd simulations

You can use SimSpace Weaver to simulate crowds in real-world environments. SimSpace Weaver enables you to scale your simulations to millions of dynamic objects with their own behaviors.

Create city-scale environments

Use SimSpace Weaver to create a digital twin of an entire city. Create simulations for city planning, to design traffic routing, and to plan environmental hazard response. You can use your own geospatial data sources as the building blocks for your environments.

Create immersive and interactive experiences

Create simulation experiences that multiple users can participate and interact in. Use popular development tools such as Unreal Engine and Unity to build 3-dimensional (3D) virtual worlds. Customize your 3D experience with your own content and behaviors.

Setting up for SimSpace Weaver

To get set up for using SimSpace Weaver for the first time, you must set up your AWS account and your local environment. When you complete these tasks, you will be ready for the getting started tutorials.

Setup tasks

1. Set up your AWS account to use SimSpace Weaver.
2. Set up your local environment for SimSpace Weaver.

Set up your AWS account to use SimSpace Weaver

Complete the following tasks to set up your AWS account to use SimSpace Weaver.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Add permissions to use SimSpace Weaver

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Creating a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Creating a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Example IAM policy to grant permissions to use SimSpace Weaver

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateAndRunSimulations",
            "Effect": "Allow",
            "Action": [
                "simspaceweaver:*",
                "iam:GetRole",
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:UpdateRole",
  "iam:CreatePolicy",
  "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "iam:GetRolePolicy",
                "iam:DeleteRolePolicy",
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:PutBucketPolicy",
                "s3:CreateBucket",
                "s3:ListBucket",
                "s3:PutEncryptionConfiguration",
                "s3:DeleteBucket",
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack",
                "cloudformation:DescribeStacks"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PassAppRoleToSimSpaceWeaver",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "simspaceweaver.amazonaws.com"
                }
            }
        }
    ]
}
```

Set up your local environment for SimSpace Weaver

SimSpace Weaver simulations run in containerized Amazon Linux 2 (AL2) environments. You must have an AL2 environment to compile and link your apps with the SimSpace Weaver app SDK. The standard local development environment is an AL2 container in Docker. If you choose not to use Docker, we provide alternate instructions to run an AL2 environment in Windows Subsystem for Linux (WSL). You can also use your own method to create a local AL2 environment. For some additional ways to run AL2 locally, see the Amazon EC2 documentation.

Important

Docker on Microsoft Windows is the standard development environment. For your convenience, we suggest other ways to set up your local development environment, but they are not standard and are unsupported.

Topics

Set up the SimSpace Weaver distribution package for Amazon Linux 2 (AL2) in Docker

This section provides instructions for setting up your local SimSpace Weaver distribution zip with an AL2 environment in Docker. For instructions to set up with AL2 in Windows Subsystem for Linux (WSL), see Set up the SimSpace Weaver distribution package for Amazon Linux 2 (AL2) in Windows Subsystem for Linux (WSL).

Requirements

- Microsoft Windows 10 or higher, or a compatible Linux system
- Microsoft Visual Studio 2019 or later, with the *Desktop development with C++* workload installed
- CMake3
- Git
- Docker Desktop
- AWS CLI
- Python 3.9

To set up the SimSpace Weaver distribution zip with AL2 in Docker

1. If you have not already configured your AWS credentials for the AWS CLI, follow these instructions: Configuring the AWS CLI.
2. Download the SimSpace Weaver app SDK distributable package. It contains the following:
   - Binaries and libraries for SimSpace Weaver app development
   - Helper scripts that automate parts of the development workflow
   - Sample applications that demonstrate SimSpace Weaver concepts
3. Unzip the file to an *`sdk-folder`* of your choice.
4. Go to the *`sdk-folder`*.
5. Enter the following command to install the required Python packages:

   ```
   pip install -r PackagingTools/python_requirements.txt
   ```
6. Enter the following command to setup the SimSpace Weaver distribution with a Docker image.

   ```
   python setup.py
   ```

   This command does the following:

   - Creates an AL2 docker image with all the requirements for building SimSpace Weaver projects installed.
   - Creates the CloudFormation resources required to launch a simulation.
     - The sample CloudFormation stack template can be found in *`sdk-folder`*`/PackagingTools/sample-stack-template.yaml`
   - Configures the provided sample projects with the correct paths for your local system.

Troubleshooting

- Docker appears stuck
  - If the console output appears to be stuck after Docker commands are called, try restarting the Docker engine. If that doesn't work, restart your computer.

Set up the SimSpace Weaver distribution package for Amazon Linux 2 (AL2) in Windows Subsystem for Linux (WSL)

This section provides instructions for setting up your SimSpace Weaver distribution zip with an AL2 environment in Windows Subsystem for Linux (WSL). For instructions to set up AL2 in Docker, see Set up the SimSpace Weaver distribution package for Amazon Linux 2 (AL2) in Docker.

> **Important**
>
> This section describes a solution that uses a version of AL2 that is not owned, developed, or supported by Amazon. This solution is provided for your convenience only, if you choose not to use Docker. Amazon and AWS assume no liability if you choose to use this solution.

Requirements

- Hyper-V on Windows 10
- Windows Subsystem for Linux (WSL)
- Third-party open source AL2 distribution for WSL (download version 2.0.20200722.0-update.2) (see the instructions)

> **Important**
>
> Our WSL instructions use the *2.0.20200722.0-update.2* version of the AL2 distribution for WSL. You might experience errors if you use any other version.

To set up the SimSpace Weaver distibution zip with AL2 in WSL

1. At a **Windows command prompt**, start your AL2 environment in WSL.

```
wsl -d Amazon2
```

> **Important**
>
> While you are running in WSL, include the `--al2` option when running one of the `quick-start.py` Python helper scripts located at `sdky-folder/Samples/sample-name/tools/cloud/quick-start.py`.

2. At a **Linux shell prompt**, update your yum package manager.

```
yum update -y
```

> **Important**
>
> If this step times-out, you might need to switch to WSL1 and retry these procedures. Exit your WSL AL2 session and enter the following at your **Windows command prompt**:
>
> ```
> wsl --set-version Amazon2 1
> ```

3. Install the unzip tool.

```
yum install -y unzip
```

4. Remove any AWS CLI that `yum` installed. Try both of the following commands if you are unsure if `yum` installed an AWS CLI.

```
yum remove awscli
```

```
yum remove aws-cli
```

5. Make a temporary directory and go to it.

```
mkdir ~/temp
cd ~/temp
```

6. Download and install the AWS CLI:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

```
unzip awscliv2.zip
./aws/install
```

7. You can remove the temporary directory.

   ```
   cd ~
   rm -rf temp
   ```

8. Restart the shell session to update the path in the environment.

   ```
   exec
   ```

9. Configure your AWS credentials for the AWS CLI in your AL2 environment. For more information, see Configuring the AWS CLI. If you use AWS IAM Identity Center, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*.

   ```
   aws configure
   ```

10. Install Git.

    ```
    yum install -y git
    ```

11. Install `wget`.

    ```
    yum install -y wget
    ```

12. Create a folder for the SimSpace Weaver app SDK.

    ```
    mkdir sdk-folder
    ```

13. Go to your SDK folder.

    ```
    cd sdk-folder
    ```

14. Download the SimSpace Weaver app SDK distributable package. It contains the following:

    - Binaries and libraries for SimSpace Weaver app development
    - Helper scripts that automate parts of the development workflow
    - Sample applications that demonstrate SimSpace Weaver concepts

```
wget https://artifacts.simspaceweaver.us-east-2.amazonaws.com/latest/
SimSpaceWeaverAppSdkDistributable.zip
```

15. Unzip the file.

```
unzip *.zip
```

16. Run the WSL setup script.

```
source ./setup-wsl-distro.sh
```

17. Enter the following command to install the required Python packages:

```
pip install -r PackagingTools/python_requirements.txt
```

18. Run the SimSpace Weaver distribution zip setup script:

```
python setup.py --samples --cloudformation
```

    This command does the following:

    - Creates the CloudFormation resources required to launch a simulation.
        - The sample CloudFormation stack template can be found in `*sdk-folder*/PackagingTools/sample-stack-template.yaml`
    - Configures the provided sample projects with the correct paths for your local system.

Note

You only need to do this one time for your AL2 environment in WSL.

Use of licensed software with AWS SimSpace Weaver

AWS SimSpace Weaver allows you to build simulations with your choice of simulation engine and content. In connection with your use of SimSpace Weaver, you are responsible for obtaining, maintaining, and adhering to the license terms of any software or content you use in your simulations. Verify your licensing agreement allows you to deploy your software and content in a virtual hosted environment.

Getting started with SimSpace Weaver

This section provides tutorials to help you get started with SimSpace Weaver. These tutorials introduce you to the general workflow for building simulations with SimSpace Weaver. These tutorials demonstrate how to create, deploy, and run simulations in SimSpace Weaver. We recommend that you begin with the quick start tutorial to get a simulation running in minutes. Go through the other tutorials after that to learn more.

These tutorials use a sample application (`PathfindingSample`) included in the SimSpace Weaver app SDK .zip file that you downloaded during the setup procedures. The sample application demonstrates the concepts that all SimSpace Weaver simulations share, including spatial partitioning, cross-partition entity handoff, apps, and subscriptions.

In the tutorials, you will create a simulation with four spatial partitions. A separate instance of the `PathfindingSample` spatial app manages each individual partition. The spatial apps create entities in their own partitions. The entities move to a particular position in the simulation world, avoiding obstacles as they move. You can use a separate client application (included in the SimSpace Weaver app SDK) to view the simulation.

Topics

Quick start tutorial for SimSpace Weaver

This tutorial guides you through the process to build and run a simulation on SimSpace Weaver in minutes. We recommend that you begin with this tutorial and then go through the detailed tutorial afterward.

Requirements

Before you begin, be sure that you completed the steps in Setting up for SimSpace Weaver.

Note

The scripts used here are provided for your convenience and are NOT required. See the detailed tutorial for how these steps can be done manually.

Step 1: Enable logging (optional)

To turn on logging

1. Navigate to:

   ```
   sdk-folder/Samples/PathfindingSample/tools
   ```

2. Open the schema file in a text editor:

   ```
   pathfinding-single-worker-schema.yaml
   ```

3. Find the `simulation_properties:` section at the beginning of the file:

   ```
   simulation_properties:
     default_entity_index_key_type: "Vector3<f32>"
   ```

4. Insert the following 2 lines after the line `simulation_properties:`:

   ```
     log_destination_service: "logs"
     log_destination_resource_name: "MySimulationLogs"
   ```

5. Confirm that your `simulation_properties:` section is the same as the following:

   ```
   simulation_properties:
     log_destination_service: "logs"
     log_destination_resource_name: "MySimulationLogs"
     default_entity_index_key_type: "Vector3<f32>"
   ```

6. Save the file and exit your text editor.

Step 2: Quick-start with the console client (option 1)

Navigate to:

```
sdk-folder/Samples/PathfindingSample/tools/cloud
```

Run one of the following commands:

- Docker: `python quick-start.py --consoleclient`
- WSL: `python quick-start.py —-consoleclient --al2`

By default, this will launch a simulation with a single partition on a single worker. Other configurations can be launched by passing the `--schema {file name}.yaml` from the `/Samples/PathfindingSample/tools/` folder.

Note

See Detailed tutorial: Learn the details while building the sample application for an in-depth explanation of what this script does.

Step 2: Quick-start with the Unreal Engine client (option 2)

See Launching the Unreal Engine view client.

Stop and delete your simulation

Navigate to:

```
sdk-folder/Samples/PathfindingSample/tools/cloud
```

Find the names of your simulations:

```
aws simspaceweaver list-simulations
```

Stop and delete the simulation

```
python stop-and-delete.py --simulation simulation-name
```

Troubleshooting

- **FileNotFoundError: cmake**

```
subprocess.run('cmake')
...
 FileNotFoundError: The system cannot find the file specified
```

  - **Resolution:** The script cannot find the command `cmake`. Please ensure you have the minimum recommended CMake version installed, and that it can be called with the `cmake` command in the PATH. Use the command `cmake -version` to verify.
- **ImportError: DLL load failed while importing libweaver_app_sdk_python_v1: The specified module could not be found.**
  - **Resolution:** This error occurs when the Python 3.9 is not being used to launch the Weaver Python SDK. Please ensure the python version associated with the "python" command is Python 3.9. You can check by running the `python --version` command.
- **Quick-start script appears stuck at after starting Docker Build.**
  - **Resolution:** Sometimes Docker needs a few minutes to warm up. If this issue persists for more than ~5 minutes, please restart Docker or your system.
- **target_compile_features no known features for CXX compiler "GNU":**
  - **Resolution:** Clear your Docker cache, delete the weaverappbuilder Docker image, delete your project build artifacts, and re run `setup.py`. This should reset your Docker environment and resolve the error.

Detailed tutorial: Learn the details while building the sample application

The quick start tutorial covered how to build, start, stop, and delete a sample simulation using `quick-start.py` and `stop-and-delete.py`. This tutorial will go over in detail how these scripts work, and the additional parameters these scripts can take to maximize flexibility for custom Weaver simulations.

Requirements

Before you begin, be sure that you completed the steps in Setting up for SimSpace Weaver.

Step 1: Enable logging (optional)

To turn on logging

1. Navigate to:

```
sdk-folder/Samples/PathfindingSample/tools
```

2. Open the schema file in a text editor:

```
pathfinding-single-worker-schema.yaml
```

3. Find the `simulation_properties:` section at the beginning of the file:

```
simulation_properties:
  default_entity_index_key_type: "Vector3<f32>"
```

4. Insert the following 2 lines after the line `simulation_properties:`:

```
  log_destination_service: "logs"
  log_destination_resource_name: "MySimulationLogs"
```

5. Confirm that your `simulation_properties:` section is the same as the following:

```
simulation_properties:
  log_destination_service: "logs"
  log_destination_resource_name: "MySimulationLogs"
  default_entity_index_key_type: "Vector3<f32>"
```

6. Save the file and exit your text editor.

Step 2: Start your simulation

As show in the quick start tutorial, the most basic steps to launch a sample simulation are:

1. Navigate to:

```
sdk-folder/Samples/PathfindingSample/tools/cloud
```

2. Run one of the follwing commands:

- **Docker:** `python quick-start.py`
- **WSL:** `python quick-start.py --al2`

This script automates common terminal commands, all of which can be executed manually using the AWS CLI. These steps are:

1. Upload the Weaver schema to S3.
   - SimSpace Weaver uses a schema to configure your simulation. The schema is a YAML-formatted plain text file. For more infomration, see Configuring your simulation.
2. Build and upload a custom container (optional).
   - If your schema defines a custom container, the quick-start script will build the docker image and upload it to Amazon ECR. For more information, see Custom containers. See the `PythonBubblesSample` schema for an example of this feature.
3. Build the project.
   - `quick-start.py` calls the `build_project` function defined in `build.py`. This step will vary depending on the project. For the PathfindingSample, CMake is used. The CMake and Docker command for which can be found in `build.py`.
4. Upload the build artifacts to S3.
   - You can check your S3 buckets to make sure that all of the uploads succeeded. For more information on using Amazon S3, see Creating, configuring, and working with Amazon S3 buckets in the *Amazon Simple Storage Service User Guide*.
   - The sample application zips and S3 bucket use the following name format:
     - `weaver-sample-bucket-account-number-region`
     - Spatial app: `ProjectNameSpatial.zip`
     - View (custom) app: `ProjectNameView.zip`
5. Start the Simulation.
   - This is a wrapper around the `aws simspaceweaver start-simulation` AWS CLI call. For more information, see the AWS CLI Command Reference for SimSpace Weaver.
   - The script will loop until the simulation status is either `STARTED` or `FAILED`. It can take a few minutes for a simulation to start.

6. Get the simulation details.

   - The **DescribeSimulation** API provides details about your simulation, including its state. A simulation can be in one of the following states:

     Simulation life cycle states

     1. **`STARTING`** – Initial state after you call **StartSimulation**
     2. **`STARTED`** – all spatial apps are launched and healthy
     3. **`STOPPING`** – Initial state after you call **StopSimulation**
     4. **`STOPPED`** – All compute resources are stopped
     5. **`DELETING`** – Initial state after you call **DeleteSimulation**
     6. **`DELETED`** – All resources assigned to the simulation are deleted
     7. **`FAILED`** – The simulation had a critical error/failure and stopped
     8. **`SNAPSHOT_IN_PROGRESS`** – A snapshot is in progress

     To get your simulation details

     1. Call the **ListSimulations** API.

        ```
        aws simspaceweaver list-simulations
        ```

        The script should display details about your each of your simulations, similar to the following:

        ```

        {
            "Status": "STARTED",
            "CreationTime": 1664921418.09,
            "Name": "MyProjectSimulation_22-10-04_22_10_15",
            "Arn": "arn:aws:simspaceweaver:us-west-2:111122223333:simulation/
        MyProjectSimulation_22-10-04_22_10_15",
            "TargetStatus": "STARTED"
        }
        ```

     2. Call **DescribeSimulation** to get your simulation details. Substitute *`simulation-name`* with the **Name** of your simulation from the output of the previous step.

```
aws simspaceweaver describe-simulation --simulation simulation-name
```

The script should display more details about the simulation that you specified, similar to the following:

```

{
    "Status": "STARTED",
    "CreationTime": 1664921418.09,
    "Name": "MyProjectSimulation_22-10-04_22_10_15",
    "Arn": "arn:aws:simspaceweaver:us-west-2:111122223333:simulation/
MyProjectSimulation_22-10-04_22_10_15",
    "TargetStatus": "STARTED"
}
```

7. Start custom apps.

   - SimSpace Weaver doesn't manage the lifecycle of custom apps. You must start your custom apps. It's best practice to start your custom apps before you start your simulation clock, but you can start custom apps after you start the clock.

     You can call the **StartApp** API to start your custom apps.

     ```
     aws simspaceweaver start-app --simulation simulation-name --name app-name --
     domain domain-name
     ```

     The **StartApp** API call will create and start a new instance of the custom app using the name that you provide. If you provide the name of an app that already exists then you will receive an error. If you want to restart a particular app (instance), you must first stop that app and delete it.

     Note

     The status of your simulation must be `STARTED` before you can start custom apps.

     The sample application provides the `ViewApp` custom app to view your simulation. This app provides you with a static IP address and port number to connect the simulation clients (you will do this in a later step in this tutorial). You can think of a **domain** as a class of apps that have the same executable code and launch options. The **app name** identifies the instance of the app. For more information on SimSpace Weaver concepts, see Key concepts for SimSpace Weaver.

You can use the **DescribeApp** API to check the status of a custom app after you start it.

```
aws simspaceweaver describe-app --simulation simulation-name --app app-name --
domain domain-name
```

To start the view app in this tutorial

1. Call **StartApp** for `ViewApp`.

   ```
   aws simspaceweaver start-app --simulation simulation-name --name ViewApp --
   domain MyViewDomain
   ```

2. Call **DescribeApp** to check the status of your custom app.

   ```
   aws simspaceweaver describe-app --simulation simulation-name --app ViewApp --
   domain MyViewDomain
   ```

After the status of your custom app (instance) is `STARTED`, the output of **DescribeApp** will include the IP address and port number for that custom app (instance). In the following example output, the IP address is the value of `Address` and the port number is the value of `Actual` in the `EndpointInfo` block.

```


{
    "Status": "STARTED",
    "Domain": "MyViewDomain",
    "TargetStatus": "STARTED",
    "Simulation": "MyProjectSimulation_22-10-04_22_10_15",
    "LaunchOverrides": {
        "LaunchCommands": []
    },
    "EndpointInfo": {
        "IngressPortMappings": [
            {
```

```
                    "Declared": 7000,
                    "Actual": 4321
                }
            ],
            "Address": "198.51.100.135"
        },
        "Name": "ViewApp"
}
```

> **Note**
>
> The value of `Declared` is the port number that your app code should bind to. The value of `Actual` is the port number that SimSpace Weaver exposes to clients to connect to your app. SimSpace Weaver maps the `Declared` port to the `Actual` port.

> **Note**
>
> You can use the the procedure described at Get the IP address and port number of a custom app to get the IP address and port number of any started custom app.

8. Start the clock.

   - When you first create your simulation, it has a clock but the clock isn't running. When your clock isn't running, your simulation won't update its state. After you start the clock, it will begin sending ticks to your apps. Each tick, your spatial apps step through the entities they own and commit the results to SimSpace Weaver

   > **Note**
   >
   > It can take 30-60 seconds to start the clock.

   Call the **StartClock** API.

   ```
   aws simspaceweaver start-clock --simulation simulation-name
   ```

Note

The **StartClock** API uses your *simulation-name*, which you can find using the **ListSimulations** API:

```
aws simspaceweaver list-simulations
```

quick-start parameters

- -h, --help
  - List these parameters.
- --clean
  - Delete the contents of the build directory before building.
- --al2
  - Builds directly on the native machine instead of Docker. Only use this if running in an Amazon Linux 2 environment, such as WSL.
- --uploadonly
  - Only upload the schema and app zips to Amazon S3, don't start the simulation.
- --nobuild
  - Skip rebuilding the project.
- --nocontainer
  - Skip rebuilding the simulation container listed in the schema.
- --consoleclient
  - Automatically build and connect the console client listed in config.py.
- --schema SCHEMA
  - What schema this invocation will use. Defaults to value of 'SCHEMA' in config.py.
- --name NAME
  - What name the simulation will have. Defaults to the value of 'PROJECT_NAME'-date-time in config.py.

Step 3: Check the logs (optional)

SimSpace Weaver writes simulation management messages and the console output from your apps to Amazon CloudWatch Logs. For more information on working with logs, see [Working with log groups and log streams](#) in the *Amazon CloudWatch Logs User Guide*.

Each simulation that you create has its own log group in CloudWatch Logs. The name of the log group is specified in the simulation schema. In the following schema snippet, the value of `log_destination_service` is `logs`. This means that the value of `log_destination_resource_name` is the name of a log group. In this case, the log group is `MySimulationLogs`.

```
simulation_properties:
  log_destination_service: "logs"
  log_destination_resource_name: "MySimulationLogs"
  default_entity_index_key_type: "Vector3<f32>"
```

You can also use the **DescribeSimulation** API to find the name of the log group for simulation after you start it.

```
aws simspaceweaver describe-simulation --simulation simulation-name
```

The following example shows the part of the output from **DescribeSimulation** that describes the logging configuration. The name of the log group is shown at the end of the `LogGroupArn`.

```
    "LoggingConfiguration": {
        "Destinations": [
            {
                "CloudWatchLogsLogGroup": {
                    "LogGroupArn": "arn:aws:logs:us-west-2:111122223333:log-
group:MySimulationLogs"
                }
            }
        ]
    },
```

Each simulation log group contains several log streams:

- **Management log stream** – simulation management messages produced by the SimSpace Weaver service.

  ```
  /sim/management
  ```

- **Errors log stream** – error messages produced by the SimSpace Weaver service. This log stream only exists if there are errors. SimSpace Weaver stores errors written by your apps in their own app log streams (see the following log streams).

  ```
  /sim/errors
  ```

- **Spatial app log streams** (1 for each spatial app on each worker) – console output produced by spatial apps. Each spatial app writes to its own log stream. The *spatial-app-id* is all characters after the trailing slash at the end of the *worker-id*.

  ```
  /domain/spatial-domain-name/app/worker-worker-id/spatial-app-id
  ```

- **Custom app log streams** (1 for each custom app instance) – console output produced by custom apps. Each custom app instance writes to its own log stream.

  ```
  /domain/custom-domain-name/app/custom-app-name/random-id
  ```

- **Service app log streams** (1 for each service app instance) – console output produced by service apps. Each service app writes to its own log stream. The *service-app-id* is all characters after the trailing slash at the end of the *service-app-name*.

  ```
  /domain/service-domain-name/app/service-app-name/service-app-id
  ```

Note

The sample application doesn't have service apps.

Step 4: View your simulation

The SimSpace Weaver app SDK provides different options to view the sample application. You can use the sample console client if you don't have any local support for Unreal Engine development. The instructions for the Unreal Engine client assume that you are using Windows.

The console client displays a list of entity events as they occur. The client gets the entity event information from the `ViewApp`. If your console client displays the list of events, then it confirms network connectivity with the `ViewApp` and activity in your simulation.

The `PathfindingSample` simulation creates stationary and moving entities on a 2-dimensional plane. The moving entities move around the stationary entities. The Unreal Engine client provides a visualization of the entity events.

Console client

The console client can be automatically built and connected when launching a sample with `quick-start.py` if you include the `--consoleclient` option. To build and connect the console client after `quick-start.py` has already been called, do the following:

Navigate to:

```
sdk-folder/Clients/TCP/CppConsoleClient
```

Run the script to build and connect the client:

```
python start_client.py --host ip-address --port port-number
```

The script will do the following:

1. Build the console client with CMake.
2. Launch the built executable with the given IP address and port number.

    ```
    .\WeaverNngConsoleClient.exe --url tcp://ip-address:port-number
    ```

Unreal Engine client

See Launching the Unreal Engine view client.

Step 5: Stop and delete your simulation

Navigate to:

```
sdk-folder/Samples/PathfindingSample/tools/cloud
```

Find the names of your simulations:

```
aws simspaceweaver list-simulations
```

Stop and delete the simulation:

```
python stop-and-delete.py --simulation simulation-name
```

The script `stop-and-delete.py` will do the following:

1. Call the AWS CLI command to stop a simulation.
   - `aws simspaceweaver stop-simulation`
   - For more information, see AWS CLI Command Reference for SimSpace Weaver.
2. Call the AWS CLI command to delete a simulation.
   - `aws simpaceweaver delete-simulation`
   - For more information, see AWS CLI Command Reference for SimSpace Weaver.

stop-and-delete parameters

- -h, --help
  - List these parameters.
- --simulation SIMULATION
  - The name of the simulation to stop-and-delete
- --stop
  - Only stop the simulation. Doesn't delete it.
- --delete
  - Only delete a simulation. Will only work if he simulation is either `STOPPED` or `FAILED`.

Troubleshooting

See Troubleshooting in the quick start tutorial.

Working with SimSpace Weaver

This chapter provides information and guidance to help you build your own applications in SimSpace Weaver.

Topics

Configuring your simulation

A **simulation schema** (or **schema**) is a YAML-formatted text file that specifies the configuration for a simulation. You can use the same schema to start multiple simulations. The schema file is located in the project folder for your simulation. You can use any text editor to edit the file. SimSpace Weaver only reads your schema when it starts the simulation. Any edits that you make to a schema file only affect new simulations that you start after the edits.

To configure your simulation, edit your simulation schema file (use the appropriate path separator for your operating system):

```
project-folder\tools\project-name-schema.yaml
```

You upload the simulation schema when you create a new simulation. The quick start helper script for your project will upload the schema as part of its process to build your simulation:

```
project-folder\tools\windows\quick-start.py
```

For more information about running the quick-start script, see the Detailed tutorial in the Getting started chapter of this guide.

Simulation configuration parameters

The simulation schema contains bootstrapping information, including:

- **Simulation properties** – SDK version and compute configuration (type and number of workers)
- **Clocks** – tick rate and tolerances
- **Spatial partitioning strategies** – spatial topology (such as a grid), bounds, and placement groups (spatial partition grouping on workers)
- **Domains and their apps** – app bucket, path, and launch command(s)

SimSpace Weaver uses your schema configuration to configure and arrange spatial partitions, launch apps, and advance the simulation at your specified tick rate.

Note

The create-project script in the SimSpace Weaver app SDK will automatically generate a simulation schema for you, based on the sample application.

The following topics describe the parameters in the simulation schema. For a full description of the simulation schema, see SimSpace Weaver simulation schema reference.

Topics

- SDK version
- Simulation properties

- Workers
- Clock
- Partitioning strategies
- Domains

SDK version

The `sdk_version` field specifies the version of SimSpace Weaver that the schema is formatted for. Valid values: `1.17`, `1.16`, `1.15`, `1.14`, `1.13`, `1.12`

Important

The value of `sdk_version` only includes the major version number and first minor version number. For example, the value `1.12` specifies all versions `1.12.x`, such as `1.12.0`, `1.12.1`, and `1.12.2`.

Simulation properties

The `simulation_properties` section of your schema specifies the logging configuration and a data type for the index field (usually the spatial location) of entities.

```
simulation_properties:
  log_destination_service: "logs"
  log_destination_resource_name: "MySimulationLogs"
  default_entity_index_key_type: "Vector3<f32>"
```

The value of `log_destination_service` determines the interpretation of the value of `log_destination_resource_name`. Currently, the only supported value is `logs`. This means that the value of `log_destination_resource_name` is the name of a log group in Amazon CloudWatch Logs

Note

Logging is optional. If you don't configure log destination properties then your simulation won't produce logs.

The `default_entity_index_key_type` property is required. The only valid value is `Vector3<f32>`.

Workers

The `workers` section specifies the type and number of workers that you want for your simulation. SimSpace Weaver uses its own worker types that map to Amazon EC2 instance types.

```
workers:
  MyComputeWorkers:
    type: "sim.c5.24xlarge"
    desired: 1
```

Enabling multi-worker simulations

You can create a simulation that uses more than 1 worker. By default, simulations use 1 worker. You must modify your simulation schema before you start the simulation.

Note

You can't change a simulation that has already started. If you want to enable multi-worker for a running simulation, you must stop and delete the simulation first.

To use more than one worker, set the `desired` number of compute instances to a value greater than 1. There is a maximum number of apps for each worker. For more information, see SimSpace Weaver endpoints and quotas. SimSpace Weaver will only use more than 1 worker when the number of apps on a worker exceeds this limit. SimSpace Weaver can place an app on any of the available workers. App placement on a specific worker isn't guaranteed.

The following schema snippet demonstrates a configuration for a simulation that requests 2 workers. SimSpace Weaver will attempt to allocate the second worker if the number of apps exceeds the maximum number of apps for 1 worker.

```
workers:
  MyComputeWorkers:
    type: "sim.c5.24xlarge"
    desired: 2
```

Clock

The `clock` section specifies properties of the simulation clock. Currently, you can only configure the **tick rate** (the number of ticks per second that the clock sends to apps). The tick rate is a maximum rate. The effective tick rate could be lower because all operations (such as entity updates) for a tick must finish before the next tick can start. The tick rate is also called the **clock rate**.

The valid values for `tick_rate` depend on the `sdk_version` specified in your schema.

Valid values for the tick rate

- Versions earlier than `"1.14"`:
  - `10`
  - `15`
  - `30`
- Version `"1.14"` or later:
  - `"10"`
  - `"15"`
  - `"30"`
  - `"unlimited"`

    For more information, see Unlimited tick rate.

Important

- For schemas with a `sdk_version` earlier than `"1.14"` the value of `tick_rate` is an **integer**, such as `30`.
- For schemas with a `sdk_version` of `"1.14"` or later, the value of `tick_rate` is a **string**, such as `"30"`. The value **must include the double quotes**.

  If you convert a version `"1.12"` or `"1.13"` schema to version `"1.14"` or later, you must enclose the value of `tick_rate` in double quotes.

Unlimited tick rate

You can set the `tick_rate` to `"unlimited"` to enable your simulation to run as fast as your code can execute. With an unlimited tick rate, SimSpace Weaver sends the next tick immediately after all apps finish the commits for the current tick.

> **Important**
>
> Unlimited tick rate isn't supported in SimSpace Weaver versions before 1.14.0. The minimum value of `sdk_version` in the schema is `"1.14"`.

Unlimited tick rate in SimSpace Weaver Local

SimSpace Weaver Local implements `"unlimited"` as if the schema specified a tick rate of 10 kHz (10000). The effect is the same as an unlimited tick rate in the AWS Cloud. You still specify `tick_rate: "unlimited"` in your schema. For more information about SimSpace Weaver Local, see Local development in SimSpace Weaver.

Frequently asked questions about the clock

Q1. Can I change a STARTED simulation to use a different tick rate?

You can't change the tick rate of a simulation that already exists in the AWS Cloud at any stage of its life cycle. You also can't change the tick rate of a simulation running in SimSpace Weaver Local. You can set the `tick_rate` in the schema and start a new simulation from that schema.

Q2. Can I run my simulation with an unlimited tick rate in a version earlier than 1.14?

No, unlimited tick rate isn't supported in versions before 1.14.0.

Troubleshooting clock errors

If your simulation fails to start, you can check the value of `"StartError"` in the output of the **DescribeSimulation** API. An invalid `tick_rate` value in your schema will produce the following errors.

Note

The error output shown here is displayed on multiple lines to improve readability. The actual error output is a single line.

- The `sdk_version` is earlier than `"1.14"` and the value of `tick_rate` is an invalid integer. Valid values: `10`, `15`, `30`

```
"[{\"errorType\":\"SchemaFormatInvalid\",\"errorMessage\":
    \"$.clock.tick_rate: does not have a value in the enumeration [10, 15, 30]\"}]"
```

- The `sdk_version` is earlier than `"1.14"` and the value of `tick_rate` is a string. Valid values: `10`, `15`, `30`

```
"[{\"errorType\":\"SchemaFormatInvalid\",\"errorMessage\":
    \"$.clock.tick_rate: does not have a value in the enumeration [10, 15, 30]\"},
    {\"errorType\":\"SchemaFormatInvalid\",
    \"errorMessage\":\"$.clock.tick_rate: string found, integer expected\"}]"
```

- The `sdk_version` is `"1.14"` or later and the value of `tick_rate` is an invalid string. Valid values: `"10"`, `"15"`, `"30"`, `"unlimited"`

```
"[{\"errorType\":\"SchemaFormatInvalid\",\"errorMessage\":
    \"$.clock.tick_rate: does not have a value in the enumeration [10, 15, 30,
 unlimited]\"}]"
```

- The `sdk_version` is `"1.14"` or later and the value of `tick_rate` is an integer. Valid values: `"10"`, `"15"`, `"30"`, `"unlimited"`

```
"[{\"errorType\":\"SchemaFormatInvalid\",\"errorMessage\":
    \"$.clock.tick_rate: does not have a value in the enumeration [10, 15, 30,
 unlimited]\"},
    {\"errorType\":\"SchemaFormatInvalid\",
    \"errorMessage\":\"$.clock.tick_rate: integer found, string expected\"}]"
```

Partitioning strategies

The `partitioning_strategies` section specifies configuration properties for the partitions of spatial apps. You provide your own name for a partitioning strategy (a set of properties in this section) and use it in your spatial app configuration.

```
partitioning_strategies:
  MyGridPartitioning:
    topology: "Grid"
    aabb_bounds:
      x: [0, 1000]
      y: [0, 1000]
    grid_placement_groups:
      x: 1
      y: 1
```

The `topology` property specifies the type of coordinate system that your simulation uses. The value `Grid` specifies a 2-dimensional (2D) grid.

For a `Grid` topology, the simulation space is modeled as an axis-aligned bounding box (AABB). You specify the coordinate bounds for each axis of your AABB in the `aabb_bounds` property. All entities that exist spatially in your simulation must have a position inside the AABB.

Grid placement groups

A **placement group** is a collection of spatial app partitions that you want SimSpace Weaver to place on the same worker. You specify the number and arrangement of placement groups (in a grid) in the `grid_placement_groups` property. SimSpace Weaver will attempt to evenly distribute the partitions across the placement groups. The ownership areas of spatial apps with partitions in the same placement group will be spatially adjacent.

We recommend that x * y is equal to your desired number of workers. If it isn't equal, SimSpace Weaver will attempt to balance your placement groups across the available workers.

If you don't specify a placement group configuration, SimSpace Weaver will calculate one for you.

Domains

You provide a name for a set of configuration properties for a domain. The launch setting for apps in a domain determines the type of domain:

- **`launch_apps_via_start_app_call`** – custom domain
- **`launch_apps_by_partitioning_strategy`** – spatial domain
- **`launch_apps_per_worker`** (not included in the sample application) – service domain

Important

SimSpace Weaver supports up to 5 domains for each simulation. This includes all spatial, custom, and service domains.

```
domains:
  MyViewDomain:
    launch_apps_via_start_app_call: {}
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MyViewApp.zip"
      launch_command: ["MyViewApp"]
      required_resource_units:
        compute: 1
      endpoint_config:
        ingress_ports:
          - 7000
  MySpatialDomain:
    launch_apps_by_partitioning_strategy:
      partitioning_strategy: "MyGridPartitioning"
      grid_partition:
        x: 2
        y: 2
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MySpatialApp.zip"
      launch_command: ["MySpatialApp"]
      required_resource_units:
        compute: 1
```

Note

SimSpace Weaver app SDK version 1.12.x projects use separate buckets for the app .zip files and the schema:

- weaver-*`lowercase-project-name-account-number`*-app-zips-*`region`*

- weaver-*lowercase-project-name-account-number*-schemas-*region*

Topics

App configuration

You specify the configuration of an app (`app_config`) as part of the configuration for its domain. All types of domains use these same app configuration properties.

```
app_config:
  package: "s3://weaver-myproject-111122223333-us-west-2/MyViewApp.zip"
  launch_command: ["MyViewApp"]
  required_resource_units:
    compute: 1
```

Note

SimSpace Weaver app SDK version 1.12.x projects use separate buckets for the app .zip files and the schema:

- weaver-*lowercase-project-name-account-number*-app-zips-*region*
- weaver-*lowercase-project-name-account-number*-schemas-*region*

The `package` property specifies the S3 URI of a zip file in an S3 bucket. The zip file contains the app executable (also called a *binary*) and any other resources that it needs (such as libraries). Each instance of the app executable runs in a Docker container on a worker.

The `launch_command` property specifies the name of the executable and any command-line options to run the app. The value of `launch_command` is an array. Each token of the entire launch command string is an element in the array.

Example

- For the launch command: `MyTestApp --option1 value1`
- Specify: `launch_command: ["MyTestApp", "-option1", "value1"]`

The `required_resource_units` property specifies the number of compute resource units that SimSpace Weaver should allocate to this app. A compute resource unit is a fixed amount of processing capacity (vCPU) and memory (RAM) on a worker. You can increase this value to increase the amount of computing power available to the app when it runs on a worker. There is a limited number of compute resource units on each worker. For more information, see SimSpace Weaver endpoints and quotas.

Configuring spatial domains

For spatial domains, you must specify a `partitioning_strategy`. The value of this property is the name that you gave to a partitioning strategy that you defined in another part of the schema.

```
MySpatialDomain:
  launch_apps_by_partitioning_strategy:
    partitioning_strategy: "MyGridPartitioning"
    grid_partition:
      x: 2
      y: 2
  app_config:
    package: "s3://weaver-myproject-111122223333-us-west-2/MySpatialApp.zip"
    launch_command: ["MySpatialApp"]
    required_resource_units:
      compute: 1
```

Note

SimSpace Weaver app SDK version 1.12.x projects use separate buckets for the app .zip files and the schema:

- weaver-*lowercase-project-name-account-number*-app-zips-*region*
- weaver-*lowercase-project-name-account-number*-schemas-*region*

A partitioning strategy with a `Grid` topology (the only topology supported in this release) directs SimSpace Weaver to arrange spatial app partitions of this domain in a grid. The `grid_partition` property specifies the number rows and columns of the partition grid.

SimSpace Weaver will start 1 instance of the spatial app for each cell in the partition grid. For example, if a spatial domain has `grid_partition` values `x: 2` and `y: 2`, there are 2 * 2 = 4 partitions in the spatial domain. SimSpace Weaver will start 4 instances of the app configured in the spatial domain and assign 1 partition to each app instance.

Topics

Resource requirements for spatial domains

You can assign up to 17 compute resource units for each worker. You specify the number of compute resource units that each spatial app uses in the `app_config` section of your spatial domain.

Example schema snippet showing the compute resource units for a spatial app

```
  MySpatialDomain:
    launch_apps_by_partitioning_strategy:
      partitioning_strategy: "MyGridPartitioning"
      grid_partition:
        x: 2
        y: 2
    app_config:
      package: "s3://weaver-myproject-111122223333-artifacts-us-west-2/
MySpatialApp.zip"
      launch_command: ["MySpatialApp"]
      required_resource_units:
        compute: 1
```

To calculate the number of compute resource units that a domain requires, multiply the number of cells in your grid (in your `grid_partition`, x * y) by the number of compute resource units assigned to the spatial apps.

For the previous example, the domain `MySpatialDomain` specifies:

- `x`: 2
- `y`: 2
- `compute`: 1

The grid for `MySpatialDomain` has 2 * 2 = 4 cells. The spatial domain requires 4 * 1 = 4 compute resource units.

The total number of compute resource units for all domains specified in your schema must be less than or equal to the `desired` number of workers multiplied by the maximum number of compute resource units for each worker (17).

Multiple spatial domains

You can configure your simulation to use more than 1 spatial domain. For example, you can use 1 spatial domain to control the main actors in a simulation (such as people and cars) and a different spatial domain to control the environment.

You can also use multiple spatial domains to assign different resources to different parts of your simulation. For example, if your simulation has a type of entity that has 10x more entity instances than another type, you can create different domains to handle each entity type and allocate more resources for the domain with more entities.

Important

SimSpace Weaver versions before 1.14.0 don't support multiple spatial domains.

Important

AWS SimSpace Weaver Local doesn't currently support multiple spatial domains. For more information about SimSpace Weaver Local, see Local development in SimSpace Weaver.

Important

SimSpace Weaver supports up to 5 domains for each simulation. This includes all spatial, custom, and service domains.

Configure multiple spatial domains

To configure more than 1 spatial domain, add the other spatial domain definitions as separate named sections in your schema. Each domain must specify the `launch_apps_by_partitioning_strategy` key. See the following example schema.

```
sdk_version: "1.14"
workers:
  MyComputeWorkers:
    type: "sim.c5.24xlarge"
    desired: 1
clock:
  tick_rate: "30"
partitioning_strategies:
  MyGridPartitioning:
    topology: Grid
    aabb_bounds:
      x: [0, 1000]
      y: [0, 1000]
domains:
  MySpatialDomain:
    launch_apps_by_partitioning_strategy:
      partitioning_strategy: "MyGridPartitioning"
      grid_partition:
        x: 2
        y: 2
    app_config:
      package: "s3://weaver-myproject-111122223333-artifacts-us-west-2/
MySpatialApp.zip"
      launch_command: ["MySpatialApp"]
      required_resource_units:
        compute: 1
  MySecondSpatialDomain:
    launch_apps_by_partitioning_strategy:
      partitioning_strategy: "MyGridPartitioning"
      grid_partition:
        x: 2
        y: 2
    app_config:
      package: "s3://weaver-myproject-111122223333-artifacts-us-west-2/
MySpatialApp2.zip"
      launch_command: ["MySpatialApp2"]
      required_resource_units:
        compute: 1
```

Placing spatial domains together

In some scenarios, you might want to place partitions for a spatial domain on workers next to partitions from another domain. This can improve performance characteristics if those partitions create cross-domain subscriptions to each other.

Add the top level key `placement_constraints` to your schema to specify which domains SimSpace Weaver should place together. The required `on_workers` key must refer to a named `workers` configuration in the schema.

Example schema snippet showing spatial domains placed together

```
workers:
  MyComputeWorkers:
    type: "sim.c5.24xlarge"
    desired: 2
placement_constraints:
  - placed_together: ["MySpatialDomain", "MySecondSpatialDomain"]
    on_workers: ["MyComputeWorkers"]
```

Important

- If you use placement groups:
  - Make sure that x * y is a multiple of the number of workers.
  - Make sure that the placement group values are common divisors for the grid dimensions of the domains you place together.
- If you **don't use** placement groups:
  - Make sure that 1 axis of your spatial domain grids has a common divisor that is equal to the number of workers.

For more information about placement groups, see Partitioning strategies.

Frequently asked questions about spatial domains

Q1. How can I add another spatial domain to an existing simulation?

- **For a running simulation** – You can't change the configuration for a running simulation. Change the domain configuration in the schema, upload the schema and app zips, and start a new simulation.
- **For a new simulation** – Add the domain configuration to the schema, upload the schema and app zips, and start the new simulation.

Troubleshooting spatial domains

The might get the following error when you try to start your simulation but your domain configuration is invalid.

```
"StartError": "[{\"errorType\":\"SchemaFormatInvalid\",\"errorMessage\":
    \"We were unable to determine an arrangement of your domains that would fit
    within the provided set of workers. This can generally be resolved by
    increasing the number of workers if able, decreasing your domains\u0027
    [\u0027\u0027grid_partition\u0027\u0027] values, or adjusting the
    dimensions of your [\u0027\u0027grid_placement_groups\u0027\u0027].\"}]"
```

Potential causes

- The schema allocates more compute resource units for apps than are available on workers.
- SimSpace Weaver can't determine an arrangement to place domains together on workers. This happens when you specify multiple spatial domains but there isn't a common divisor or multiple between domain grids, such as between a 2x4 grid and a 3x5 grid).

Network endpoints

Custom and service apps can have network endpoints that external clients can connect to. You specify a list of port numbers as the value for `ingress_ports` in the `endpoint_config`. These port numbers are both TCP and UDP. The custom or service app should bind to the port numbers that you specify in `ingress_ports`. SimSpace Weaver dynamically allocates port numbers at runtime and maps these ports to the dynamic ports. You can call the **describe-app** API after your apps have started to find the dynamic (actual) port numbers. For more information, see Get the IP address and port number of a custom app from the quick start tutorial.

```
domains:
  MyViewDomain:
    launch_apps_via_start_app_call: {}
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MyViewApp.zip"
      launch_command: ["MyViewApp"]
      required_resource_units:
        compute: 1
      endpoint_config:
        ingress_ports:
          - 7000
```

Note

SimSpace Weaver app SDK version 1.12.x projects use separate buckets for the app .zip files and the schema:

- weaver-*lowercase-project-name-account-number*-app-zips-*region*
- weaver-*lowercase-project-name-account-number*-schemas-*region*

Note

`endpoint_config` is an optional property for custom apps and service apps. If you don't specify an `endpoint_config` then the app won't have a network endpoint.

Configuring service domains

The presence of `launch_apps_per_worker:` in a domain configuration indicates that it is a service domain that has service apps. SimSpace Weaver starts and stops service apps for you. When SimSpace Weaver starts and stops an app, the app is considered to have a *managed lifecycle* . SimSpace Weaver currently supports starting 1 or 2 service apps on each and every worker.

Example Example of a domain configured to launch 1 service app on each worker

```
domains:
  MyServiceDomain:
    launch_apps_per_worker:
      count: 1
    app_config:
      package: "s3://example-bucket/PlayerConnectionServiceApp.zip"
      launch_command: ["PlayerConnectionServiceApp"]
      required_resource_units:
        compute: 1
      endpoint_config:
        ingress_ports:
          - 9000
          - 9001
```

Example Example of a domain configured to launch 2 service apps on each worker

```
domains:
  MyServiceDomain:
    launch_apps_per_worker:
      count: 2
    app_config:
      package: "s3://example-bucket/PlayerConnectionServiceApp.zip"
      launch_command: ["PlayerConnectionServiceApp"]
      required_resource_units:
        compute: 1
      endpoint_config:
        ingress_ports:
          - 9000
          - 9001
```

Maximum duration of a simulation

Each simulation in AWS SimSpace Weaver has a *maximum duration* setting that specifies the maximum time that the simulation can run. You provide the maximum duration as a parameter when you start a simulation. The `StartSimulation` application programming interface (API) has an optional parameter `MaximumDuration`. The value of the parameter is a number of minutes (m or M), hours (h or H), or days (d or D). For example, 1h or 1H means 1 hour. SimSpace Weaver stops your simulation when it reaches this limit.

Maximum value

The highest valid value for `MaximumDuration` is `14D`, or its equivalent in hours (`336H`) or minutes (`20160M`).

Default value

The `MaximumDuration` parameter is optional. If you don't provide a value, SimSpace Weaver uses a value of `14D`.

Minimum value

The lowest valid value for `MaximumDuration` is a value that is numerically equivalent to `0`. For example, the values `0M`, `0H`, and `0D`, are all numerically equivalent to `0`.

If you provide the minimum value for maximum duration, your simulation immediately transitions to the `STOPPING` state as soon as it reaches the `STARTED` state.

Starting a simulation using the console

You can provide a value for the **Maximum duration** when you start a simulation in the SimSpace Weaver console. Enter the value in the **Maximum duration** field of the **Simulation settings** form when you choose **Start simulation**.

Important

If you don't provide a value for **Maximum duration**, SimSpace Weaver uses the default value (`14D`).

The status of a simulation that reaches its maximum duration

When SimSpace Weaver automatically stops a simulation that reaches its maximum duration, the **status** of the simulation is `STOPPING` (if in progress) or `STOPPED`. In the SimSpace Weaver console, the **target status** of the simulation is still `STARTED`, because that was the last state requested by a user.

Developing apps

SimSpace Weaver development requires an Amazon Linux 2 (AL2) environment to build apps because your simulations run on Amazon Linux in the AWS Cloud. If you're using Windows, you can use scripts in the SimSpace Weaver app SDK to create and launch a Docker container that runs AL2 with the dependencies that you need to build SimSpace Weaver apps. You can also launch an AL2 environment using Windows Subsystem for Linux (WSL), or use a native AL2 system. For more information, see Set up your local environment for SimSpace Weaver.

Note

Regardless of how you configure your local development environment, your apps run in Docker containers when you upload them to run in the AWS Cloud. **Your apps don't have direct access to the host operating system**.

General flow of a SimSpace Weaver app

1. Create an application.
2. Loop:
    a. Begin the update by creating a `Transaction`.
        - Exit the loop if the simulation is shutting down.
    b. Process subscription and ownership entity events.
    c. Update the simulation.
    d. Commit the `Transaction` to end the update.
3. Destroy the application.

Spatial apps

Each spatial app has an ownership area that is a spatial region of the simulation world. Entities located in a spatial app's ownership area are stored in the app's assigned partition. The single spatial app has full ownership (read and write permissions) over all entities within its assigned partition. No other apps can write to those entities. The spatial app advances the state of its entities. Each spatial app owns only 1 partition. SimSpace Weaver uses the spatial location of an entity to index and assign it to a spatial app partition.

The SimSpace Weaver app SDK provides a sample application. You can find the source code for the spatial app of the sample application in the following folder (use the correct path separator for your operating system):

```
sdk-folder\Samples\PathfindingSample\src\SpatialApp
```

Custom apps

You create and use custom apps to interact with the simulation.

Custom apps can

- Create entities
- Subscribe to other partitions
- Commit changes

General flow of a custom app

1. Create an application.
2. Subscribe to a specific region in the simulation:
   a. Create a `Transaction` to begin the first update.
   b. Create a subscription for the specific region.
   c. Commit the `Transaction` to end the first update.
3. Loop:
   a. Create a `Transaction` to begin the update.
      - Exit the loop if the simulation is shutting down.
   b. Process state changes.
   c. Commit the `Transaction` to end the update.
4. Destroy the application.

After a custom app creates an entity, it must transfer the entity to a spatial domain in order for the entity to exist spatially within the simulation. SimSpace Weaver uses the entity's spatial location to place the entity in the appropriate spatial app partition. The custom app that created the entity can't update or delete the entity after transferring it to a spatial domain.

The SimSpace Weaver app SDK provides a sample application. You can use the custom apps included in the sample application as models for your own custom apps. You can find the source code for the view app (a custom app) of the sample application in the following folder (use the correct path separator for your operating system):

```
sdk-folder\Samples\PathfindingSample\src\ViewApp
```

Developing client applications

Some of the reasons you might want to connect a client to a simulation include:

- Inject real-time traffic information into a city-scale simulation.
- Create *human-in-the-loop* simulations, where a human operator controls some aspect of the simulation.
- Make it possible for users to interact with the simulation, such as for a training simulation.

The custom apps in these examples act as the interface between the simulation state and the outside world. Clients connect to the custom apps to interact with the simulation.

SimSpace Weaver doesn't handle the client applications and their communication with your custom apps. You're responsible for the design, creation, operation, and security of your client applications and their communication with your custom apps. SimSpace Weaver only exposes an IP address and port number for each of your custom apps so that clients can connect to them.

The SimSpace Weaver app SDK provides clients for its sample application. You can use these clients as models for your own client applications. You can find the source code for the sample application clients in the following folder:

Docker

```
sdk-folder\packaging-tools\clients\PathfindingSampleClients
```

WSL

Important

We provide these instructions for your convenience. They are for use with Windows Subsystem for Linux (WSL), and are unsupported. For more information, see Set up your local environment for SimSpace Weaver.

```
sdk-folder/packaging-tools/clients/PathfindingSampleClients
```

For more information about building and using the sample application clients, see the tutorials in Getting started with SimSpace Weaver.

Get the IP address and port number of a custom app

To view your simulation, you create a custom app and connect to it with a client. For more information, see the tutorials in Getting started with SimSpace Weaver. You can use the following procedure to get the IP address and port number of your custom app. Use the appropriate path separator for your operating system (for example, \ in Windows and / in Linux).

To get your IP address and port number

1. Use the **ListSimulations** API to get the name of your simulation.

```
aws simspaceweaver list-simulations
```

Example output:

```


{
    "Simulations": [
        {
            "Status": "STARTED",
            "CreationTime": 1664921418.09,
            "Name": "MyProjectSimulation_22-10-04_22_10_15",
            "Arn": "arn:aws:simspaceweaver:us-west-2: 111122223333:simulation/
MyProjectSimulation_22-10-04_22_10_15",
            "TargetStatus": "STARTED"
        }
    ]

}
```

2. Use the **DescribeSimulation** API to get a list of domains in your simulation.

```
aws simspaceweaver describe-simulation --simulation simulation-name
```

Look for the `Domains` section in the `LiveSimulationState` section of the output.

Example output:

```


    "LiveSimulationState": {
        "Domains": [
            {
                "Type": "",
                "Name": "MySpatialSimulation",
                "Lifecycle": "Unknown"
            },
            {
                "Type": "",
                "Name": "MyViewDomain",
                "Lifecycle": "ByRequest"
            }
        ],
```

3. Use the **ListApps** API to get a list of custom apps in a domain. For example, the domain name for the view (custom) app in the sample project is `MyViewDomain`. Look for the app name in the output.

```
aws simspaceweaver list-apps --simulation simulation-name --domain domain-name
```

Example output:

```
{
    "Apps": [
        {
            "Status": "STARTED",
            "Domain": "MyViewDomain",
            "TargetStatus": "STARTED",
            "Name": "ViewApp",
            "Simulation": "MyProjectSimulation_22-10-04_22_10_15"
        }
    ]
}
```

4. Use the **DescribeApp** API to get the IP address and port number. For the sample project, the domain name is `MyViewDomain` and the app name is `ViewApp`.

```
aws simspaceweaver describe-app --simulation simulation-name --domain domain-name
 --app app-name
```

The IP address and port number are in the `EndpointInfo` block in the output. The IP address is the value of `Address` and the port number is the value of `Actual`.

Example output:

```
{
    "Status": "STARTED",
    "Domain": "MyViewDomain",
    "TargetStatus": "STARTED",
    "Simulation": "MyProjectSimulation_22-10-04_22_10_15",
    "LaunchOverrides": {
        "LaunchCommands": []
    },
    "EndpointInfo": {
        "IngressPortMappings": [
            {
                "Declared": 7000,
                "Actual": 4321
            }
        ],
        "Address": "198.51.100.135"
    },
    "Name": "ViewApp"
}
```

Note

The value of `Declared` is the port number that your app code should bind to. The value of `Actual` is the port number that SimSpace Weaver exposes to clients to connect to your app. SimSpace Weaver maps the `Declared` port to the `Actual` port.

Launching the Unreal Engine view client

Navigate to:

```
sdk-folder/Samples/PathfindingSample/tools/cloud
```

1. Run one of the following commands:
   - Docker: `python quick-start.py`
   - WSL: `python quick-start.py --al2`
2. Get the IP address and "Actual" port number. These will be in the console output from running quick-start.py, or get them by following the procedures at Get the IP address and port number of a custom app.
3. Navigate to:

   ```
   sdk-folder/Clients/TCP/UnrealClient/lib
   ```
4. Run the following commands to build the NNG library:

   ```
   cmake -S . -B build
   cmake --build build --config RelWithDebInfo
   cmake --install build
   ```
5. In a **text editor**, open `view_app_url.txt`.

6. Update the URL with the IP address and port number for your view app: `tcp://ip-address:actual-port-number` (it should look like `tcp://198.51.100.135:1234`).
7. In **Unreal editor**, choose **play**.

Troubleshooting

- **NNG CMake install step fails with “Maybe need administrative privileges“:**

```
CMake Error at build/_deps/nng-build/src/cmake_install.cmake:39 (file):
  file cannot create directory: C:/Program Files
  (x86)/ThirdPartyNngBuild/lib.  Maybe need administrative privileges.
Call Stack (most recent call first):
  build/_deps/nng-build/cmake_install.cmake:37 (include)
  build/cmake_install.cmake:73 (include)
```

  - **Resolution:** If `nng.lib` or `nng.so` exist in the UnrealClient/lib directory, this error can be safely ignored. If not, then try running the cmake build commands in a terminal with administrator privileges.
- **“CMake to find a package configuration file provided by nng”:**

```
CMake Error at CMakeLists.txt:23 (find_package):
By not providing "Findnng.cmake" in CMAKE_MODULE_PATH this project has
 asked CMake to find a package configuration file provided by "nng", but
 CMake did not find one.
```

  - **Resolution:** CMake is having trouble finding the `Findnng.cmake` file. When building with CMake, add the argument `-DTHIRD_PARTY_LIB_PATH sdk-folder/ThirdParty`. Ensure the `Findnng.cmake` file is still in the `ThirdParty` directory before rerunning the CMake build.

```
cmake -S . -B build -DTHIRD_PARTY_LIB_PATH sdk-folder/ThirdParty
cmake --build build --config RelWithDebInfo
cmake --install build
```

Local development in SimSpace Weaver

You can deploy your SimSpace Weaver applications locally for rapid testing and debugging.

Requirements

- Complete the steps in Setting up for SimSpace Weaver.

Topics

Step 1: Launch your local simulation

1. Navigate to

   ```
   cd sdk-folder/Samples/sample-name/tools/local
   ```

2. Run the following command to build and launch your simulation locally.

   ```
   python quick-start.py
   ```

   This script will do the following:

   1. Build the project.
      - `quick-start.py` calls the `build_project` function defined in build.py. This step will vary depending on the project. For the PathfindingSample, CMake is used. The CMake and Docker command for which can be found in build.py.
   2. Launch your local simulation
      - The script will launch one local process for each spatial partition defined in the schema.
      - The script will launch one process for each custom app defined in the schema.
      - The spatial apps will be launched first, followed by the custom apps — each in the order they appear in the schema.

Important

When launching in an environment that does not support GUI, such as a console SSH session, use the `--noappwindow` option to redirect all output to the current terminal.

Important

For Linux users, the script assumes your system has the `xterm` command. If your Linux distribution does not have the `xterm` command, use the `--noappwindow` option to redirect all output to the current terminal.

- -h, --help
  - List these parameters.
- --clean
  - Delete the contents of the build directory before building.
- --nobuild
  - Skip rebuilding the project.
- --noappwindow
  - Don't open a new window for each app. Instead, redirect the stdout to the current terminal.
- --logfile
  - Write console output to a logs file.
- --consoleclient
  - Automatically connect the console client listed in the config.
- --schema SCHEMA
  - What schema this invocation will use. Defaults to 'SCHEMA' in config.py.

Step 2: View your local simulation

To view your local simulation, you can use any of the clients that are included with the SimSpaceWeaverAppSdkDistributable. For more information on building and using the sample clients, see the tutorials in Getting started with SimSpace Weaver.

You must update the IP address and port number in the client to connect to the view app for your local simulation. Always use the following values with SimSpace Weaver Local:

```
tcp://127.0.0.1:7000
```

Depending on the client you select, you can update the IP address and port number as follows:

- **Unreal** – Change the URL on line 1 of `view_app_url.txt`
- **Console** – Launch the client with the IP address and port number URL as a parameter

Step 3: Stop your local simulation (optional on Windows)

Note

This step is required on Linux but optional on Windows.

1. Navigate to:

   ```
   sdk-folder/Samples/sample-name/tools/local
   ```

2. Run the following command to stop your local simulation and delete any shared memory resources.

   ```
   python stop-and-delete.py
   ```

   This script will do the following:

   - Stop the local processes.
   - Delete the shared memory object (only needed on Linux).

stop-and-delete.py parameters

- -h, --help
  - List these parameters.
- --stop
  - Only attempt to stop the processes.

- --delete
  - Only attempt to delete the shared memory resources.
- --process
  - The name of the process to stop. Use this if your process name doesn't match the package name in the schema.
- --schema SCHEMA
  - What schema this invocation will use. Defaults to value of 'SCHEMA' in config.py.

Troubleshooting local development in SimSpace Weaver

- **Linux: xterm command not found / cannot open**
  - The local scripts assume the xterm command exists when running on Linux. If you do not have the xterm command or are running in an environment that does not support GUI, use the `--noappwindow` option when running the quick-start script.
- **No app windows are opening!**
  - This happens when the local simulation crashes immediately. To see the console output after the crash, use the `--noappwindow` or `--logfile` options when running the quick-start script.
- **The simulation isn’t ticking after the view app starts or view client connects!**
  - Running with the –`noappwindow` option typically resolves these kinds of issues. Otherwise, restarting a few times also has success (although at a much lower rate).

AWS SimSpace Weaver app SDK

The SimSpace Weaver app SDK provides APIs that you can use to control the entities in your simulation and respond to SimSpace Weaver events. It includes the following namespace:

- **API –** core definitions of the API and its use

Link with the following library:

- `libweaver_app_sdk_cxx_v1_full.so`

Important

The library is available for dynamic linking when you run your apps in the AWS Cloud. You don't need to upload it with your apps.

Note

The SimSpace Weaver app SDK APIs control data within your simulation. These APIs are separate from the SimSpace Weaver service APIs, which control your SimSpace Weaver service resources (such as simulations, apps, and clocks) in AWS. For more information, see SimSpace Weaver API references.

Topics

API methods return a Result

The majority of SimSpace Weaver API functions have a return type `Aws::WeaverRuntime::Result<T>`. If the function has executed successfully, the `Result` contains T. Otherwise, the `Result` contains an `Aws::WeaverRuntime::ErrorCode` that represents an error code from the Rust App SDK.

Example Example

```
Result<Transaction> BeginUpdate(Application& app)
```

This method:

- Returns `Transaction` if `BeginUpdate()` executes successfully.
- Returns `Aws::WeaverRuntime::ErrorCode` if `BeginUpdate()` fails.

Interacting with the app SDK at the top level

Lifecycle

- The SimSpace Weaver app SDK manages app life cycle. You don't need to read or write the lifecycle state of an app.

Partitions

- Use `Result <PartitionSet> AssignedPartitions(Transaction& txn);` to get owned partitions.
- Use `Result <PartitionSet> AllPartitions(Transaction& txn);` to get all partitions in the simulation.

Simulation management

This section describes solutions for common simulation management tasks.

Topics

- Start a simulation
- Update a simulation
- Terminate a simulation

Start a simulation

Use `CreateApplication()` to create an app.

Example Example

```
Result<Application> applicationResult = Api::CreateApplication();
```

```
if (!applicationResult)
{
    ErrorCode errorCode = WEAVERRUNTIME_EXPECT_ERROR(applicationResult);

    std::cout << "Failed to create application. Error code " <<
        static_cast<std::underlying_type_t<ErrorCode>>(errorCode) <<
        " Last error message "<< Api::LastErrorMessage() << ".";

    return 1;
}

/**
* Run simulation
*/
RunSimulation(std::move(applicationResult.assume_value()));
```

Update a simulation

Use the following `BeginUpdate` functions to update the app:

- `Result<Transaction> BeginUpdate(Application& app)`
- `Result<bool> BeginUpdateWillBlock(Application& app)` – tells you if `BeginUpdate()` will block or not block.

Use `Result<void> Commit(Transaction& txn)` to commit the changes.

Example Example

```
Result<void> AppDriver::RunSimulation(Api::Application app) noexcept
{
    while (true)
    {
        {
            bool willBlock;

            do
            {
                WEAVERRUNTIME_TRY(willBlock, Api::BeginUpdateWillBlock(m_app));
            } while (willBlock);
        }
        WEAVERRUNTIME_TRY(Transaction transaction, Api::BeginUpdate(app));

        /**
         * Simulate app.
         */
        WEAVERRUNTIME_TRY(Simulate(transaction));
        WEAVERRUNTIME_TRY(Api::Commit(std::move(transaction)));
    }

    return Success();
}
```

Terminate a simulation

Use `Result<void> DestroyApplication(Application&& app)` to terminate the app and the simulation.

Other apps find out that the simulation is shutting down when they receive `ErrorCode::ShuttingDown` from their calls to `BeginUpdateWillBlock()` or `BeginUpdate()`. When an app receives `ErrorCode::ShuttingDown`, it can call `Result<void> DestroyApplication(Application&& app)` to terminate itself.

Example Example

```
Result<void> AppDriver::EncounteredAppError(Application&& application) noexcept
{
    const ErrorCode errorCode = WEAVERRUNTIME_EXPECT_ERROR(runAppResult);

    switch (errorCode)
    {
    case ErrorCode::ShuttingDown:
        {
            // insert custom shutdown process here.

            WEAVERRUNTIME_TRY(Api::DestroyApplication(std::move(application)));
            return Success();
        }
    default:
        {
            OnAppError(errorCode);
            return errorCode;
        }
    }
}
```

Important

Only call `Result<void> DestroyApplication(Application&& app)` after `Api::Commit()`. Destroying an application during an update can cause undefined behavior.

Important

You must call `DestroyApplication()` before the program exits to make sure that the application reports as terminating successfully.
Failure to call `DestroyApplication()` when the program exits will cause the status to be considered as `FATAL`.

Subscriptions

You create a subscription with a subscription area and a domain ID. The domain ID represents the domain that owns that subscription area. A `BoundingBox2F32` describes the subscription area. Use the following function to create a subscription:

```
Result<SubscriptionHandle> CreateSubscriptionBoundingBox2F32(Transaction& txn, DomainId
 id, const BoundingBox2F32& boundingBox)
```

Example Example

```
Result<void> CreateSubscriptionInSpatialDomain(Transaction& transaction)
{
    WEAVERRUNTIME_TRY(Api::PartitionSet partitionSet, Api::AllPartitions(transaction));


    Api::DomainId spatialDomainId;

    for (const Api::Partition& partition : partitionSet.partitions)
    {
        if (partition.domain_type == Api::DomainType::Spatial)
        {
            /**
            * Get the spatial domain ID.
            */
            spatialDomainId = partition.domain_id;
            break;
        }
    }

    constexpr Api::BoundingBox2F32 subscriptionBounds {
        /* min */ { /* x */ 0, /* y */ 0 },
        /* max */ { /* x */ 1000, /* y */ 1000 } }

    WEAVERRUNTIME_TRY(
        Api::SubscriptionHandle subscriptionHandle,
        Api::CreateSubscriptionBoundingBox2F32(
        transaction,
        spatialDomainId,
        subscriptionBounds));

    return Success();
}
```

You can use the `Api::SubscriptionHandle` returned by `CreateSubscriptionBoundingBox2F32()` to modify the subscription. You pass it as an argument to the following functions:

```
Result<void> ModifySubscriptionBoundingBox2F32(Transaction& txn, SubscriptionHandle
 handle, const BoundingBox2F32& boundingBox)
```

```
Result<void> DeleteSubscription(Transaction& txn, SubscriptionHandle handle)
```

Entities

You call the `Store` and `Load` APIs using the `Api:Entity` of the `Result<Api::Entity>` returned from `CreateEntity()`, or from an ownership change event when an entity enters the app's subscription area (for more information, see Entity events). We recommend that you track your `Api::Entity` objects so that you can use them with these APIs.

Topics

- Create entities

- Transfer an entity to a spatial domain
- Write and read entity field data
- Store the position of an entity
- Load the position of an entity

Create entities

Use `CreateEntity()` to create an entity. You define the meaning of the `Api::TypeId` that you pass to this function.

```
Namespace
{
    constexpr Api::TypeId k_entityTypeId { /* value */ 512 };
}

Result<void> CreateEntity(Transaction& transaction)
{
    WEAVERRUNTIME_TRY(
        Api::Entity entity,
        Api::CreateEntity(
            transaction, Api::BuiltinTypeIdToTypeId(k_entityTypeId )));
}
```

Note

The values 0-511 for `Api::BuiltinTypeId` are reserved. Your entity TypeID (`k_entityTypeId` in this example) must have a value of 512 or higher.

Transfer an entity to a spatial domain

After a custom app or service app creates an entity, the app must transfer the entity to a spatial domain in order for the entity to exist spatially in the simulation. Entities in a spatial domain can be read by other apps and updated by a spatial app. Use the `ModifyEntityDomain()` API to transfer an entity to a spatial domain.

```
AWS_WEAVERRUNTIME_API Result<void> ModifyEntityDomain(Transaction& txn, const Entity&
 entity, DomainId domainId) noexcept;
```

If the `DomainId` doesn't match the assigned `Partition` of the calling app, then the `DomainId` must be for a `DomainType::Spatial` `Domain`. The ownership transfer to the new `Domain` occurs during the `Commit(Transaction&&)`.

Parameters

`txn`

The current `Transaction`.

`entity`

The target `Entity` for the change of `Domain`.

`domainId`

The `DomainId` of the destination `Domain` for the `Entity`.

This API returns `Success` if the entity domain was successfully changed.

Write and read entity field data

All entity data fields are blob types. You can write up to 1,024 bytes of data to an entity. We recommend that you keep blobs as small as possible because larger sizes will reduce performance. When you write to a blob, you pass SimSpace Weaver a pointer to the data and its length. When you read from a blob, SimSpace Weaver provides you with a pointer and a length to read. All reads must be complete before the app calls `Commit()`. Pointers returned from a read call are invalidated when the app calls `Commit()`.

Important

- Reading from a cached blob pointer after a `Commit()` is unsupported and can cause the simulation to fail.
- Writing to a blob pointer returned from a read call is unsupported and can cause the simulation to fail.

Topics

- Store the field data of an entity
- Load the field data of an entity

- Loading the field data of removed entities

Store the field data of an entity

The following examples demonstrate how you can store (write to the state fabric) the field data of an entity that the app owns. These examples use the following function:

```
AWS_WEAVERRUNTIME_API Result<void> StoreEntityField(
    Transaction& txn,
    const Entity& entity,
    TypeId keyTypeId,
    FieldIndex index,
    std::int8_t* src,
    std::size_t length) noexcept;
```

The `Api::TypeId keyTypeId` parameter represents the data type of the passed in data.

The `Api::TypeId keyTypeId` parameter should receive the corresponding `Api::TypeId` from `Api::BuiltinTypeId`. If there is no appropriate conversion, you can use `Api::BuiltinTypeId::Dynamic`.

For complex data types, use `Api::BuiltInTypeId::Dynamic`.

Note

The value of `FieldIndex index` must be greater than 0. The value 0 is reserved for the index key (see `StoreEntityIndexKey()`).

Example Example using primitive data types

```
namespace
{
    constexpr Api::FieldIndex k_isTrueFieldId { /* value */ 1 };
}

Result<void> SetEntityFields(
    Api::Entity& entity,
    Transaction& transaction)
{
    bool value = true;

    auto* src = reinterpret_cast<std::int8_t*>(value);
    size_t length = sizeof(*value);

    WEAVERRUNTIME_TRY(Api::StoreEntityField(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(
            Aws::WeaverRuntime::Api::BuiltinTypeId::Bool),
        k_isTrueFieldId,
        src,
        length));
}
```

Example Example using a struct to hold the data

```
namespace
{
    constexpr Api::FieldIndex k_dataFieldId { /* value */ 1 };
}

struct Data
{
    bool boolData;
    float floatData;
};

Result<void> SetEntityFields(
    Api::Entity& entity,
    Transaction& transaction)
{
    Data data = { /* boolData */ false, /* floatData */ -25.93 };

    auto* src = reinterpret_cast<std::int8_t*>(data);
    size_t length = sizeof(*data);

    WEAVERRUNTIME_TRY(Api::StoreEntityField(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(
            Aws::WeaverRuntime::Api::BuiltinTypeId::Dynamic),
        k_dataFieldId,
        src,
        length));
}
```

Load the field data of an entity

The following examples demonstrate how you can load (read from the state fabric) the field data of an entity. These examples use the following function:

```
Result<std::size_t> LoadEntityField(
    Transaction& txn,
    const Entity& entity,
    TypeId keyTypeId,
    FieldIndex index,
    std::int8_t** dest) noexcept;
```

The `Api::TypeId keyTypeId` parameter should receive the corresponding `Api::TypeId` from `Api::BuiltinTypeId`. If there is no appropriate conversion, you can use `Api::BuiltinTypeId::Dynamic`.

Note

The value of `FieldIndex` index must be greater than 0. The value 0 is reserved for the index key (see `StoreEntityIndexKey()`).

Example Example using primitive data types

```
namespace
{
    constexpr Api::FieldIndex k_isTrueFieldId { /* value */ 1 };
}

Result<void> LoadEntityFields(
    Api::Entity& entity,
    Transaction& transaction)
{
    std::int8_t* dest = nullptr;

    WEAVERRUNTIME_TRY(Api::LoadEntityField(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(
            Aws::WeaverRuntime::Api::BuiltinTypeId::Bool),
        k_isTrueFieldId,
        &dest));

    bool isTrueValue = *reinterpret_cast<bool*>(dest);
}
```

Example Example using a struct to hold the data

```
namespace
{
    constexpr Api::FieldIndex k_dataFieldId { /* value */ 1 };
}

struct Data
{
    bool boolData;
    float floatData;
};

Result<void> LoadEntityFields(
    Api::Entity& entity,
    Transaction& transaction)
{
    std::int8_t* dest = nullptr;

    WEAVERRUNTIME_TRY(Api::LoadEntityField(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(
            Aws::WeaverRuntime::Api::BuiltinTypeId::Dynamic),
        k_dataFieldId,
        &dest));

    Data dataValue = *reinterpret_cast<Data*>(dest);
}
```

Loading the field data of removed entities

You can't load (read from the state fabric) entity field data for entities that have been removed from the app's ownership and subscription areas. The following example results in an error because it calls `Api::LoadIndexKey()` on an entity as a result of an

`Api::ChangeListAction::Remove`. The second example shows a correct way to store and load entity data directly in the app.

Example Example of incorrect code

```
Result<void> ProcessSubscriptionChanges(Transaction& transaction)
{
    /* ... */

    WEAVERRUNTIME_TRY(Api::SubscriptionChangeList subscriptionChangeList,
        Api::AllSubscriptionEvents(transaction));

    for (const Api::SubscriptionEvent& event :
        subscriptionChangeList.changes)
    {
        switch (event.action)
        {
        case Api::ChangeListAction::Remove:
            {
                std::int8_t* dest = nullptr;

                /**
                 * Error!
                 * This calls LoadEntityIndexKey on an entity that
                 * has been removed from the subscription area.
                 */
                WEAVERRUNTIME_TRY(Api::LoadEntityIndexKey(
                    transaction,
                    event.entity,
                    Api::BuiltinTypeIdToTypeId(
                        Api::BuiltinTypeId::Vector3F32),
                    &dest));

                AZ::Vector3 position =
                    *reinterpret_cast<AZ::Vector3*>(dest);
                break;
            }
        }

    }

    /* ... */
}
```

Example Example of a correct way to store and load entity data in the app

```
Result<void> ReadAndSaveSubscribedEntityPositions(Transaction& transaction)
{
    static std::unordered_map<Api::EntityId, AZ::Vector3>
        positionsBySubscribedEntity;

    WEAVERRUNTIME_TRY(Api::SubscriptionChangeList subscriptionChangeList,
        Api::AllSubscriptionEvents(transaction));

    for (const Api::SubscriptionEvent& event :
        subscriptionChangeList.changes)
    {
        switch (event.action)
        {
        case Api::ChangeListAction::Add:
            {
                std::int8_t* dest = nullptr;

                /**
                 * Add the position when the entity is added.
                 */
                WEAVERRUNTIME_TRY(Api::LoadEntityIndexKey(
                    transaction,
                    event.entity,
                    Api::BuiltinTypeIdToTypeId(
                        Api::BuiltinTypeId::Vector3F32),
                    &dest));

                AZ::Vector3 position =
                    *reinterpret_cast<AZ::Vector3*>(dest);
                positionsBySubscribedEntity.emplace(
                    event.entity.descriptor->id, position);

                break;
            }
        case Api::ChangeListAction::Update:
            {
                std::int8_t* dest = nullptr;

                /**
                 * Update the position when the entity is updated.
                 */
                WEAVERRUNTIME_TRY(Api::LoadEntityIndexKey(
                    transaction,
                    event.entity,
                    Api::BuiltinTypeIdToTypeId(
                        Api::BuiltinTypeId::Vector3F32),
                    &dest));

                AZ::Vector3 position =
                    *reinterpret_cast<AZ::Vector3*>(dest);
                positionsBySubscribedEntity[event.entity.descriptor->id] =
                    position;

                break;
            }
        case Api::ChangeListAction::Remove:
            {
                /**
                 * Load the position when the entity is removed.
                 */
                AZ::Vector3 position = positionsBySubscribedEntity[
                    event.entity.descriptor->id];

                /**
                 * Do something with position...
                 */
                break;
            }
        }
    }

    /* ... */
}
```

Store the position of an entity

You can store (write to the state fabric) the position of an entity using an integer data structure. These examples use the following function:

```
Result<void> StoreEntityIndexKey(
    Transaction& txn,
    const Entity& entity,
    TypeId keyTypeId,
    std::int8_t* src,
    std::size_t length)
```

Note

You must provide `Api::BuiltinTypeId::Vector3F32` to `Api::StoreEntityIndexKey()`, as shown in the following examples.

Example Example using an array to represent the position

```
Result<void> SetEntityPositionByFloatArray(
    Api::Entity& entity,
    Transaction& transaction)
{
    std::array<float, 3> position = { /* x */ 25, /* y */ 21, /* z */ 0 };

    auto* src = reinterpret_cast<std::int8_t*>(position.data());
    std::size_t length = sizeof(position);

    WEAVERRUNTIME_TRY(Api::StoreEntityIndexKey(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(Api::BuiltinTypeId::Vector3F32),
        src,
        length));
}
```

Example Example using a struct to represent the position

```
struct Position
{
   float x;
   float y;
   float z;
};

Result<void> SetEntityPositionByStruct(
    Api::Entity& entity,
    Transaction& transaction)
{
    Position position = { /* x */ 25, /* y */ 21, /* z */ 0 };

    auto* src = reinterpret_cast<std::int8_t*>(&position);
    std::size_t length = sizeof(position);

    WEAVERRUNTIME_TRY(Api::StoreEntityIndexKey(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(Api::BuiltinTypeId::Vector3F32),
        src,
        length));
}
```

Load the position of an entity

You can load (read from the state fabric) the position of an entity using an integer data structure. These examples use the following function:

Note

You must provide `Api::BuiltinTypeId::Vector3F32` to `Api::LoadEntityIndexKey()`, as shown in the following examples.

Example Example using an array to represent the position

```
Result<void> GetEntityPosition(Api::Entity& entity,
    Transaction& transaction)
{
    std::int8_t* dest = nullptr;

    WEAVERRUNTIME_TRY(Aws::WeaverRuntime::Api::LoadEntityIndexKey(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(
            Aws::WeaverRuntime::Api::BuiltinTypeId::Vector3F32),
        &dest));

    std::array<float, 3> position =
        *reinterpret_cast<std::array<float, 3>*>(dest);
}
```

Example Example using a struct to represent the position

```
struct Position
```

```
{struct
   float x;
   float y;
   float z;
};

Result<void> GetEntityPosition(Api::Entity& entity, Transaction& transaction)
{
    std::int8_t* dest = nullptr;

    WEAVERRUNTIME_TRY(Aws::WeaverRuntime::Api::LoadEntityIndexKey(
        transaction,
        entity,
        Api::BuiltinTypeIdToTypeId(
            Aws::WeaverRuntime::Api::BuiltinTypeId::Vector3F32),
        &dest));

    Position position = *reinterpret_cast<Position*>(dest);
}
```

Entity events

You can use the following functions in the SimSpace Weaver app SDK to get all ownership and subscription events:

- `Result<OwnershipChangeList> OwnershipChanges(Transaction& txn)`
- `Result<SubscriptionChangeList> AllSubscriptionEvents(Transaction& txn)`

You can use the SimSpace Weaver demo framework if you need callback-driven entity event processing. For more information, see the following header file:

- `*sdk-folder*/packaging-tools/samples/ext/DemoFramework/include/DemoFramework/EntityEventProcessor.h`

You can also create your own entity event processing.

Topics

- Iterate through ownership change events for entities

Iterate through events for owned entities

Use `OwnershipChanges()` to get a list of events for owned entities (entities in the app's ownership area). The function has the following signature:

```
Result<OwnershipChangeList> OwnershipChanges(Transaction& txn)
```

Then iterate through the entities with a loop, as demonstrated in the following example.

Example Example

```
WEAVERRUNTIME_TRY(Result<Api::OwnershipChangeList> ownershipChangesResult,
 Api::OwnershipChanges(transaction));

for (const Api::OwnershipChange& event : ownershipChangeList.changes)
{
    Api::Entity entity = event.entity;
    Api::ChangeListAction action = event.action;

    switch (action)
    {
    case Api::ChangeListAction::None:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Remove:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Add:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Update:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Reject:
        // insert code to handle the event
        break;
    }
}
```

Event types

- `None` – The entity is in the area and its position and field data weren't modified.
- `Remove` – The entity was removed from the area.
- `Add` – The entity was added to the area.
- `Update` – The entity is in the area and was modified.
- `Reject` – The app failed to remove the entity from the area.

Note

In the case of a `Reject` event, the app will attempt the transfer again on the next tick.

Iterate through events for subscribed entities

Use `AllSubscriptionEvents()` to get a list of events for subscribed entities (entities in the app's subscription area). The function has the following signature:

```
Result<SubscriptionChangeList> AllSubscriptionEvents(Transaction& txn)
```

Then iterate through the entities with a loop, as demonstrated in the following example.

Example Example

```
WEAVERRUNTIME_TRY(Api::SubscriptionChangeList subscriptionChangeList,
 Api::AllSubscriptionEvents(transaction));

for (const Api::SubscriptionEvent& event : subscriptionChangeList.changes)
{
    Api::Entity entity = event.entity;
    Api::ChangeListAction action = event.action;

    switch (action)
    {
    case Api::ChangeListAction::None:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Remove:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Add:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Update:
        // insert code to handle the event
        break;
    case Api::ChangeListAction::Reject:
        // insert code to handle the event
        break;
    }
}
```

Event types

- `None` – The entity is in the area and its position and field data weren't modified.
- `Remove` – The entity was removed from the area.
- `Add` – The entity was added to the area.
- `Update` – The entity is in the area and was modified.
- `Reject` – The app failed to remove the entity from the area.

Note

In the case of a `Reject` event, the app will attempt the transfer again on the next tick.

Iterate through ownership change events for entities

To get events where an entity moves between an ownership area and subscription area, compare the changes between the current and previous entity ownership and subscription events.

You can handle these events by reading:

- `Api::SubscriptionChangeList`
- `Api::OwnershipEvents`

You can then compare the changes to previously stored data.

The following example shows how you can handle entity ownership change events. This example assumes that for entities transitioning between being subscribed entities and owned entities (in either direction), the ownership remove/add event occurs first followed by the subscription remove/add event in the next tick.

Example Example

```
Result<void> ProcessOwnershipEvents(Transaction& transaction)
{
    using EntityIdsByAction =
        std::unordered_map<Api::ChangeListAction,
        std::vector<Api::EntityId>>;
    using EntityIdSetByAction =
        std::unordered_map<Api::ChangeListAction,
        std::unordered_set<Api::EntityId>>;

    static EntityIdsByAction m_entityIdsByPreviousOwnershipAction;

    EntityIdSetByAction entityIdSetByAction;

    /**
     * Enumerate Api::SubscriptionChangeList items
     * and store Add and Remove events.
     */
    WEAVERRUNTIME_TRY(Api::SubscriptionChangeList subscriptionEvents,
        Api::AllSubscriptionEvents(transaction));

    for (const Api::SubscriptionEvent& event : subscriptionEvents.changes)
    {
        const Api::ChangeListAction action = event.action;

        switch (action)
        {
        case Api::ChangeListAction::Add:
        case Api::ChangeListAction::Remove:

            {
                entityIdSetByAction[action].insert(
                    event.entity.descriptor->id);
                break;
            }
        case Api::ChangeListAction::None:
        case Api::ChangeListAction::Update:
        case Api::ChangeListAction::Reject:
            {
                break;
            }
        }
    }

    EntityIdsByAction entityIdsByAction;

    /**
     * Enumerate Api::OwnershipChangeList items
     * and store Add and Remove events.
     */

    WEAVERRUNTIME_TRY(Api::OwnershipChangeList ownershipChangeList,
        Api::OwnershipChanges(transaction));

    for (const Api::OwnershipChange& event : ownershipChangeList.changes)
    {
        const Api::ChangeListAction action = event.action;

        switch (action)
        {
        case Api::ChangeListAction::Add:
        case Api::ChangeListAction::Remove:
            {
                entityIdsByAction[action].push_back(
                    event.entity.descriptor->id);
                break;
            }
        case Api::ChangeListAction::None:
        case Api::ChangeListAction::Update:
        case Api::ChangeListAction::Reject:
            {
                break;
            }
        }

    }

    std::vector<Api::EntityId> fromSubscribedToOwnedEntities;
    std::vector<Api::EntityId> fromOwnedToSubscribedEntities;

    /**
     * Enumerate the *previous* Api::OwnershipChangeList Remove items
     * and check if they are now in
     * the *current* Api::SubscriptionChangeList Add items.
     *
     * If true, then that means
     * OnEntityOwnershipChanged(bool isOwned = false)
     */
    for (const Api::EntityId& id : m_entityIdsByPreviousOwnershipAction[
        Api::ChangeListAction::Remove])
    {
        if (entityIdSetBySubscriptionAction[
            Api::ChangeListAction::Add].find(id) !=
                entityIdSetBySubscriptionAction[
                Api::ChangeListAction::Add].end())
        {
            fromOwnedToSubscribedEntities.push_back(id);
        }
    }


    /**
     * Enumerate the *previous* Api::OwnershipChangeList Add items
     * and check if they are now in
     * the *current* Api::SubscriptionChangeList Remove items.
     *
     * If true, then that means
     * OnEntityOwnershipChanged(bool isOwned = true)
     */
    for (const Api::EntityId& id : m_entityIdsByPreviousOwnershipAction[
        Api::ChangeListAction::Add])
    {
        if (entityIdSetBySubscriptionAction[
            Api::ChangeListAction::Remove].find(id) !=

                entityIdSetBySubscriptionAction[
                Api::ChangeListAction::Remove].end())
        {
            fromSubscribedToOwnedEntities.push_back(id);
        }
    }

    m_entityIdsByPreviousOwnershipAction = entityIdsByOwnershipAction;

    return Success();
}
```

Result and error handling

The `Aws::WeaverRuntime::Result<T>` class uses a third-party `Outcome` library. You can use the following pattern to check the `Result` and catch errors returned by API calls.

```
void DoBeginUpdate(Application& app)
{
    Result<Transaction> transactionResult = Api::BeginUpdate(app);

    if (transactionResult)
    {
        Transaction transaction =
            std::move(transactionResult).assume_value();

        /**
         * Do things with transaction ...
         */
    }
    else
    {
        ErrorCode errorCode = WEAVERRUNTIME_EXPECT_ERROR(transactionResult);
        /**
         * Macro compiles to:
         * ErrorCode errorCode = transactionResult.assume_error();
         */
    }
}
```

Result control statement macro

Inside a function with a return type `Aws::WeaverRuntime::Result<T>`, you can use the `WEAVERRUNTIME_TRY` macro instead of the previous code pattern. The macro will execute the function passed to it. If the passed function fails, the macro will make the enclosing function return an error. If the passed function succeeds, execution progresses to the next line. The following example shows a rewrite of the previous `DoBeginUpdate()` function. This version uses the `WEAVERRUNTIME_TRY` macro instead of the if-else control structure. Note that the return type of the function is `Aws::WeaverRuntime::Result<void>`.

```
Aws::WeaverRuntime::Result<void> DoBeginUpdate(Application& app)
{
    /**
```

```
     * Execute Api::BeginUpdate()
     * and return from DoBeginUpdate() if BeginUpdate() fails.
     * The error is available as part of the Result.
     */
    WEAVERRUNTIME_TRY(Transaction transaction, Api::BeginUpdate(m_app));

    /**
     * Api::BeginUpdate executed successfully.
     *
     * Do things here.
     */

    return Aws::Success();
}
```

If `BeginUpdate()` fails, the macro makes `DoBeginUpdate()` return early with a failure. You can use the `WEAVERRUNTIME_EXPECT_ERROR` macro to get the `Aws::WeaverRuntime::ErrorCode` from `BeginUpdate()`. The following example shows how the `Update()` function calls `DoBeginUpdate()` and gets the error code on failure.

```
void Update(Application& app)
{
    Result<void> doBeginUpdateResult = DoBeginUpdate(app);

    if (doBeginUpdateResult)
    {
        /**
         * Successful.
         */
    }
    else
    {
        /**
         * Get the error from Api::BeginUpdate().
         */
        ErrorCode errorCode = WEAVERRUNTIME_EXPECT_ERROR(doBeginUpdateResult);

    }
}
```

You can make the error code from `BeginUpdate()` available to a function that calls `Update()` by changing the return type of `Update()` to `Aws::WeaverRuntime::Result<void>`. You can repeat this process to keep sending the error code further down the call stack.

Generics and domain types

The SimSpace Weaver app SDK provides the single-precision data types `Api::Vector2F32` and `Api::BoundingBox2F32`, and the double-precision `Api::Vector2F64` and `Api::BoundingBox2F64`. These data types are passive data structures with no convenience methods. Note that the API only uses `Api::Vector2F32` and `Api::BoundingBox2F32`. You can use these data types to create and modify subscriptions.

The SimSpace Weaver demo framework provides a minimal version of the AzCore math library, which contains `Vector3` and `Aabb` . For more information, see the header files in:

- `*sdk-folder*/packaging-tools/samples/ext/DemoFramework/include/AzCore/Math`

Miscellaneous app SDK operations

Topics

- AllSubscriptionEvents and OwnershipChanges contain events from the last call
- Release read locks after processing SubscriptionChangeList
- Create a standalone app instance for testing

AllSubscriptionEvents and OwnershipChanges contain events from the last call

The return values of calls to `Api::AllSubscriptionEvents()` and `Api::OwnershipChanges()` contain events from the last call, **not the last tick**. In the following example, `secondSubscriptionEvents` and `secondOwnershipChangeList` are empty because their functions are called immediately after the first calls.

If you wait 10 ticks and then call `Api::AllSubscriptionEvents()` and `Api::OwnershipChanges()`, their results will both contain events and changes from the last 10 ticks (not the last tick).

Example Example

```
Result<void> ProcessOwnershipChanges(Transaction& transaction)
```

```
{
    WEAVERRUNTIME_TRY(
        Api::SubscriptionChangeList firstSubscriptionEvents,
        Api::AllSubscriptionEvents(transaction));
    WEAVERRUNTIME_TRY(
        Api::OwnershipChangeList firstOwnershipChangeList,
        Api::OwnershipChanges(transaction));

    WEAVERRUNTIME_TRY(
        Api::SubscriptionChangeList secondSubscriptionEvents,
        Api::AllSubscriptionEvents(transaction));
    WEAVERRUNTIME_TRY(
        Api::OwnershipChangeList secondOwnershipChangeList,
        Api::OwnershipChanges(transaction));

    /**
     * secondSubscriptionEvents and secondOwnershipChangeList are
     * both empty because there are no changes since the last call.
     */
}
```

Note

The function `AllSubscriptionEvents()` is implemented but the function `SubscriptionEvents()` is **not implemented**.

Release read locks after processing SubscriptionChangeList

When you begin an update, there are shared memory segments for the committed data in other partitions for the previous tick. These shared memory segments might be locked by readers. An app can't fully commit until all the readers have released the locks. As an optimization, an app should call `Api::ReleaseReadLeases()` to release the locks after processing `Api::SubscriptionChangelist` items. This reduces contention at commit time. `Api::Commit()` releases the read leases by default, but it's a best practice to manually release them after processing subscription updates.

Example Example

```
Result<void> ProcessSubscriptionChanges(Transaction& transaction)
{
    WEAVERRUNTIME_TRY(ProcessSubscriptionChanges(transaction));

    /**
     * Done processing Api::SubscriptionChangeList items.
     * Release read locks.
     */

    WEAVERRUNTIME_EXPECT(Api::ReleaseReadLeases(transaction));

    ...
}
```

Create a standalone app instance for testing

You can use `Api::CreateStandaloneApplication()` to create a standalone app to test app logic before running the code in an actual simulation.

Example Example

```
int main(int argc, char* argv[])
{
    Api::StandaloneRuntimeConfig config = {
        /* run_for_seconds (the lifetime of the app) */ 3,
        /* tick_hertz (the app clock rate) */ 10 };

    Result<Application> applicationResult =
        Api::CreateStandaloneApplication(config);

    ...
}
```

AWS SimSpace Weaver demo framework

The AWS SimSpace Weaver demo framework (demo framework) is a library of utilities that you can use to develop SimSpace Weaver apps.

The demo framework provides

- Code samples and programming patterns for you to use and examine
- Abstractions and utility functions that streamline development for simple apps
- A simpler way to test experimental features of the SimSpace Weaver app SDK

We designed the SimSpace Weaver app SDK with low-level access to SimSpace Weaver APIs in order to deliver higher performance. In contrast, we designed the demo framework to provide higher-level abstractions and access to APIs that make SimSpace Weaver easier to use. The cost of ease of use is a lower level of performance compared to directly using the SimSpace Weaver app SDK. Simulations that can tolerate lower performance (such as those without real-time performance requirements) might be good candidates to use the demo framework. We recommend that you use the native functionality in the SimSpace Weaver app SDK for complex applications because the demo framework isn't a complete toolkit.

The demo framework includes

- Working code samples that support and demonstrate:
  - App flow management
  - Callback-driven entity event processing
- A set of third-party utility libraries:
  - spdlog (a logging library)
  - A minimal version of AZCore (a math library) that contains only:
    - Vector3
    - Aabb
  - cxxopts (a command line option parser library)
- Utility functions specific to SimSpace Weaver

The demo framework consists of a library, source files, and CMakeLists. The files are included in the SimSpace Weaver app SDK distributable package.

Working with service quotas

This section describes how to work with the service quotas for SimSpace Weaver. **Quotas** are also called **limits**. For a list of service quotas, see SimSpace Weaver endpoints and quotas. The APIs in this section are from the set of **app APIs**. App APIs are a different than the service APIs. The app APIs are part of the SimSpace Weaver app SDK. You can find the documentation for the app APIs in the app SDK folder on your local system:

```
sdk-folder\SimSpaceWeaverAppSdk-sdk-version\documentation\index.html
```

Topics

Get the limits for an app

You can use the **RuntimeLimits** app API to query the limits for an app.

```
Result<Limit> RuntimeLimit(Application& app, LimitType type)
```

Parameters

Application& app

A reference to the app.

LimitType type

An enum with the following limit types:

```
enum LimitType {
    Unset = 0,
    EntitiesPerPartition = 1,
    RemoteEntityTransfers = 2,
    LocalEntityTransfers = 3
};
```

The following example queries the entity count limit.

```
WEAVERRUNTIME_TRY(auto entity_limit,
    Api::RuntimeLimit(m_app, Api::LimitType::EntitiesPerPartition))
Log::Info("Entity count limit", entity_limit.value);
```

Get the amount of resources used by an app

You can call the **RuntimeMetrics** app API to get the amount of resources used by an app:

```
Result<std::reference_wrapper<const AppRuntimeMetrics>> RuntimeMetrics(Application&
 app) noexcept
```

Parameters

Application& app

A reference to the app.

The API returns a reference to a struct that contains the metrics. A counter metric holds a running total value and only increases. A gauge metric holds a value that can increase or decrease. The application runtime updates a counter whenever an event increases the value. The runtime only updates the gauges when you call the API. SimSpace Weaver guarantees that the reference is valid for the lifetime of the app. Repeat calls to the API won't change the reference.

```
struct AppRuntimeMetrics {
    uint64_t total_committed_ticks_gauge,

    uint32_t active_entity_gauge,
    uint32_t ticks_since_reset_counter,

    uint32_t load_field_counter,
    uint32_t store_field_counter,

    uint32_t created_entity_counter,
    uint32_t deleted_entity_counter,

    uint32_t entered_entity_counter,
    uint32_t exited_entity_counter,

    uint32_t rejected_incoming_transfer_counter,
    uint32_t rejected_outgoing_transfer_counter
}
```

Reset metrics

The **ResetRuntimeMetrics** app API resets the values in the `AppRuntimeMetrics` struct.

```
Result<void> ResetRuntimeMetrics(Application& app) noexcept
```

The following example demonstrates how you can call **ResetRuntimeMetrics** in your app.

```
if (ticks_since_last_report > 100)
{
    auto metrics = WEAVERRUNTIME_EXPECT(Api::RuntimeMetrics(m_app));
    Log::Info(metrics);

    ticks_since_last_report = 0;

    WEAVERRUNTIME_EXPECT(Api::ResetRuntimeMetrics(m_app));
}
```

Exceeding a limit

An app API call that exceeds a limit will return an `ErrorCode::CapacityExceeded`, except for entity transfers. SimSpace Weaver handles entity transfers asynchronously as part of **Commit** and **BeginUpdate** app API operations, so there isn't a specific operation that returns an error if a transfer fails because of the entity transfer limit. To detect transfer failures, you can compare the current values of `rejected_incoming_transfer_counter` and `rejected_outgoing_transfer_counter` (in the `AppRuntimeMetrics` struct) with their previous values. Rejected entities won't be in the partition, but the app can still simulate them.

Running out of memory

SimSpace Weaver uses a garbage collector process to clean up and release freed memory. It's possible to write data faster than the garbage collector can release memory. If this happens, write operations might exceed the app's reserved memory limit. SimSpace Weaver will return an internal error with a message that contains `OutOfMemory` (and additional details). For more information, see Spread writes across time.

Best practices

The following best practices are general guidelines for designing your apps to avoid exceeding limits. They might not apply to your specific app design.

Monitor frequently and slow down

You should monitor your metrics frequently and slow down operations that are close to reaching a limit.

Avoid exceeding subscription limits and transfer limits

If possible, design your simulation to reduce the number of remote subscriptions and entity transfers. You can use placement groups to place multiple partitions on the same worker and reduce the need for remote entity transfers between workers.

Spread writes across time

The number and size of updates in a tick can have a significant impact on the time and memory required to commit a transaction. Large memory requirements can cause the application runtime to run out of memory. You can spread writes across time to lower the average total size of updates per tick. This can help improve performance and avoid exceeding limits. We recommend that you don't write more than an average of 12 MB on each tick or 1.5 KB for each entity.

Debugging simulations

You can use the following methods to get information about your simulations.

Topics

Use SimSpace Weaver Local and look at console output

We recommend that you develop your simulations locally first and then run them in the AWS Cloud. You can view console output directly when you run with SimSpace Weaver Local. For more information, see Local development in SimSpace Weaver.

Look at your logs in Amazon CloudWatch Logs

When you run your simulation in the AWS Cloud the console output of your apps is sent to log streams in Amazon CloudWatch Logs. Your simulation also writes other log data. You must enable logging in your simulation schema if you want your simulation to write log data. For more information, see SimSpace Weaver logs in Amazon CloudWatch Logs.

Warning

Your simulation can produce large amounts of log data. The log data can grow very quickly. You should watch your logs closely and stop your simulations when you don't need them running anymore. Logging can create large costs.

Use describe API calls

You can use the following service APIs to get information about your simulations in the AWS Cloud.

- **ListSimulations** – get a list of all of your simulations in the AWS Cloud.

  Example Example

  ```
  aws simspaceweaver list-simulations
  ```
- **DescribeSimulation** – get details about a simulation.

  Example Example

  ```
  aws simspaceweaver describe-simulation --simulation MySimulation
  ```
- **DescribeApp** – get details about an app.

  Example Example

  ```
  aws simspaceweaver describe-app --simulation MySimulation --domain MyCustomDomain --
  app MyCustomApp
  ```

For more information about the SimSpace Weaver APIs, see SimSpace Weaver API references.

Connect a client

You can connect a client to a running custom or service app that you defined with an `endpoint_config` in your simulation schema. The SimSpace Weaver app SDK includes sample clients that you can use to view the sample application. You can look at the source code for these sample clients and the sample application to see how you can create your own clients. For more information about how to build and run the sample clients, see the tutorials in Getting started with SimSpace Weaver.

You can find the source code for the sample clients in the following folder:

- `*sdk-folder*\packaging-tools\clients\PathfindingSampleClients\`

Debugging local simulations

You can debug your SimSpace Weaver Local apps with Microsoft Visual Studio. For more information about how to debug with Visual Studio, see the Microsoft Visual Studio documentation.

To debug your local simulation

1. Make sure that your `schema.yaml` is in your working directory.
2. In **Visual Studio**, open the context menu for each app that you want to debug (such as `PathfindingSampleLocalSpatial` or `PathfindingSampleLocalView`) and set the working directory in the debugging section.
3. Open the context menu for the app you want to debug and select **Set as Startup project**.
4. Choose **F5** to start debugging the app.

The requirements to debug a simulation are the same as the requirements to run a simulation normally. You must start the number of spatial apps specified in the schema. For example, if your schema specifies a 2x2 grid and you start a spatial app in debug mode, the simulation will not run until you start 3 more spatial apps (in debug mode or not in debug mode).

To debug a custom app, you must first start your spatial apps and then start the custom app in the debugger.

Note that your simulation runs in lock step. As soon as an app reaches a breakpoint, all other apps will pause. After you continue from that breakpoint, the other apps will continue.

Custom containers

AWS SimSpace Weaver apps run in containerized Amazon Linux 2 (AL2) environments. In the AWS Cloud, SimSpace Weaver runs your simulations in Docker containers built from an `amazonlinux:2` image served from Amazon Elastic Container Registry (Amazon ECR). You can create a custom Docker image, store it in in Amazon ECR, and use that image for your simulation instead of the default Docker image that we provide.

You can use a custom container to manage your software dependencies and include additional software components that aren't in the standard Docker image. For example, you can add the publicly-available software libraries that your app uses to the container and only put your custom code in the app zip file.

Important

We only support AL2 Docker images hosted in Amazon ECR repositories, either in Amazon ECR Public Gallery or your private Amazon ECR registry. We don't support Docker images hosted outside Amazon ECR. For more information about Amazon ECR, see *Amazon Elastic Container Registry Documentation*.

Topics

- Create a custom container
- Modify a project to use a custom container
- Frequently asked questions about custom containers
- Troubleshooting custom containers

Create a custom container

These instructions assume that you know how to use Docker and Amazon Elastic Container Registry (Amazon ECR). For more information about Amazon ECR, see the *Amazon ECR User Guide*.

Prerequisites

- The IAM identity (use or role) that you use to perform these actions has the correct permissions to use Amazon ECR

- Docker is installed on your local system

To create a custom container

1. Create your `Dockerfile`.

   A `Dockerfile` to run AWS SimSpace Weaver apps starts with the Amazon Linux 2 image in Amazon ECR.

   ```
   # parent image required to run AWS SimSpace Weaver apps
   FROM public.ecr.aws/amazonlinux/amazonlinux:2
   ```

2. Build your `Dockerfile`.
3. Upload your container image to Amazon ECR.
   - Use the AWS Management Console.
   - Use the AWS Command Line Interface.

   Note

   If you get an `AccessDeniedException` error when you try to upload your container image to Amazon ECR, your IAM identity (user or role) might not have the necessary permissions to use Amazon ECR. You can attach the `AmazonEC2ContainerRegistryPowerUser` AWS managed policy to your IAM identity and try again. For more information about how to attach a policy, see Adding and removing IAM identity permissions in the *AWS Identity and Access Management User Guide*.

Modify a project to use a custom container

These instructions assume that you already know how to use AWS SimSpace Weaver and want to make your app storage and development workflows in the AWS Cloud more efficient.

Prerequisites

- You have a custom container in Amazon Elastic Container Registry (Amazon ECR). For more information about creating a custom container, see Create a custom container.

To modify your project to use a custom container

1. Add permissions to your project's simulation app role to use Amazon ECR.

   a. If you don't already have an IAM policy with following permissions, create the policy. We suggest the policy name `simspaceweaver-ecr`. For more information about how to create an IAM policy, see Creating IAM policies in the *AWS Identity and Access Management User Guide*.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "Statement",
               "Effect": "Allow",
               "Action": [
                   "ecr:BatchGetImage",
                   "ecr:GetDownloadUrlForLayer",
                   "ecr:GetAuthorizationToken"
               ],
               "Resource": "*"
           }
       ]
   }
   ```

   b. Find the name of your project's simulation app role:

      i. In a text editor, open the AWS CloudFormation template:

      ```
      sdk-folder\PackagingTools\sample-stack-template.yaml
      ```

      ii. Find the `RoleName` property under `WeaverAppRole`. The value is the name of your project's simulation app role.

      Example

      ```
      AWSTemplateFormatVersion: "2010-09-09"
      Resources:
        WeaverAppRole:
          Type: 'AWS::IAM::Role'
          Properties:
            RoleName: 'weaver-MySimulation-app-role'
      ```

```
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
        - Effect: Allow
          Principal:
            Service:
              - 'simspaceweaver.amazonaws.com'
```

c. Attach the `simspaceweaver-ecr` policy to the project's simulation app role. For more information about how to attach a policy, see [Adding and removing IAM identity permissions](#) in the *AWS Identity and Access Management User Guide*.

d. Navigate to *`sdk-folder`* and run the following command to update the sample SimSpace Weaver stack:

```
python setup.py --cloudformation
```

2. Specify your container images in the project's simulation schema.

- You can add the optional `default_image` property under `simulation_properties` to specify a default custom container image for all domains.
- Add the `image` property in the `app_config` for a domain that you want to use a custom container image. Specify the Amazon ECR repository URI as the value. You can specify a different image for each domain.
  - If `image` isn't specified for a domain and `default_image` is specified, apps in that domain use the default image.
  - If `image` isn't specified for a domain and `default_image` isn't specified, apps in that domain run in a standard SimSpace Weaver container.

Example Schema snippet that includes custom container settings

```
sdk_version: "1.17.0"
simulation_properties:
  log_destination_service: "logs"
  log_destination_resource_name: "MySimulationLogs"
  default_entity_index_key_type: "Vector3<f32>"
  default_image: "111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-
repository:latest" # image to use if no image specified for a domain
domains:
  MyCustomDomain:
    launch_apps_via_start_app_call: {}
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MyViewApp.zip"
      launch_command: ["MyViewApp"]
      required_resource_units:
        compute: 1
      endpoint_config:
        ingress_ports:
          - 7000
      image: "111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-
repository:latest" # custom container image to use for this domain
  MySpatialDomain:
    launch_apps_by_partitioning_strategy:
      partitioning_strategy: "MyGridPartitioning"
      grid_partition:
        x: 2
        y: 2
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MySpatialApp.zip"
      launch_command: ["MySpatialApp"]
      required_resource_units:
        compute: 1
      image: "111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-
repository:latest" # custom container image to use for this domain
```

3. Build and upload your project as usual.

Frequently asked questions about custom containers

Q1. What do I do if I want to change the contents of my container?

- **For a running simulation** – You can't change the container for a running simulation. You must build a new container and start a new simulation that uses that container.
- **For a new simulation** – Build a new container, upload it to Amazon Elastic Container Registry (Amazon ECR), and start a new simulation that uses that container.

Q2. How can I change the container image for my simulation?

- **For a running simulation** – You can't change the container for a running simulation. You must start a new simulation that uses the new container.

- **For a new simulation –** Specify the new container image in your project's simulation schema. For more information, see Modify a project to use a custom container.

Troubleshooting custom containers

Topics

- AccessDeniedException when uploading your image to Amazon Elastic Container Registry (Amazon ECR)
- A simulation that uses a custom container fails to start

AccessDeniedException when uploading your image to Amazon Elastic Container Registry (Amazon ECR)

If you get an `AccessDeniedException` error when you try to upload your container image to Amazon ECR, your IAM identity (user or role) might not have the necessary permissions to use Amazon ECR. You can attach the `AmazonEC2ContainerRegistryPowerUser` AWS managed policy to your IAM identity and try again. For more information about how to attach a policy, see Adding and removing IAM identity permissions in the *AWS Identity and Access Management User Guide*.

A simulation that uses a custom container fails to start

Troubleshooting tips

- If logging is enabled for your simulation, check your error logs.
- Test your simulation without a custom container.
- Test your simulation locally. For more information, see Local development in SimSpace Weaver.

Working with Python

You can use Python for your SimSpace Weaver apps and client. The Python software development kit (Python SDK) is included as part of the standard SimSpace Weaver app SDK distributable package. Development with Python works in a similar way as development in the other supported languages.

Important

SimSpace Weaver only supports Python version 3.9.

Important

SimSpace Weaver support for Python requires SimSpace Weaver version 1.15.0 or later.

Topics

Creating a Python project

Python custom container

To run your Python-based SimSpace Weaver simulation in the AWS Cloud, you can create a custom container that includes the necessary dependencies. For more information, see Custom containers.

A Python custom container must include the following:

- gcc
- openssl-devel
- bzip2-devel
- libffi-devel
- wget
- tar
- gzip
- make

- Python (version 3.9)

If you use the PythonBubblesSample template to create your project, you can run the quick-start.py script (located in the tools folder of your project) to create a Docker image with the necessary dependencies. The script uploads the image to Amazon Elastic Container Registry (Amazon ECR).

The quick-start.py script uses the following Dockerfile:

```
FROM public.ecr.aws/amazonlinux/amazonlinux:2
RUN yum -y install gcc openssl-devel bzip2-devel libffi-devel
RUN yum -y install wget
RUN yum -y install tar
RUN yum -y install gzip
RUN yum -y install make
WORKDIR /opt
RUN wget https://www.python.org/ftp/python/3.9.0/Python-3.9.0.tgz
RUN tar xzf Python-3.9.0.tgz
WORKDIR /opt/Python-3.9.0
RUN ./configure --enable-optimizations
RUN make altinstall
COPY requirements.txt ./
RUN python3.9 -m pip install --upgrade pip
RUN pip3.9 install -r requirements.txt
```

You can add your own dependencies to the Dockerfile:

```
RUN yum -y install dependency-name
```

The requirements.txt file contains a list of Python packages required for the PythonBubblesSample sample simulation:

```
Flask==2.1.1
```

You can add your own Python package dependencies to the requirements.txt:

```
package-name==version-number
```

The Dockerfile and requirements.txt are in the tools folder of your project.

Important

You technically don't have to use a custom container with your Python simulation, but we strongly recommend that you use a custom container. The standard Amazon Linux 2 (AL2) container that we provide doesn't have Python. Therefore, if you don't use a custom container that has Python, you must include Python and the required dependencies in each app zip file that you upload to SimSpace Weaver.

Starting a Python simulation

You can start your Python-based simulation the same way as a regular SimSpace Weaver simulation, both in SimSpace Weaver Local and in SimSpace Weaver in the AWS Cloud. For more information, see the tutorials in Getting started with SimSpace Weaver.

The `PythonBubblesSample` includes its own Python sample client. For more information, see The sample Python client.

The sample Python client

If you use the `PythonBubblesSample` template to create a project then your project contains a Python sample client. You can use the sample client to view the `PythonBubblesSample` simulation. You can also use the sample client as the starting point to create your own Python client.

The following procedure assumes that you created a `PythonBubblesSample` project and started its simulation.

To start the Python client

1. In a **command prompt window**, go to the `PyBubbleClient` sample project folder.

```
cd sdk-folder\Clients\HTTP\PyBubbleClient
```

2. Run the Python client.

```
python tkinter_client.py --host ip-address --port port-number
```

Parameters

host

The IP address of your simulation. For a simulation started in the AWS Cloud, you can find your simulation's IP address in the SimSpace Weaver console or use the procedure in Get the IP address and port number of a custom app of the quick start tutorial. For a local simulation, use `127.0.0.1` as the IP address.

port

The port number of your simulation. For a simulation started in the AWS Cloud, this is the `Actual` port number. You can find your simulation's port number in the SimSpace Weaver console or use the procedure in Get the IP address and port number of a custom app of the quick start tutorial. For a local simulation, use `7000` as the port number.

simsize

The maximum number of entities to display in the client.

Frequently asked questions about using Python

Q1. What versions of Python are supported?

SimSpace Weaver only supports Python version 3.9.

Troubleshooting problems related to Python

Topics

- Failure during custom container creation
- Your Python simulation fails to start
- A Python simulation or view client throws a ModuleNotFound error

Failure during custom container creation

If you get an error `no basic auth credentials` after you run `quick-start.py` then there could be a problem with your temporary credentials for Amazon ECR. Run the following command with your AWS Region ID and AWS account number:

```
aws ecr get-login-password --region region | docker login --username AWS --password-
stdin account_id.dkr.ecr.region.amazonaws.com
```

Example

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-
stdin 111122223333.dkr.ecr.region.amazonaws.com
```

Important

Make sure that the AWS Region you specify is the same one that you use for your simulation. Use one of the AWS Regions that SimSpace Weaver supports. For more information, see SimSpace Weaver endpoints and quotas.

After you run the `aws ecr` command, run `quick-start.py` again.

Other troubleshooting resources to check

- Troubleshooting custom containers
- Amazon ECR troubleshooting in the *Amazon ECR User Guide*
- Setting up with Amazon ECR in the *Amazon ECR User Guide*

Your Python simulation fails to start

You might see an `Unable to start app` error in your simulation's management log. This can happen if your custom container creation failed. For more information, see Failure during custom container creation. For more information about logs, see SimSpace Weaver logs in Amazon CloudWatch Logs.

If you're sure that there's nothing wrong with your container, check your app's Python source code. You can use SimSpace Weaver Local to test your app. For more information, see Local development in SimSpace Weaver.

A Python simulation or view client throws a ModuleNotFound error

Python throws a `ModuleNotFound` error when it can't locate a required Python package.

If your simulation is in the AWS Cloud, make sure that your custom container has all of the required dependencies listed in your `requirements.txt`. Remember to run `quick-start.py` again if you edit `requirements.txt`.

If you get the error for the `PythonBubblesSample` client, use `pip` to install the indicated package:

```
pip install package-name==version-number
```

Support for other engines

You can use your own custom C++ engine with SimSpace Weaver. We are currently developing support for the following engines. There is separate documentation for each of these engines.

Important

Integrations with the engines listed here are experimental. They are available for preview.

Engines

- Unity (minimum version 2022.3.19.F1)
- Unreal Engine (minimum version 5.0)

Unity

You must have the Unity development environment already installed before you build SimSpace Weaver simulations with Unity. For more information, see the separate directions:

```
sdk-folder\Unity-Guide.pdf
```

Unreal Engine

You must build an Unreal Engine dedicated server from source code. The SimSpaceWeaverAppSdkDistributable includes a version of the PathfindingSample for Unreal Engine. For more information, see the separate directions:

```
sdk-folder\Unreal-Engine-Guide.pdf
```

Use of licensed software with AWS SimSpace Weaver

AWS SimSpace Weaver allows you to build simulations with your choice of simulation engine and content. In connection with your use of SimSpace Weaver, you are responsible for obtaining, maintaining, and adhering to the license terms of any software or content you use in your simulations. Verify your licensing agreement allows you to deploy your software and content in a virtual hosted environment.

Managing your resources with AWS CloudFormation

You can use AWS CloudFormation to manage your AWS SimSpace Weaver resources. AWS CloudFormation is a separate AWS service that helps you specify, provision, and manage your AWS infrastructure as code. With AWS CloudFormation you create a JSON or YAML file, called a *template*. Your template specifies the details of your infrastructure. AWS CloudFormation uses your template to provision your infrastructure as a single unit, called a *stack*. When you delete your stack, you can have AWS CloudFormation delete everything in the stack at the same time. You can manage your template using standard source code management processes (for example, tracking it in a version control system like Git). For more information about AWS CloudFormation, see the *AWS CloudFormation User Guide*.

Your simulation resource

In AWS, a *resource* is an entity that you can work with. Examples include an Amazon EC2 instance, an Amazon S3 bucket, or an IAM role. Your SimSpace Weaver simulation is a resource. In configurations, you usually specify an AWS resource in the form `AWS::`*`service`*`::resource`. For SimSpace Weaver, you specify your simulation resource as `AWS::SimSpaceWeaver::Simulation`. For more information about your simulation resource in AWS CloudFormation, see the SimSpace Weaver section in the *AWS CloudFormation User Guide*.

How can I use AWS CloudFormation with SimSpace Weaver?

You can create an AWS CloudFormation template that specifies the AWS resources that you want to provision. Your template can specify an entire architecture, part of an architecture, or a small solution. For example, you could specify an architecture for your SimSpace Weaver solution that includes Amazon S3 buckets, IAM permissions, a supporting database in Amazon Relational Database Service or Amazon DynamoDB, and your `Simulation` resource. You can then use AWS CloudFormation to provision all of those resources as a unit, and at the same time.

Example template that creates IAM resources and starts a simulation

The following example template creates an IAM role and permissions that SimSpace Weaver needs to perform actions in your account. The SimSpace Weaver app SDK scripts create the role and permissions in a specific AWS Region when you create a project, but you can use an AWS CloudFormation template to deploy the simulation to another AWS Region without running the scripts again. For example, you can do this to set up a backup simulation for disaster recovery purposes.

In this example, the original simulation name is `MySimulation`. A bucket for the schema already exists in the AWS Region where AWS CloudFormation will build the stack. The bucket contains a version of the schema that is properly configured to run the simulation in that AWS Region. Recall that the schema specifies the location of your app zip files, which is an Amazon S3 bucket in the same AWS Region as the simulation. The app zips bucket and files must already exist in the AWS Region when AWS CloudFormation builds the stack, otherwise your simulation won't start. Note that the bucket name in this example includes the AWS Region, but that doesn't determine where the bucket is actually located. You must make sure that the bucket is actually in that AWS Region (you can check the bucket properties in the Amazon S3 console, with the Amazon S3 APIs, or with the Amazon S3 commands in the AWS CLI).

This example uses some built-in functions and parameters in AWS CloudFormation to perform variable substitution. For more information, see Intrinsic function reference and Pseudo parameters reference in the *AWS CloudFormation User Guide*.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  WeaverAppRole:
    Type: AWS::IAM::Role
    Properties:
      RoleName: SimSpaceWeaverAppRole
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
        - Effect: Allow
          Principal:
            Service:
              - simspaceweaver.amazonaws.com
          Action:
            - sts:AssumeRole
      Path: /
      Policies:
        - PolicyName: SimSpaceWeaverAppRolePolicy
          PolicyDocument:
            Version: 2012-10-17
            Statement:
            - Effect: Allow
              Action:
                - logs:PutLogEvents
                - logs:DescribeLogGroups
                - logs:DescribeLogStreams
                - logs:CreateLogGroup
                - logs:CreateLogStream
              Resource: *
            - Effect: Allow
              Action:
                - cloudwatch:PutMetricData
              Resource: *
            - Effect: Allow
              Action:
                - s3:ListBucket
                - s3:PutObject
                - s3:GetObject
              Resource: *
  MyBackupSimulation:
    Type: AWS::SimSpaceWeaver::Simulation
    Properties:
      Name: !Sub 'mySimulation-${AWS::Region}'
      RoleArn: !GetAtt WeaverAppRole.Arn
      SchemaS3Location:
        BucketName: !Sub 'weaver-mySimulation-${AWS::AccountId}-schemas-${AWS::Region}'
        ObjectKey: !Sub 'schema/mySimulation-${AWS::Region}-schema.yaml'
```

Using snapshots with AWS CloudFormation

A snapshot is a backup of a simulation. The following example starts a new simulation from a snapshot instead of from a schema. The snapshot in this example was created from a SimSpace Weaver app SDK project simulation. AWS CloudFormation creates the new simulation resource and initializes it with data from the snapshot. The new simulation can have a different `MaximumDuration` than the original simulation.

We recommend that you make and use a copy of your original simulation's app role. The original simulation's app role could be deleted if you delete that simulation's AWS CloudFormation stack.

```
Description: "Example - Start a simulation from a snapshot"
Resources:
  MyTestSimulation:
    Type: "AWS::SimSpaceWeaver::Simulation"
    Properties:
      MaximumDuration: "2D"
      Name: "MyTestSimulation_from_snapshot"
      RoleArn: "arn:aws:iam::111122223333:role/weaver-MyTestSimulation-app-role-copy"

      SnapshotS3Location:
        BucketName: "weaver-mytestsimulation-111122223333-artifacts-us-west-2"
        ObjectKey: "snapshot/MyTestSimulation_22-12-15_12_00_00-230428-1207-13.zip"
```

Snapshots

You can create a *snapshot* to back up your simulation entity data at any point in time. SimSpace Weaver creates a .zip file in an Amazon S3 bucket. You can create a new simulation with the snapshot. SimSpace Weaver initializes the State Fabric of your new simulation with the entity data stored in the snapshot, starts the spatial and service apps that were running when the snapshot was created, and sets the clock to the appropriate tick. SimSpace Weaver gets the configuration of your simulation from the snapshot instead of from a schema file. Your app .zip files must be in the same location in Amazon S3 as they were in the original simulation. You must start any custom apps separately.

Topics

Use cases for snapshots

Return to a previous state and explore branching scenarios

You can create a snapshot of your simulation to save it at a specific state. You can then create multiple new simulations from that snapshot and explore different scenarios that could branch from that state.

Disaster recovery and security best practices

We recommend that you regularly backup your simulation, especially for simulations that run for more than 1 hour or use multiple workers. Backups can help you recover from disasters and security incidents. Snapshots provide a way for you to backup your simulation. Snapshots require your app .zip files to exist in the same location in Amazon S3 as they were before. If you need to be able to move your app .zip files to another location, you must use a custom backup solution.

For more information about other best practices, see Best practices when working with SimSpace Weaver and Security best practices for SimSpace Weaver.

Extend the duration of your simulation

Your *simulation resource* is the representation of your simulation in SimSpace Weaver. All simulation resources have a `MaximumDuration` setting. A simulation resource automatically stops when it reaches its `MaximumDuration`. The maximum value of `MaximumDuration` is `14D` (14 days).

If you need your simulation to persist longer than the `MaximumDuration` of its simulation resource, you can create a snapshot before the simulation resource reaches its `MaximumDuration`. You can start a new simulation (create a new simulation resource) with your snapshot. SimSpace Weaver initializes your entity data from the snapshot, starts the same spatial and service apps that ran before, and restores the clock. You can start your custom apps and perform any additional custom initialization. You can set the `MaximumDuration` of the new simulation resource to a different value when you start it.

Use the SimSpace Weaver console to work with snapshots

You can use the SimSpace Weaver console to create a snapshot of your simulation.

Topics

- Use the console to create a snapshot
- Use the console to start a simulation from a snapshot

Use the console to create a snapshot

To create a snapshot

1. Sign to the AWS Management Console and connect to the SimSpace Weaver console.
2. Choose **Simulations** from the navigation pane.
3. Select the radio button next to simulation name. Your simulation's **Status** must be **Started**.
4. At the top of the page, choose **Create snapshot**.
5. Under **Snapshot settings**, for **Snapshot destination**, enter the Amazon S3 URI of a bucket or a bucket and folder where you want SimSpace Weaver to create your snapshot. You can choose **Browse S3** if you prefer to browse your available buckets and select a location.

   Important

   The Amazon S3 bucket must be in the same AWS Region as the simulation.

   Note

   SimSpace Weaver creates a `snapshot` folder inside your selected snapshot destination. SimSpace Weaver creates the snapshot .zip file in that `snapshot` folder.
6. Choose **Create snapshot**.

Use the console to start a simulation from a snapshot

To start a simulation from a snapshot, your snapshot .zip file must exist in an Amazon S3 bucket that your simulation can access. Your simulation uses permissions defined in the app role that you select when you start the simulation. All of the app .zip files from the original simulation must exist at their same locations as when the snapshot was created.

To start a simulation from a snapshot

1. Sign to the AWS Management Console and connect to the SimSpace Weaver console.

2. Choose **Simulations** from the navigation pane.
3. At the top of the page, choose **Start simulation**.
4. Under **Simulation settings**, enter a name and optional description for your simulation. Your simulation name must be unique in your AWS account.
5. For **Simulation start method**, choose **Use a snapshot in Amazon S3**.
6. For **Amazon S3 URI for snapshot**, enter the Amazon S3 URI of your snapshot file, or choose **Browse S3** to browse and select the file.

   Important

   The Amazon S3 bucket must be in the same AWS Region as the simulation.
7. For **IAM role**, select the app role that your simulation will use.
8. For **Maximum duration**, enter the maximum amount of time that your simulation resource should run for. The maximum value is `14D`. For more information about the maximum duration, see .
9. Under **Tags - *optional***, choose **Add new tag** if you want to add a tag.
10. Choose **Start simulation**.

Use the AWS CLI to work with snapshots

You can use the AWS CLI to call the SimSpace Weaver APIs from a command prompt. You must have the AWS CLI installed and configured properly. For more information, see Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

Topics

Use the AWS CLI to create a snapshot

To create a snapshot

- At a **command prompt**, call the `CreateSnapshot` API.

```
aws simspaceweaver create-snapshot --simulation simulation-name —destination s3-
destination
```

Parameters

simulation

The name of a started simulation. You can use `aws simspaceweaver list-simulations` to see the names and statuses of your simulations.

destination

A string that specifies the destination Amazon S3 bucket and optional object key prefix for your snapshot file. Your object key prefix is usually a folder in your bucket. SimSpace Weaver creates your snapshot inside a `snapshot` folder at this destination.

Important

The Amazon S3 bucket must be in the same AWS Region as the simulation.

Example

```
aws simspaceweaver create-snapshot —simulation
 MyProjectSimulation_23-04-29_12_00_00 —destination BucketName=weaver-
myproject-111122223333-artifacts-us-west-2,ObjectKeyPrefix=myFolder
```

For more information about the `CreateSnapshot` API, see CreateSnapshot in the *AWS SimSpace Weaver API Reference*.

Use the AWS CLI to start a simulation from a snapshot

To start a simulation from a snapshot

- At a **command prompt**, call the `StartSimulation` API.

```
aws simspaceweaver start-simulation --name simulation-name --role-arn role-arn --
snapshot-s3-location s3-location
```

Parameters

name

The name of the new simulation. The simulation name must be unique in your AWS account. You can use `aws simspaceweaver list-simulations` to see the names of your existing simulations.

role-arn

The Amazon Resource Name (ARN) of the app role that your simulation will use.

snapshot-s3-location

A string that specifies the Amazon S3 bucket and object key of your snapshot file.

Important

The Amazon S3 bucket must be in the same AWS Region as the simulation.

Example

```
aws simspaceweaver start-simulation —name MySimulation —role-arn
 arn:aws:iam::111122223333:role/weaver-MyProject-app-role —snapshot-s3-location
 BucketName=weaver-myproject-111122223333-artifacts-us-west-2,ObjectKey=myFolder/
snapshot/MyProjectSimulation_23-04-29_12_00_00-230429-1530-27.zip
```

For more information about the `StartSimulation` API, see StartSimulation in the *AWS SimSpace Weaver API Reference*.

Frequently asked questions about snapshots

Does my simulation continue to run during a snapshot?

Your simulation resources continue to run during a snapshot and you continue to receive billing charges for that time. The time counts towards your simulation's maximum duration. Your apps don't receive ticks while the snapshot is in progress. If your clock status was `STARTED` when the snapshot creation started, your clock will still indicate `STARTED` status. Your apps receive ticks again after the snapshot finishes. If your clock status was `STOPPED` then your clock status will remain STOPPED. Note that a simulation with a STARTED status is running even if its clock status is STOPPED.

What happens if a snapshot is in progress and my simulation reaches its maximum duration?

Your simulation will finish the snapshot and then stop as soon as the snapshot process ends (either successfully or unsuccessfully). We recommend that you test the snapshot process beforehand to find out how long it takes, the size of the snapshot file you can expect, and if it should complete successfully.

What happens if I stop a simulation that has a snapshot in progress?

A snapshot in progress stops immediately when you stop the simulation. It won't create a snapshot file.

How can I stop a snapshot in progress?

The only way to stop a snapshot in progress is to the stop the simulation. **You can't restart a simulation after you stop it.**

How long will it take to complete my snapshot?

The time required to create a snapshot depends on your simulation. We recommend that you test the snapshot process beforehand to find out how long it will take for your simulation.

How large will my snapshot file be?

The size of a snapshot file depends on your simulation. We recommend that you test the snapshot process beforehand to find out how large the file could be for your simulation.

Messaging

The messaging API simplifies application to application communication within the simulation. APIs to send and receive messages are part of the SimSpace Weaver app SDK. Messaging currently uses a best effort approach to send and receive messages. SimSpace Weaver attempts to send/ receive messages on the next simulation tick, but there are no delivery, ordering, or arrival time guarantees.

Topics

Use cases for messaging

Communicate between simulation applications

Use the messaging API to communicate between the applications in your simulation. Use it to change the state of entities at a distance, change entity behavior, or broadcast information to the entire simulation.

Acknowledge receipt of a message

Sent messages contain information about the sender in the message header. Use this information to send back an acknowledgement reply on receipt of a message.

Forward data received by a custom app to other apps within the simulation

Messaging is not a replacement for how clients connect to custom apps running in SimSpace Weaver. However, messaging does allow users a method of forwarding data from custom apps receiving client data to other apps that do not have an external connection. The message flow can also work in reverse, allowing apps with no external connection to forward data to a custom app then to a client.

Using the messaging APIs

The messaging APIs are contained within the SimSpace Weaver app SDK (minimum version 1.16.0). Messaging is supported in C++, Python, and our integrations with Unreal Engine 5 and Unity.

There are two functions that handle message transactions: `SendMessage` and `ReceiveMessages`. All sent messages contain a destination and a payload. The `ReceiveMessages` API returns a list of messages currently in an app's inbound message queue.

C++

Send message

```
AWS_WEAVERRUNTIME_API Result<void> SendMessage(
    Transaction& txn,
    const MessagePayload& payload,
    const MessageEndpoint& destination,
    MessageDeliveryType deliveryType = MessageDeliveryType::BestEffort
    ) noexcept;
```

Receive messages

```
AWS_WEAVERRUNTIME_API Result<MessageList> ReceiveMessages(
    Transaction& txn) noexcept;
```

Python

Send message

```
api.send_message(
 txn, # Transaction
 payload, # api.MessagePayload
 destination, # api.MessageDestination
 api.MessageDeliveryType.BestEffort # api.MessageDeliveryType
)
```

Receive messages

```
api.receive_messages(
 txn, # Transaction
) -> api.MessageList
```

Topics

- Sending messages
- Receiving messages
- Replying to the sender

Sending messages

Messages consist of a transaction (similar to other Weaver API calls), a payload, and a destination.

Message payload

The message payload is a flexible data structure of up to 256 bytes. We recommend the following as a best practice for creating your message payloads.

To create the message payload

1. Create a data structure (such as a `struct` in C++) that defines the contents of the message.
2. Create the message payload that contains the values to send in your message.
3. Create the `MessagePayload` object.

Message destination

The destination of a message is defined by the `MessageEndpoint` object. This includes both an endpoint type and an endpoint ID. The only endpoint type currently supported is `Partition`, which enables you to address messages to other partitions in the simulation. The endpoint ID is the partition ID of your target destination.

You can only provide 1 destination address in a message. Create and send multiple messages if you want to send messages to more than 1 partition at the same time.

For guidance on how to resolve a message endpoint from a position, see Tips when working with messaging.

Send the message

You can use the `SendMessage` API after you create the destination and payload objects.

C++

```
Api::SendMessage(transaction, payload, destination,
 MessageDeliveryType::BestEffort);
```

Python

```
api.send_message(txn, payload, destination, api.MessageDeliveryType.BestEffort)
```

Full example of sending messages

The following example demonstrates how you can construct and send a generic message. This example sends 16 individual messages. Each message contains a payload with a value betwen 0 and 15, and the current simulation tick.

Example

C++

```
// Message struct definition
struct MessageTickAndId
{
    uint32_t id;
    uint32_t tick;
};

Aws::WeaverRuntime::Result<void> SendMessages(Txn& txn) noexcept
{
     // Fetch the destination MessageEndpoint with the endpoint resolver
    WEAVERRUNTIME_TRY(
        Api::MessageEndpoint destination,
        Api::Utils::MessageEndpointResolver::ResolveFromPosition(
        txn,
            "MySpatialSimulation",
            Api::Vector2F32 {231.3, 654.0}
        )
    );
    Log::Info("destination: ", destination);

    WEAVERRUNTIME_TRY(auto tick, Api::CurrentTick(txn));

    uint16_t numSentMessages = 0;
    for (std::size_t i=0; i<16; i++)
    {
        // Create the message that'll be serialized into payload
        MessageTickAndId message {i, tick.value};

        // Create the payload out of the struct
        const Api::MessagePayload& payload = Api::Utils::CreateMessagePayload(
            reinterpret_cast<const std::uint8_t*>(&message),
            sizeof(MessageTickAndId)
        );

        // Send the payload to the destination
        Result<void> result = Api::SendMessage(txn, payload, destination);
        if (result.has_failure())
        {
            // SendMessage has failure modes, log them
            auto error = result.as_failure().error();
            std::cout<< "SendMessage failed, ErrorCode: " << error << std::endl;
            continue;
        }

        numSentMessages++;
    }

    std::cout << numSentMessages << " messages is sent to endpoint"
       << destination << std::endl;
    return Aws::WeaverRuntime::Success();
}
```

Python

```
# Message data class
@dataclasses.dataclass
class MessageTickAndId:
    tick: int = 0
    id: int = 0

# send messages
def _send_messages(self, txn):
    tick = api.current_tick(txn)
    num_messages_to_send = 16

    # Fetch the destination MessageEndpoint with the endpoint resolver
    destination = api.utils.resolve_endpoint_from_domain_name_position(
       txn,
       "MySpatialSimulation",
       pos
   )
    Log.debug("Destination_endpoint = %s", destination_endpoint)

   for id in range(num_messages_to_send):
       # Message struct that'll be serialized into payload
        message_tick_and_id = MessageTickAndId(id = id, tick = tick.value)

       # Create the payload out of the struct
```

```
        message_tick_and_id_data = struct.pack(
            '<ii',
            message_tick_and_id.id,
            message_tick_and_id.tick
        )
        payload = api.MessagePayload(list(message_tick_and_id_data))

        # Send the payload to the destination
        Log.debug("Sending message: %s, endpoint: %s",
            message_tick_and_id,
            destination
        )
        api.send_message(
            txn,
            payload,
            destination,
            api.MessageDeliveryType.BestEffort
        )

    Log.info("Sent %s messages to %s", num_messages_to_send, destination)
    return True
```

Receiving messages

SimSpace Weaver delivers messages into a partition's inbound message queue. Use the `ReceiveMessages` API to get a `MessageList` object that contains the messages from the queue. Process each message with the `ExtractMessage` API to get the message data.

Example

C++

```
Result<void> ReceiveMessages(Txn& txn) noexcept
{
     // Fetch all the messages sent to the partition owned by the app
    WEAVERRUNTIME_TRY(auto messages, Api::ReceiveMessages(txn));
    std::cout << "Received" << messages.messages.size() << " messages" << std::endl;
    for (Api::Message& message : messages.messages)
    {
        std::cout << "Received message: " << message << std::endl;

         // Deserialize payload to the message struct
        const MessageTickAndId& receivedMessage
            = Api::Utils::ExtractMessage<MessageTickAndId>(message);
        std::cout << "Received MessageTickAndId, Id: " << receivedMessage.id
            <<", Tick: " << receivedMessage.tick << std::endl;
    }

    return Aws::WeaverRuntime::Success();
}
```

Python

```
# process incoming messages
def _process_incoming_messages(self, txn):
    messages = api.receive_messages(txn)
    for message in messages:
        payload_list = message.payload.data
        payload_bytes = bytes(payload_list)
        message_tick_and_id_data_struct
           = MessageTickAndId(*struct.unpack('<ii', payload_bytes))

        Log.debug("Received message. Header: %s, message: %s",
                    message.header, message_tick_and_id_data_struct)

    Log.info("Received %s messages", len(messages))
    return True
```

Replying to the sender

Every received message contains a message header with information about the message's original sender. You can use the message.header.source_endpoint to send a reply.

Example

C++

```
Result<void> ReceiveMessages(Txn& txn) noexcept
{
     // Fetch all the messages sent to the partition owned by the app
    WEAVERRUNTIME_TRY(auto messages, Api::ReceiveMessages(txn));
    std::cout << "Received" << messages.messages.size() << " messages" << std::endl;
    for (Api::Message& message : messages.messages)
    {
        std::cout << "Received message: " << message << std::endl;

         // Deserialize payload to the message struct
        const MessageTickAndId& receivedMessage
            = Api::Utils::ExtractMessage<MessageTickAndId>(message);
        std::cout << "Received MessageTickAndId, Id: " << receivedMessage.id
            <<", Tick: " << receivedMessage.tick << std::endl;

        // Get the sender endpoint and payload to bounce the message back
        Api::MessageEndpoint& sender = message.header.source_endpoint;
        Api::MessagePayload& payload = message.payload;
        Api::SendMessage(txn, payload, sender);
    }

    return Aws::WeaverRuntime::Success();
}
```

Python

```
# process incoming messages
def _process_incoming_messages(self, txn):
    messages = api.receive_messages(txn)
    for message in messages:
        payload_list = message.payload.data
        payload_bytes = bytes(payload_list)
        message_tick_and_id_data_struct
           = MessageTickAndId(*struct.unpack('<ii', payload_bytes))

        Log.debug("Received message. Header: %s, message: %s",
                    message.header, message_tick_and_id_data_struct)
       # Get the sender endpoint and payload
       # to bounce the message back
       sender = message.header.source_endpoint
       payload = payload_list
       api.send_message(
           txn,
           payload_list,
           sender,
           api.MessageDeliveryType.BestEffort

    Log.info("Received %s messages", len(messages))
    return True
```

When to use messaging

Messaging in SimSpace Weaver offers another pattern for exchanging information between simulation applications. Subscriptions provide a pull mechanism to read data from specific applications or areas of the simulation; messages provide a push mechanism to send data to specific applications or areas of the simulation.

Below are two use cases where its more helpful to push data using messaging rather than pulling or reading data through a subscription.

Example 1: Sending a command to another app to change an entity position

```
// Message struct definition
struct MessageMoveEntity
{
     uint64_t entityId;
    std::array<float, 3> destinationPos;
};

// Create the message
MessageMoveEntity message {45, {236.67, 826.22, 0.0} };

// Create the payload out of the struct
const Api::MessagePayload& payload = Api::Utils::CreateMessagePayload(
    reinterpret_cast<const std::uint8_t*>(&message),
    sizeof(MessageTickAndId)
);

// Grab the MessageEndpoint of the recipient app.
Api::MessageEndpoint destination = ...

// One way is to resolve it from the domain name and position
WEAVERRUNTIME_TRY(
    Api::MessageEndpoint destination,
    Api::Utils::MessageEndpointResolver::ResolveFromPosition(
    txn,
        "MySpatialSimulation",
        Api::Vector2F32 {200.0, 100.0}
    )
);

// Then send the message
```

```
Api::SendMessage(txn, payload, destination);
```

On the receiving side, the app updates the position of the entity and writes it to the State Fabric.

```
Result<void> ReceiveMessages(Txn& txn) noexcept
{
    WEAVERRUNTIME_TRY(auto messages, Api::ReceiveMessages(txn));
    for (Api::Message& message : messages.messages)
    {
        std::cout << "Received message: " << message << std::endl;
         // Deserialize payload to the message struct
        const MessageMoveEntity& receivedMessage
            = Api::Utils::ExtractMessage<MessageMoveEntity>(message);

        ProcessMessage(txn, receivedMessage);
    }

    return Aws::WeaverRuntime::Success();
}

void ProcessMessage(Txn& txn, const MessageMoveEntity& receivedMessage)
{
     // Get the entity corresponding to the entityId
    Entity entity = EntityFromEntityId (receivedMessage.entityId);

    // Update the position and write to StateFabric
    WEAVERRUNTIME_TRY(Api::StoreEntityIndexKey(
            txn,
            entity,
            k_vector3f32TypeId, // type id of the entity
            reinterpret_cast<std::int8_t*>(&receivedMessage.destinationPos),
            sizeof(receivedMessage.destinationPos)));

}
```

Example 2: Sending a create entity message to a spatial app

```
struct WeaverMessage
{
    const Aws::WeaverRuntime::Api::TypeId messageTypeId;
};

const Aws::WeaverRuntime::Api::TypeId k_createEntityMessageTypeId = { 1 };
```

```
struct CreateEntityMessage : WeaverMessage
{
    const Vector3 position;
   const Aws::WeaverRuntime::Api::TypeId typeId;
};


CreateEntityMessage messageData {
    k_createEntityMessageTypeId,
    Vector3{ position.GetX(), position.GetY(), position.GetZ() },
    Api::TypeId { 0 }
}

WEAVERRUNTIME_TRY(Api::MessageEndpoint destination,
 Api::Utils::MessageEndpointResolver::ResolveFromPosition(
    transaction, "MySpatialDomain", DemoFramework::ToVector2F32(position)
));

Api::MessagePayload payload = Api::Utils::CreateMessagePayload(
    reinterpret_cast<const uint8_t*>(&messageData),
    sizeof(CreateEntityMessage));

Api::SendMessage(transaction, payload, destination);
```

On the receiving side, the app creates a new entity in the State Fabric and updates its position.

```
Result<void> ReceiveMessages(Txn& txn) noexcept
{
    WEAVERRUNTIME_TRY(auto messageList, Api::ReceiveMessages(transaction));
    WEAVERRUNTIME_TRY(auto tick, Api::CurrentTick(transaction));
    for (auto& message : messageList.messages)
    {
        // cast to base WeaverMessage type to determine MessageTypeId
        WeaverMessage weaverMessageBase =
 Api::Utils::ExtractMessage<WeaverMessage>(message);
        if (weaverMessageBase.messageTypeId == k_createEntityMessageTypeId)
        {
            CreateEntityMessage createEntityMessageData =
                Api::Utils::ExtractMessage<CreateEntityMessage>(message);
        CreateActorFromMessage(transaction, createEntityMessageData));
        }
        else if (weaverMessageBase.messageTypeId == k_tickAndIdMessageTypeId)
        {
            ...
        }
    }
}

void ProcessMessage(Txn& txn, const CreateEntityMessage& receivedMessage)
{
    // Create entity
    WEAVERRUNTIME_TRY(
        Api::Entity entity,
        Api::CreateEntity(transaction, receivedMessage.typeId)
    );

    // Update the position and write to StateFabric
    WEAVERRUNTIME_TRY(Api::StoreEntityIndexKey(
        transaction,
        entity,
        receivedMessage.typeId,
        reinterpret_cast<std::int8_t*>(&receivedMessage.position),
        sizeof(receivedMessage.position)));
}
```

Tips when working with messaging

Resolve an endpoint from a position or app name

You can use the `AllPartitions` function to get the spatial boundaries and a domain ID that you need to determine message partition IDs and message destinations. However, if you know the position you want to message, but not its Partition ID, you can use the MessageEndpointResolver function.

```
/**
* Resolves MessageEndpoint's from various inputs
**/
class MessageEndpointResolver
{
    public:
    /**
    * Resolves MessageEndpoint from position information
    **/
    Result<MessageEndpoint> ResolveEndpointFromPosition(
        const DomainId& domainId,
        const weaver_vec3_f32_t& pos);

    /**
    * Resolves MessageEndpoint from custom app name
    **/
    Result<MessageEndpoint> ResolveEndpointFromCustomAppName(
        const DomainId& domainId,
        const char* agentName);
};
```

Serializing and deserializing the message payload

You can use the following functions to create and read message payloads. For more information, see MessagingUtils.h in the app SDK library on your local system.

```
/**
* Utility function to create MessagePayload from a custom type
*
* @return The @c MessagePayload.
*/
template <class T>
AWS_WEAVERRUNTIME_API MessagePayload CreateMessagePayload(const T& message) noexcept
{
    const std::uint8_t* raw_data = reinterpret_cast<const std::uint8_t*>(&message);

    MessagePayload payload;
    std::move(raw_data, raw_data + sizeof(T), std::back_inserter(payload.data));

    return payload;
}

/**
* Utility function to convert MessagePayload to custom type
*/
template <class T>
AWS_WEAVERRUNTIME_API T ExtractMessage(const MessagePayload& payload) noexcept
{
    return *reinterpret_cast<const T*>(payload.data.data());
}
```

Messaging errors and troubleshooting

You might experience the following errors when you use the messaging APIs.

Endpoint resolution errors

These errors can occur before an app sends a message.

Domain name check

Sending a message to an invalid endpoint results in the following error:

```
ManifoldError::InvalidArgument {"No DomainId found for the given domain name" }
```

This can happen when you try to send a message to a custom app and that custom app hasn't joined the simulation yet. Use the `DescribeSimulation` API to make sure your custom app has launched before you send a message to it. This behavior is the same in SimSpace Weaver Local and the AWS Cloud.

Position check

Trying to resolve an endpoint with a valid domain name but an invalid position results in the following error.

```
ManifoldError::InvalidArgument {"Could not resolve endpoint from domain : DomainId
 { value: domain-id } and position: Vector2F32 { x: x-position, y: y-position}" }
```

We suggest using the `MessageEndpointResolver` in the `MessageUtils` library contained in the SimSpace Weaver app SDK.

Message sending errors

The following errors can occur as an app sends a message.

Message sending limit per app, per tick, exceeded

The current limit for the number of messages that can be sent per app per simulation tick is 128. Subsequent calls on the same tick will fail with the following error:

```
ManifoldError::CapacityExceeded {"At Max Outgoing Message capacity: {}", 128}
```

SimSpace Weaver tries to send unsent messages on the next tick. Lower the send frequency to resolve this problem. Combine message payloads that are smaller than the 256 byte limit to lower the number of outbound messages.

This behavior is the same in SimSpace Weaver Local and in the AWS Cloud.

Message payload size limit exceeded

The current limit for message payload size is 256 bytes in both SimSpace Weaver Local and in the AWS Cloud. Sending a message with a payload larger than 256 bytes results in the following error:

```
ManifoldError::CapacityExceeded {"Message data too large! Max size: {}", 256}
```

SimSpace Weaver checks each message and only rejects those that exceed the limit. For example, if your app tries to send 10 messages and 1 fails the check, only that 1 message is rejected. SimSpace Weaver sends the other 9 messages.

This behavior is the same in SimSpace Weaver Local and the AWS Cloud.

Destination is the same as source

Apps can't send messages to partitions they own. You get the following error if an app sends a message to a partition it owns.

```
ManifoldError::InvalidArgument { "Destination is the same as source" }
```

This behavior is the same in SimSpace Weaver Local and the AWS Cloud.

Best effort messaging

SimSpace Weaver doesn't guarantee message delivery. The service will try to complete delivery of messages on the subsequent simulation tick, but messages might get lost or be delayed.

Best practices when working with SimSpace Weaver

We recommend the following best practices when working with SimSpace Weaver.

Topics

Set up billing alarms

It's easy to provision resources in AWS and leave them running all the time, even when they aren't needed anymore. This can result in runaway costs that can be a surprise when you get your bill. You can configure an alarm in Amazon CloudWatch that will trigger and notify you when your costs exceed a threshold that you set. You can examine your costs using cost management tools. For more information, see:

- Create a billing alarm to monitor your estimated AWS charges
- What is AWS Cost Management

Use SimSpace Weaver Local

We recommend that you use SimSpace Weaver Local to develop and test your simulations before uploading them to the SimSpace Weaver service in the AWS Cloud. The benefits of developing with SimSpace Weaver Local include:

- No need to wait for large uploads
- No limit on the number of local simulations you can create
- You aren't charged for the compute time on your local computer
- Direct access to console output from your apps
- Modify, rebuild, and restart your local simulation without having to recreate it in the AWS Cloud

Stop simulations that you don't need

You get billing charges for a simulation while it is running. You must stop a simulation to stop getting charges for it. Running simulations also count towards your quota for the maximum number of simulations. A running simulation that has logging configured can also generate large amounts of logs, which you also get billing charges for. You should stop any simulation that you don't need in order to stop getting additional charges.

Important

Stopping the simulation clock doesn't stop the simulation, the clock just stops publishing ticks to your apps. You can't restart a simulation after you stop it.

Delete resources that you don't need

Each simulation that you create in SimSpace Weaver also creates resources in other AWS services. You can get billing charges for resources and data in these other services. Running and failed simulations count towards your quota for the maximum number of simulations. You should delete unneeded failed simulations so that you can start new simulations. When you delete a simulation, resources for your simulation that exist in other AWS services might not be deleted. For example, any simulation log data in Amazon CloudWatch Logs will remain there until you delete it. You will get billing charges for that log data. You should cleanup all the associated resources for your simulations if you don't need them anymore.

Have backups

It's a good idea to have backups and backup plans for everything. You shouldn’t assume that just because your data is in AWS that you don't have to back it up. You must create your own system if you need to back up your simulation state. Consider using multiple AWS Regions and having a plan in place to be able to quickly switch your production workload to another AWS Region if you need to. For more information about AWS Regions that support SimSpace Weaver, see SimSpace Weaver endpoints and quotas.

Security in AWS SimSpace Weaver

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS SimSpace Weaver, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using SimSpace Weaver. The following topics show you how to configure SimSpace Weaver to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your SimSpace Weaver resources.

Topics

Data protection in AWS SimSpace Weaver

The AWS shared responsibility model applies to data protection in AWS SimSpace Weaver. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with SimSpace Weaver or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

Data is considered *at rest* when it is located in non-volatile (persistent) data storage, such as a disk. Data located in volatile data storage, such as memory and registers, is not considered to be *at rest*.

When you use SimSpace Weaver, the only data at rest are:

- Apps and schemas that you upload to Amazon Simple Storage Service (Amazon S3)
- Simulation log data stored in Amazon CloudWatch

Other data that SimSpace Weaver uses internally doesn't persist after you stop your simulation.

To learn how to encrypt your data at rest, see:

- Encrypt your data in Amazon S3
- Encrypt your log data

Encryption in transit

Your connections to the SimSpace Weaver API through the AWS Command Line Interface (AWS CLI), AWS SDK, and SimSpace Weaver app SDK, use TLS encryption with the Signature Version 4 signing process. AWS manages authentication using the IAM-defined access policies for the security credentials you use to connect.

Internally, SimSpace Weaver uses TLS to connect to other AWS services that it uses.

Important

Communications between your apps and their clients don't involve SimSpace Weaver. It's your responsibility to encrypt communications with simulation clients, if required. We recommend that you create a solution to encrypt all data in transit across client connections.

To learn more about AWS services that can support your encryption solutions, see the AWS Security Blog.

Inter-network traffic privacy

SimSpace Weaver compute resources reside within 1 Amazon VPC shared by all SimSpace Weaver customers. All internal SimSpace Weaver service traffic stays within the AWS network and doesn't travel across the internet. Communication between simulation clients and your apps travels across the internet.

Identity and Access Management for AWS SimSpace Weaver

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use SimSpace Weaver resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in SimSpace Weaver.

Service user – If you use the SimSpace Weaver service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more SimSpace Weaver features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in SimSpace Weaver, see Troubleshooting AWS SimSpace Weaver identity and access.

Service administrator – If you're in charge of SimSpace Weaver resources at your company, you probably have full access to SimSpace Weaver. It's your job to determine which SimSpace Weaver features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with SimSpace Weaver, see How AWS SimSpace Weaver works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to SimSpace Weaver. To view example SimSpace Weaver identity-based policies that you can use in IAM, see Identity-based policy examples for AWS SimSpace Weaver.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

  - **Service role –** A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role –** A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries –** A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs) –** SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies –** Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS SimSpace Weaver works with IAM

Before you use IAM to manage access to SimSpace Weaver, learn what IAM features are available to use with SimSpace Weaver.

IAM features you can use with AWS SimSpace Weaver

IAM feature	SimSpace Weaver support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how SimSpace Weaver and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for SimSpace Weaver

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for SimSpace Weaver

To view examples of SimSpace Weaver identity-based policies, see Identity-based policy examples for AWS SimSpace Weaver.

Resource-based policies within SimSpace Weaver

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for SimSpace Weaver

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of SimSpace Weaver actions, see Actions defined by AWS SimSpace Weaver in the *Service Authorization Reference*.

Policy actions in SimSpace Weaver use the following prefix before the action:

```
simspaceweaver
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "simspaceweaver:action1",
      "simspaceweaver:action2"
         ]
```

To view examples of SimSpace Weaver identity-based policies, see Identity-based policy examples for AWS SimSpace Weaver.

Policy resources for SimSpace Weaver

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of SimSpace Weaver resource types and their ARNs, see Resources defined by AWS SimSpace Weaver in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS SimSpace Weaver.

To view examples of SimSpace Weaver identity-based policies, see Identity-based policy examples for AWS SimSpace Weaver.

Policy condition keys for SimSpace Weaver

Supports service-specific policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of SimSpace Weaver condition keys, see Condition keys for AWS SimSpace Weaver in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS SimSpace Weaver.

To view examples of SimSpace Weaver identity-based policies, see Identity-based policy examples for AWS SimSpace Weaver.

Access control lists (ACLs) in SimSpace Weaver

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with SimSpace Weaver

Supports ABAC (tags in policies)	Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with SimSpace Weaver

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for SimSpace Weaver

Supports forward access sessions (FAS)	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for SimSpace Weaver

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break SimSpace Weaver functionality. Edit service roles only when SimSpace Weaver provides guidance to do so.

The SimSpace Weaver app SDK scripts use an AWS CloudFormation template to create resources in other AWS services to support your simulation. One of these resources is the *app role* for your simulation. SimSpace Weaver assumes the app role to perform actions in your AWS account on your behalf, such as to write log data to CloudWatch Logs. For more information about the app role, see Permissions that SimSpace Weaver creates for you.

Service-linked roles for SimSpace Weaver

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS SimSpace Weaver

By default, users and roles don't have permission to create or modify SimSpace Weaver resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by SimSpace Weaver, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS SimSpace Weaver in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete SimSpace Weaver resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the SimSpace Weaver console

To access the AWS SimSpace Weaver console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the SimSpace Weaver resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the SimSpace Weaver console, also attach the SimSpace Weaver *`ConsoleAccess`* or *`ReadOnly`* AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow users to create and run simulations

This example IAM policy provides the basic permissions required to create and run simulations in SimSpace Weaver.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateAndRunSimulations",
            "Effect": "Allow",
            "Action": [
                "simspaceweaver:*",
                "iam:GetRole",
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:UpdateRole",
  "iam:CreatePolicy",
  "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "iam:GetRolePolicy",
                "iam:DeleteRolePolicy",
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:PutBucketPolicy",
                "s3:CreateBucket",
                "s3:ListBucket",
                "s3:PutEncryptionConfiguration",
                "s3:DeleteBucket",
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack",
                "cloudformation:DescribeStacks"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PassAppRoleToSimSpaceWeaver",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "simspaceweaver.amazonaws.com"
                }
            }
        }
    ]
}
```

Permissions that SimSpace Weaver creates for you

When you create a SimSpace Weaver project, the service will create an AWS Identity and Access Management (IAM) role with the name `weaver-project-name-app-role` and an IAM trust policy. The trust policy allows SimSpace Weaver to assume the role so that it can perform operations for you.

App role permissions policy

The simulation app role has the following permissions policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": "*"
        }
    ]
}
```

App role trust policy

SimSpace Weaver adds a trust relationship to the simulation app role as a trust policy. SimSpace Weaver creates a trust policy for each simulation, similar to the following example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "simspaceweaver.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn":
     "arn:aws:simspaceweaver:us-west-2:111122223333:simulation/MySimName*"
        }
      }
    }
  ]
}
```

Note

In this example, the account number is `111122223333` and the simulation name is `MySimName`. These values are different in your trust policies.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can trick a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it shouldn't otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS SimSpace Weaver gives another service to the resource. If the `aws:SourceArn` value doesn't contain the account ID, such as an Amazon S3 bucket Amazon Resource Name (ARN), you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must use the extension's ARN.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the extension or if you are specifying multiple extensions, use the `aws:SourceArn` global context condition key with wildcards (`*`) for the unknown portions of the ARN. For example, `arn:aws:`*`simspaceweaver`*`:*:`*`111122223333`*`:*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in SimSpace Weaver to prevent the confused deputy problem. This policy will only permit SimSpace Weaver to assume the role when the request comes from the specified source account, and provided with the specified ARN. In this case, SimSpace Weaver can only assume the role for requests from simulations in the requestor's own account (*`111122223333`*), and only in the specified Region (*`us-west-2`*).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "simspaceweaver.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "StringLike": {
          "aws:SourceArn": "arn:aws:simspaceweaver:us-west-2:111122223333:simulation/*"
        }
      }
    }
  ]
}
```

A more secure way to write this policy is to include the simulation name in the `aws:SourceArn`, as shown in the following example, which restricts the policy to a simulation named `MyProjectSimulation_22-10-04_22_10_15`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "simspaceweaver.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "StringLike": {
          "aws:SourceArn": "arn:aws:simspaceweaver:us-west-2:111122223333:simulation/
MyProjectSimulation_22-10-04_22_10_15"
        }
      }
    }
  ]
}
```

When your `aws:SourceArn` explicitly includes an account number, you can leave out the `Condition` element test for the `aws:SourceAccount` (see the IAM User Guide for more information), such as in the following simplified policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "simspaceweaver.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:simspaceweaver:us-west-2:111122223333:simulation/
MyProjectSimulation_22-10-04_22_10_15"
        }
      }
    }
  ]
}
```

Troubleshooting AWS SimSpace Weaver identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with SimSpace Weaver and IAM.

Topics

I am not authorized to perform an action in SimSpace Weaver

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `simspaceweaver:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 simspaceweaver:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-widget`* resource using the `simspaceweaver:`*`GetWidget`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to SimSpace Weaver.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in SimSpace Weaver. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important

Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your AWS account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access SimSpace Weaver

To allow others to access SimSpace Weaver, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in SimSpace Weaver.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my SimSpace Weaver resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether SimSpace Weaver supports these features, see How AWS SimSpace Weaver works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Security event logging and monitoring in AWS SimSpace Weaver

Monitoring is an important part of maintaining the reliability, availability, and performance of SimSpace Weaver and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs.

AWS and SimSpace Weaver provide several tools for monitoring your simulation resources and responding to potential incidents.

Logs in Amazon CloudWatch

SimSpace Weaver stores its logs in CloudWatch. You can use these logs to monitor events in your simulation (such as starting and stopping apps) as well as for debugging. For more information, see SimSpace Weaver logs in Amazon CloudWatch Logs.

Amazon CloudWatch Alarms

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms are triggered when their state changes and is maintained for a specified number of periods, not by being in a particular state. For more information, see Monitoring SimSpace Weaver with Amazon CloudWatch.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in SimSpace Weaver. Using the information collected by CloudTrail, you can determine the request that was made to SimSpace Weaver, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Logging AWS SimSpace Weaver API calls using AWS CloudTrail.

Compliance Validation for AWS SimSpace Weaver

SimSpace Weaver is not in scope of any AWS compliance programs.

Third-party auditors assess the security and compliance of other AWS services as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- AWS Audit Manager – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in AWS SimSpace Weaver

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure Security in AWS SimSpace Weaver

As a managed service, AWS SimSpace Weaver is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud

Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well‐Architected Framework*.

You use AWS published API calls to access SimSpace Weaver through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Network connectivity security model

Your simulations run on compute instances within an Amazon VPC located within an AWS Region that you select. An Amazon VPC is a virtual network in the AWS Cloud, which isolates infrastructure by workload or organizational entity. Communications between compute instances within the Amazon VPC stay within the AWS network and don't travel over the internet. Some internal service communication crosses the internet, and is encrypted. Simulations for all customers running in the same AWS Region share the same Amazon VPC. Simulations for different customers use separate compute instances within the same Amazon VPC.

Communications between your simulation clients and your simulations running in SimSpace Weaver travel over the internet. SimSpace Weaver does not handle these connections. It is your responsibility to secure your client connections.

Your connections to the SimSpace Weaver service cross the internet and are encrypted. This includes connections using the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS software development kits (SDK), and the SimSpace Weaver app SDK.

Configuration and vulnerability analysis in AWS SimSpace Weaver

Configuration and IT controls are a shared responsibility between AWS and you. For more information, see the AWS shared responsibility model. AWS handles basic security tasks for the underlying infrastructure, such as patching the operating system on compute instances, firewall configuration, and AWS infrastructure disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see Best Practices for Security, Identity, and Compliance.

You are responsible for the security of your simulation software:

- Maintain your app code, including updates and security patches.
- Authenticate and encrypt communication between your simulation clients and the apps they connect to.
- Update your simulations to use the latest SDK versions, including the AWS SDK and SimSpace Weaver app SDK.

Note

SimSpace Weaver doesn't support updates to apps in a running simulation. If you need to update your apps, you must stop and delete the simulation, then create a new simulation with the updated app code. We recommend that you save the simulation state in an external data store so that you can restore it if you need to recreate the simulation.

Security best practices for SimSpace Weaver

This section describes security best practices that are specific to SimSpace Weaver. To learn more about security best practices in AWS, see Best Practices for Security, Identity, and Compliance.

Topics

Encrypt communications between your apps and their clients

SimSpace Weaver doesn't manage communications between your apps and their clients. You should implement some form of authentication and encryption for client sessions.

Periodically backup your simulation state

SimSpace Weaver doesn't save your simulation state. Simulations that are stopped (as a result of an API call, console option, or system crash) do not save their state and have no inherent way to recover them. Stopped simulations cannot be restarted. The only way to perform the equivalent of a restart is to recreate your simulation using the same configuration and data. You can use backups of your simulation state to initialize the new simulation. AWS offers highly reliable and available cloud storage and database services that you can use to save your simulation state.

Maintain your apps and SDKs

Maintain your apps, local installations of the AWS software development kits (SDKs), and the SimSpace Weaver app SDK. You can download and install new versions of the AWS SDKs. Test new versions of the SimSpace Weaver app SDK with non-production app builds to ensure that your apps continue to run as expected. You cannot update your apps in a running simulation. To update your apps:

1. Update and test the app code locally (or in a test environment).
2. Stop changing your simulation state and save it (if necessary).
3. Stop your simulation (once stopped, it cannot be restarted).
4. Delete your simulation (stopped simulations that aren't deleted count towards your service limits).
5. Recreate your simulation with the same configuration and the updated app code.
6. Initialize your simulation using saved state data (if available).
7. Start your new simulation.

Note

A new simulation created with the same configuration is separate from the old simulation. It will have a new simulation ID and send logs to a new log stream in Amazon CloudWatch.

Logging and monitoring in SimSpace Weaver

Monitoring is an important part of maintaining the reliability, availability, and performance of SimSpace Weaver and your other AWS solutions. AWS provides the following monitoring tools to watch SimSpace Weaver, report when something is wrong, and take automatic actions when appropriate:

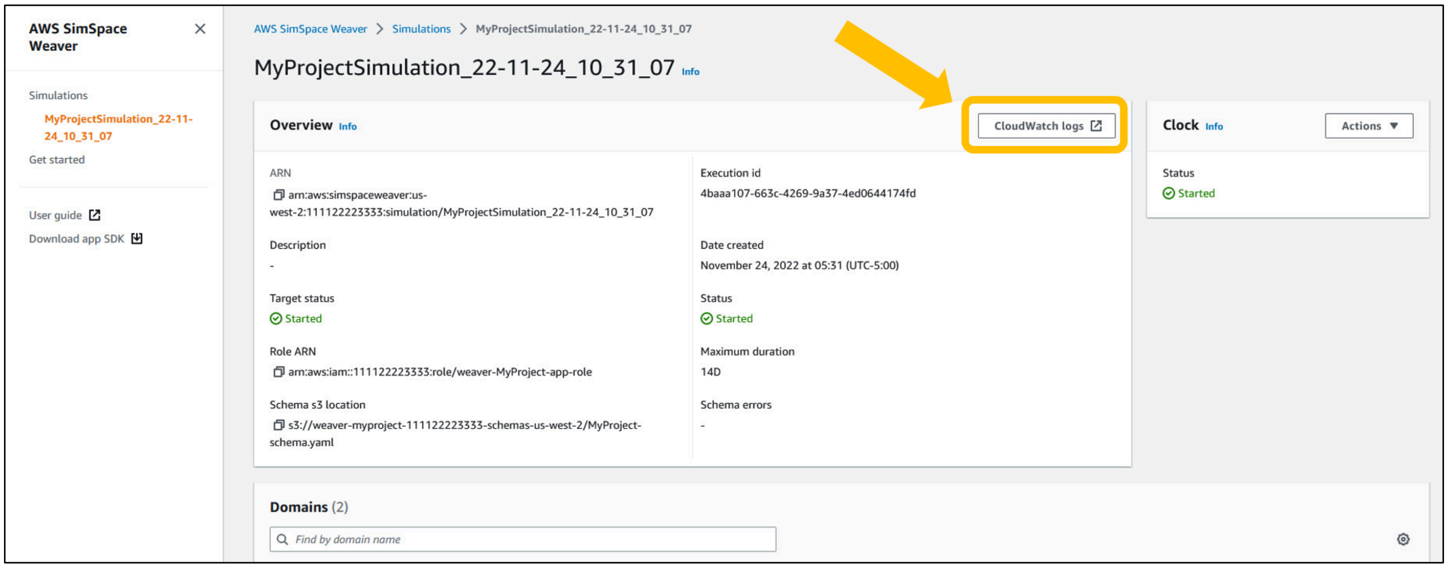
- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log data from your SimSpace Weaver workers, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log data and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

Topics

SimSpace Weaver logs in Amazon CloudWatch Logs

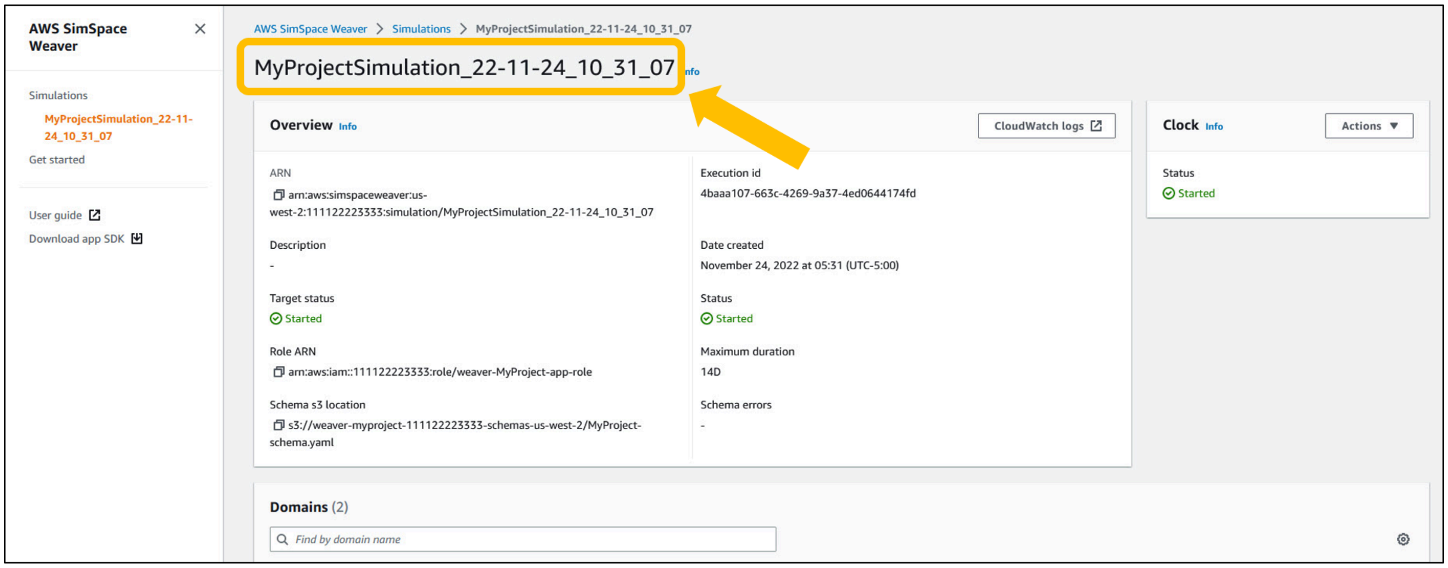
Accessing your SimSpace Weaver logs

All the logs generated from your SimSpace Weaver simulations are stored in Amazon CloudWatch Logs. To access your logs, you can use the **CloudWatch logs** button in the **Overview** pane of your simulation in the SimSpace Weaver console, which will take you directly to the logs for that specific simulation.



ARN	Execution id
arn:aws:simspaceweaver:us-west-2:111122223333:simulation/MyProjectSimulation_22-11-24_10_31_07	4baaa107-663c-4269-9a37-4ed0644174fd
Description	Date created
-	November 24, 2022 at 05:31 (UTC-5:00)
Target status	Status
Started	Started
Role ARN	Maximum duration
arn:aws:iam::111122223333:role/weaver-MyProject-app-role	14D
Schema s3 location	Schema errors
s3://weaver-myproject-111122223333-schemas-us-west-2/MyProject-schema.yaml	-

You can also access the logs through the CloudWatch console. You will need the name of your simulation in order to search for its logs.



ARN	Execution id
arn:aws:simspaceweaver:us-west-2:111122223333:simulation/MyProjectSimulation_22-11-24_10_31_07	4baaa107-663c-4269-9a37-4ed0644174fd
Description	Date created
-	November 24, 2022 at 05:31 (UTC-5:00)
Target status	Status
Started	Started
Role ARN	Maximum duration
arn:aws:iam::111122223333:role/weaver-MyProject-app-role	14D
Schema s3 location	Schema errors
s3://weaver-myproject-111122223333-schemas-us-west-2/MyProject-schema.yaml	-

SimSpace Weaver logs

SimSpace Weaver writes simulation management messages and the console output from your apps to Amazon CloudWatch Logs. For more information on working with logs, see Working with log groups and log streams in the *Amazon CloudWatch Logs User Guide*.

Each simulation that you create has its own log group in CloudWatch Logs. The name of the log group is specified in the simulation schema. In the following schema snippet, the value of `log_destination_service` is `logs`. This means that the value of `log_destination_resource_name` is the name of a log group. In this case, the log group is `MySimulationLogs`.

```
simulation_properties:
  log_destination_service: "logs"
  log_destination_resource_name: "MySimulationLogs"
  default_entity_index_key_type: "Vector3<f32>"
```

You can also use the **DescribeSimulation** API to find the name of the log group for simulation after you start it.

```
aws simspaceweaver describe-simulation --simulation simulation-name
```

The following example shows the part of the output from **DescribeSimulation** that describes the logging configuration. The name of the log group is shown at the end of the `LogGroupArn`.

```
    "LoggingConfiguration": {
        "Destinations": [
            {
                "CloudWatchLogsLogGroup": {
                    "LogGroupArn": "arn:aws:logs:us-west-2:111122223333:log-
group:MySimulationLogs"
                }
            }
        ]
    },
```

Each simulation log group contains several log streams:

- **Management log stream** – simulation management messages produced by the SimSpace Weaver service.

  ```
  /sim/management
  ```

- **Errors log stream** – error messages produced by the SimSpace Weaver service. This log stream only exists if there are errors. SimSpace Weaver stores errors written by your apps in their own app log streams (see the following log streams).

```
/sim/errors
```

- **Spatial app log streams** (1 for each spatial app on each worker) – console output produced by spatial apps. Each spatial app writes to its own log stream. The *spatial-app-id* is all characters after the trailing slash at the end of the *worker-id*.

```
/domain/spatial-domain-name/app/worker-worker-id/spatial-app-id
```

- **Custom app log streams** (1 for each custom app instance) – console output produced by custom apps. Each custom app instance writes to its own log stream.

```
/domain/custom-domain-name/app/custom-app-name/random-id
```

- **Service app log streams** (1 for each service app instance) – console output produced by service apps. Each service app writes to its own log stream. The *service-app-id* is all characters after the trailing slash at the end of the *service-app-name*.

```
/domain/service-domain-name/app/service-app-name/service-app-id
```

Monitoring SimSpace Weaver with Amazon CloudWatch

You can monitor SimSpace Weaver using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

The SimSpace Weaver service reports the following metrics in the `AWS/simspaceweaver` namespace.

SimSpace Weaver metrics at the account level

The SimSpace Weaver namespace includes the following metrics related to activity at the AWS account level.

Metric	Description
`SimulationCount`	The number of simulations for the current account. Units: Count Dimensions: none Statistics: Average, Minimum, Maximum

Logging AWS SimSpace Weaver API calls using AWS CloudTrail

AWS SimSpace Weaver is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in SimSpace Weaver. CloudTrail captures all API calls for SimSpace Weaver as events. The calls captured include calls from the SimSpace Weaver console and code calls to the SimSpace Weaver API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for SimSpace Weaver. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to SimSpace Weaver, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

SimSpace Weaver information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in SimSpace Weaver, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for SimSpace Weaver, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All SimSpace Weaver actions are logged by CloudTrail and are documented in the AWS SimSpace Weaver API Reference. For example, calls to the `ListSimulations`, `DescribeSimulation` and `DeleteSimulation` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding SimSpace Weaver log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, such as the date and time of the action, request parameters, and other details. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `ListSimulations` action.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:aws-console-signin-utils",
        "arn": "arn:aws:sts::111122223333:assumed-role/ConsoleSigninRole/aws-console-
signin-utils",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/ConsoleSigninRole",
                "accountId": "111122223333",
                "userName": "ConsoleSigninRole"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-02-14T15:57:02Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-02-14T15:57:08Z",
    "eventSource": "simspaceweaver.amazonaws.com",
    "eventName": "ListSimulations",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.10",
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/86.0.4240.0 Safari/537.36",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "1234abcd-1234-5678-abcd-12345abcd123",
    "eventID": "5678abcd-5678-1234-ab12-123abc123abc",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

SimSpace Weaver endpoints and quotas

The following tables describe the service endpoints and service quotas for SimSpace Weaver. Service quotas, also referred to as *limits*, are the maximum number of service resources or operations for your AWS account. For more information, see AWS service quotas in the *AWS General Reference*.

Service endpoints

Region name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	simspaceweaver.us-east-1.amazonaws.com	HTTPS
US East (Ohio)	us-east-2	simspaceweaver.us-east-2.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	simspaceweaver.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	simspaceweaver.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	simspaceweaver.ap-southeast-2.amazonaws.com	HTTPS
Europe (Stockholm)	eu-north-1	simspaceweaver.eu-north-1.amazonaws.com	HTTPS

Region name	Region	Endpoint	Protocol
Europe (Frankfurt)	eu-central-1	simspaceweaver.eu-central-1.amazonaws.com	HTTPS
Europe (Ireland)	eu-west-1	simspaceweaver.eu-west-1.amazonaws.com	HTTPS
AWS GovCloud (US-East)	us-gov-east-1	simspaceweaver.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud (US-West)	us-gov-west-1	simspaceweaver.us-gov-west-1.amazonaws.com	HTTPS

Service quotas

Name	Default	Adjuse	Description
Compute resource units for each app	Each supported Region: 4	No	The maximum number of compute resource units that you can allocate to each app.
Compute resource units for each worker	Each supported Region: 17	No	The number of compute resource units available for each worker.
Data fields for each entity	Each supported Region: 7	No	The maximum number of data (non-index) fields that an entity can have.

Name	Default	Adjus e	Description
Entities in a partition	Each supported Region: 8,192	No	The maximum number of entities in 1 partition.
Entity data field size	Each supported Region: 1,024 Bytes	No	The maximum size of a data (non-index) field of an entity.
Entity transfers between workers	Each supported Region: 25	No	The maximum number of entity transfers between workers, for each partition and each tick.
Entity transfers on the same worker	Each supported Region: 500	No	The maximum number of entity transfers on the same worker, for each partition and each tick.
Index fields for each entity	Each supported Region: 1	No	The maximum number of index fields that an entity can have.
Largest maximum duration (in days) for a simulation	Each supported Region: 14	No	The largest number of days that you can specify as the maximum duration for a simulatio n. All simulations have a maximum duration, even if you dont specify the value. A simulatio n automatically stops when it reaches its maximum duration.

Name	Default	Adjustable	Description
Memory for each compute resource unit	Each supported Region: 1 Gigabytes	No	The amount of random-access memory (RAM) that an app gets for each compute resource unit.
Remote subscriptions for each worker	Each supported Region: 24	No	The maximum number of remote subscriptions for each worker.
Simulation count	Each supported Region: 2	Yes	The maximum number of simulations with a target status of STARTED in your account. You can request a quota increase up to 10.
Workers for a simulation	Each supported Region: 2	Yes	The maximum number of workers that you can assign to 1 simulation. You can request a quota increase up to 10.
vCPUs for each compute resource unit	Each supported Region: 2	No	The number of virtual central processing units (vCPUs) that an app gets for each compute resource unit.

Messaging quotas

The following quotas apply to app to app messaging for SimSpace Weaver Local and in the AWS Cloud.

Name	Default	Adjustable	Description
Maximum message size	Each supported Region: 256 bytes	No	The maximum size of a message payload.
Maximum message send rate	Each supported Region: 128	No	The maximum number of messages that each app can send per tick.

Clock rates

The simulation schema specifies the *clock rate* (also called the *tick rate*) for a simulation. The following table specifies the valid clock rates that you can use.

Name	Valid values	Description
Clock rate	Each supported Region: "10", "15", "30", "unlimited"	The valid clock rates for a simulation.
Clock rate (versions 1.13 and 1.12)	Each supported Region: 10, 15, 30	The valid clock rates for a simulation.

Service quotas for SimSpace Weaver Local

The following service quotas apply to SimSpace Weaver Local only. All other quotas also apply to SimSpace Weaver Local.

Name	Default	Adjustable	Description
Maximum partitions	SimSpace Weaver Local: 24	No	The maximum number of partitions for a simulation.
Maximum apps	SimSpace Weaver Local: 24	No	The maximum total number of apps

Name	Default	Adjustable	Description
			(of any type) for a simulation.
Maximum domains	SimSpace Weaver Local: 24	No	The maximum total number of domains (of any type) for a simulation.
Entities per partition	SimSpace Weaver Local: 4,096	No	The maximum number of entities in each partition.
Fields per entity	SimSpace Weaver Local: 8	No	The maximum number of fields for each entity.
Field size	SimSpace Weaver Local: 1024 bytes	No	The maximum size of an entity field.

Troubleshooting in SimSpace Weaver

Topics

AssumeRoleAccessDenied

You might receive the following error if your simulation fails to start:

```
Unable to assume role arn:aws:iam::111122223333:role/weaver-project-name-app-role;
 verify the role exists and has trust policy on SimSpace Weaver
```

You can receive this error if one of the following is true about the AWS Identity and Access Management (IAM) role for your simulation:

- The Amazon Resource Name (ARN) refers to an IAM role that doesn't exist.
- The trust policy for the IAM role that doesn't allow the name of the new simulation to assume the role.

Check to make sure that the role exists. If the role exists, check your trust policy for the role. The `aws:SourceArn` in following example trust policy only allows a simulation (in account 111122223333) whose name begins with `MySimulation` to assume the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "simspaceweaver.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:simspaceweaver:us-
west-2:111122223333:simulation/MySimulation*"
                }
            }
        }
    ]
}
```

To allow another simulation whose name begins with `MyOtherSimulation` to assume the role, the trust policy must be modified as in the following edited example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "simspaceweaver.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": [
                        aws:SourceArn": "arn:aws:simspaceweaver:us-
west-2:111122223333:simulation/MySimulation*",
                        aws:SourceArn": "arn:aws:simspaceweaver:us-
west-2:111122223333:simulation/MyOtherSimulation*"
                        ]
                }
            }
        }
    ]
}
```

InvalidBucketName

You might receive the following error while creating a project:

```
An error occurred (InvalidBucketName) when calling the CreateBucket operation: The
 specified bucket is not valid.
```

You received this error because the name that SimSpace Weaver passed to Amazon Simple Storage Service (Amazon S3) violated bucket naming rules (for more information, see Bucket naming rules in the *Amazon Simple Storage Service User Guide*).

The `create-project` script in the SimSpace Weaver app SDK creates bucket names using the project name that you provide to the script. The bucket names use the following formats:

- Version 1.13.x or later
  - `weaver-`*`lowercase-project-name`*`-`*`account-number`*`-`*`region`*
- Version 1.12.x
  - `weaver-`*`lowercase-project-name`*`-`*`account-number`*`-app-zips-`*`region`*
  - `weaver-`*`lowercase-project-name`*`-`*`account-number`*`-schemas-`*`region`*

For example, given the following project properties:

- Project name: `MyProject`
- AWS account number: `111122223333`
- AWS Region: `us-west-2`

The project would have the following buckets:

- Version 1.13.x or later
  - `weaver-myproject-111122223333-us-west-2`
- Version 1.12.x
  - `weaver-myproject-111122223333-app-zips-us-west-2`
  - `weaver-myproject-111122223333-schemas-us-west-2`

Your project name must not violate the Amazon S3 naming rules. You must also use a project name that is short enough so that the bucket names created by the `create-project` script do not exceed the name length limit for Amazon S3 buckets.

ServiceQuotaExceededException

You might receive the following error when you start a simulation:

```
An error occurred (ServiceQuotaExceededException) when calling the StartSimulation
 operation: Failed to start simulation due to: simulation quota has already been
 reached.
```

You will receive this error if you try to start a new simulation but your account currently has the maximum number of simulations with a target status of STARTED. This includes running simulations, failed simulations, and simulations that stopped because they reached their maximum duration. You can delete a stopped or failed simulation to allow you to start a new simulation. If all of your simulations are running, you can stop and delete a running simulation. You can also request an increase to your service quotas if you aren't already at the request limit.

TooManyBuckets

You might receive the following error while creating a project:

```
An error occurred (TooManyBuckets) when calling the CreateBucket operation: You have
 attempted to create more buckets than allowed.
```

Amazon Simple Storage Service (Amazon S3) has a limit on the number of buckets that you can have in your AWS account (for more information, see Bucket restrictions and limitations in the *Amazon Simple Storage Service User Guide*).

You must do one of the following before you can continue:

- Delete 2 or more existing Amazon S3 buckets that you don't need.
- Request an Amazon S3 limit increase (for more information, see Bucket restrictions and limitations in the *Amazon Simple Storage Service User Guide*).
- Use a different AWS account.

Note

The `DeleteSimulation` API in SimSpace Weaver doesn't delete Amazon S3 resources associated with your simulation. We recommend that you remove all resources associated with your simulations when you no longer need them.

Permission denied during simulation start

When you start a simulation, you might get an error message indicating that permission was denied or that there was an error accessing your app artifacts. This problem can occur when you specify Amazon S3 buckets for your simulation that SimSpace Weaver didn't create for you (either through the console or the SimSpace Weaver app SDK scripts).

The following situations are the most likely root causes:

- **The service doesn't have permission to access one or more of the Amazon S3 buckets you specified in your simulation schema** – check your app role permissions policy, Amazon S3 bucket policies, and Amazon S3 bucket permissions to make sure that `simspaceweaver.amazonaws.com` has the correct permissions to access your buckets. For more information about the app role permissions policy, see Permissions that SimSpace Weaver creates for you.
- **Your Amazon S3 bucket could be in a different AWS Region than your simulation** – Your Amazon S3 buckets for your simulation artifacts must be in the same AWS Region as your simulation. Check your Amazon S3 console to see what AWS Region your bucket is in. If your Amazon S3 bucket is in a different AWS Region, select a bucket that is in the same AWS Region as your simulation.

Problems related to time when using Docker

If you are using Docker and you receive time-related errors while running scripts from the SimSpace Weaver app SDK, the cause could be that your Docker virtual machine clock is incorrect. This can happen if your computer was running Docker and then resumes from sleep or hibernation.

Solutions to try

- Restart Docker.

- Disable and then re-enable time synchronization in **Windows PowerShell**:

```
Get-VMIntegrationService -VMName DockerDesktopVM -Name "Time Synchronization" |
 Disable-VMIntegrationService
Get-VMIntegrationService -VMName DockerDesktopVM -Name "Time Synchronization" |
 Enable-VMIntegrationService
```

PathfindingSample console client fails to connect

You might get the following error from the console client when you connect to the `PathfindingSample` simulation described in the tutorials in Getting started with SimSpace Weaver. This error occurs because the client can't open a network connection to the `ViewApp` at the combined IP address and port number that you provided.

```
Fatal error in function nng_dial. Error code: 268435577. Error message: no link
```

For a simulation in the AWS Cloud

- **Is your network connection working correctly?** Verify that you can connect to other IP addresses or web sites that should work. Make sure that your web browser isn't loading a web site from its cache.
- **Is your simulation running?** You can use the **ListSimulations** API to get the status of your simulation. For more information, see Get the IP address and port number of a custom app. You can also use the SimSpace Weaver console to check the status of your simulations.
- **Are your apps running?** You can use the **DescribeApp** API to get the status of your apps. For more information, see Get the IP address and port number of a custom app. You can also use the SimSpace Weaver console to check the status of your simulations.
- **Are your apps running?** You can use the **DescribeApp** API to get the status of your apps. For more information, see Get the IP address and port number of a custom app. You can also use the SimSpace Weaver console to check the status of your simulations.
- **Did you use the correct IP address and port number?** When you connect over the internet, you must use the IP address and `Actual` port number of the `ViewApp`. You can find the IP `Address` and `Actual` port number in the `EndpointInfo` block of the **DescribeApp** API output. You can also use the SimSpace Weaver console to find the IP address (**URI**) and port number (**Ingress port**) for your `ViewApp` in the `MyViewDomain` detail page.

- **Is your network connection going through a firewall?** Your firewall might block your connection to the IP address or port number (or both). Check your firewall settings or check with your firewall administrator.

For a local simulation

- **Can you connect to your loopback address (127.0.0.1)?** If you have the `ping` command line tool in Windows, you can open a command prompt window and try to ping 127.0.0.1. Press **Ctrl-C** to end the ping.

  ```
  ping 127.0.0.1
  ```

  Example ping output

  ```
  C:\>ping 127.0.0.1

  Pinging 127.0.0.1 with 32 bytes of data:
  Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
  Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
  Reply from 127.0.0.1: bytes=32 time=1ms TTL=128

  Ping statistics for 127.0.0.1:
      Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
  Approximate round trip times in milli-seconds:
      Minimum = 0ms, Maximum = 1ms, Average = 0ms
  Control-C
  ^C
  C:\>
  ```

  If the ping says it lost packets, you might have other software (such as a local firewall, security settings, or anti-malware programs) that is blocking your connection.

- **Are your apps running?** Your local simulation runs as separate windows for each app. Make sure the windows for your spatial apps and `ViewApp` are open. For more information, see Local development in SimSpace Weaver.
- **Did you use the correct IP address and port number?** You must use `tcp://127.0.0.1:7000` when you connect to a local simulation. For more information, see Local development in SimSpace Weaver.

- **Do you have local security software that could block your connection?** Check your security settings, local firewall, or anti-malware programs to see if they are blocking your connection to `127.0.0.1` on TCP port *7000*.

The AWS CLI doesn't recognize `simspaceweaver`

If the AWS CLI gives you errors that suggest that it doesn't know about SimSpace Weaver, run the following command.

```
aws simspaceweaver help
```

If you get an error that starts with the following lines and lists all available choices then your AWS CLI might be an older version.

```
usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:

  aws help
  aws <command> help
  aws <command> <subcommand> help

aws: error: argument command: Invalid choice, valid choices are:
```

Run the following command to check the version of your AWS CLI.

```
aws --version
```

If the version number is earlier than 2.9.19 then you must update your AWS CLI. Note that the current version of the AWS CLI is later than 2.9.19.

To update your AWS CLI, see Install or update the latest version of the AWS CLI in the *AWS Command Line Interface User Guide for Version 2*.

SimSpace Weaver simulation schema reference

SimSpace Weaver uses a YAML file to configure the properties of a simulation. This file is called the *simulation schema* (or just *schema*). The sample simulation included in the SimSpace Weaver app SDK includes a schema that you can copy and edit for your own simulation.

Topics

Example of a complete schema

The following example shows the YAML-formatted text file that describes a SimSpace Weaver simulation. This example includes dummy values for the properties. The format of the file varies based on the value of `sdk_version` specified in it. See Schema format for a complete description of the properties and their valid values.

```
sdk_version: "1.17"
simulation_properties:
  log_destination_resource_name: "MySimulationLogs"
  log_destination_service: "logs"
  default_entity_index_key_type: "Vector3<f32>"
  default_image: "111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-
repository:latest"
workers:
  MyComputeWorkers:
    type: "sim.c5.24xlarge"
    desired: 3
clock:
  tick_rate: "30"
partitioning_strategies:
  MyGridPartitioning:
    topology: "Grid"
    aabb_bounds:
      x: [-1000, 1000]
      y: [-1000, 1000]
    grid_placement_groups:
      x: 3
      y: 3
```

```
domains:
  MyCustomDomain:
    launch_apps_via_start_app_call: {}
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MyViewApp.zip"
      launch_command: ["MyViewApp"]
      required_resource_units:
        compute: 1
      endpoint_config:
        ingress_ports: [9000, 9001]
  MyServiceDomain:
    launch_apps_per_worker:
      count: 1
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/
MyConnectionServiceApp.zip"
      launch_command: ["MyConnectionServiceApp"]
      required_resource_units:
        compute: 1
      endpoint_config:
        ingress_ports:
          - 9000
          - 9001
  MySpatialDomain:
    launch_apps_by_partitioning_strategy:
      partitioning_strategy: "MyGridPartitioning"
      grid_partition:
        x: 6
        y: 6
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MySpatialApp.zip"
      launch_command: ["MySpatialApp"]
      required_resource_units:
        compute: 1
  MySpatialDomainWithCustomContainer:
    launch_apps_by_partitioning_strategy:
      partitioning_strategy: "MyGridPartitioning"
      grid_partition:
        x: 6
        y: 6
    app_config:
      package: "s3://weaver-myproject-111122223333-us-west-2/MySpatialApp2.zip"
      launch_command: ["MySpatialApp2"]
      required_resource_units:
        compute: 1
      image: "111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-repository:latest"
placement_constraints:
  - placed_together: ["MySpatialDomain", "MySpatialDomainWithCustomContainer"]
    on_workers: ["MyComputeWorkers"]
```

Schema format

The following example shows the overall structure of a schema. The order of properties at each level of the schema doesn't matter, as long as the parent-child relationships are the same. The order matters for elements in an array.

```
sdk_version: "sdk-version-number"
simulation_properties:
  simulation-properties
workers:
  worker-group-configurations
clock:
  tick_rate: tick-rate
partitioning_strategies:
  partitioning-strategy-configurations
domains:
  domain-configurations
placement_constraints:
  placement-constraints-configuration
```

Sections

- SDK version
- Simulation properties
- Workers
- Clock
- Partitioning strategies
- Domains
- Placement constraints

SDK version

The `sdk_version` section (required) identifies the version of the SimSpace Weaver app SDK that supports this schema. Valid values: `1.17`, `1.16`, `1.15`, `1.14`, `1.13`, `1.12`

Important

The value of `sdk_version` only includes the major version number and first minor version number. For example, the value `1.12` specifies all versions `1.12.x`, such as `1.12.0`, `1.12.1`, and `1.12.2`.

```
sdk_version: "1.17"
```

Simulation properties

The `simulation_properties` section (required) specifies various properties of your simulation. Use this section to configure logging and specify a default container image. This section is required even if you don't configure logging or choose to specify a default container image.

```
simulation_properties:
  log_destination_resource_name: "log-destination-resource-name"
  log_destination_service: "log-destination-service"
  default_entity_index_key_type: "Vector3<f32>"
  default_image: "ecr-repository-uri"
```

Properties

`log_destination_resource_name`

Specifies the resource that SimSpace Weaver will write logs to.

Required: No. If this property isn't included, SimSpace Weaver won't write logs for the simulation.

Type: String

Valid values:

- The name of an CloudWatch Logs log group (for example, `MySimulationLogs`)

- The Amazon Resource Name (ARN) of a CloudWatch Logs log group (for example, `arn:aws:logs:us-west-2:111122223333:log-group/MySimulationLogs`)

> **Note**
>
> SimSpace Weaver only supports a log destination in the same account and AWS Region as the simulation.

`log_destination_service`

Indicates the type of logging destination resource when you specify a `logging_destination resource_name` that isn't an ARN.

Required: You must specify this property if the `log_destination_resource_name` is specified and isn't an ARN. You can't specify this property if the `log_destination_resource_name` isn't specified or is an ARN.

Type: String

Valid values:

- `logs`: The log destination resource is a log group.

`default_entity_index_key_type`

Specifies the data type for the index key field of simulation entities.

Required: Yes

Type: String

Valid values: `Vector3<f32>`

`default_image`

Specifies the default container image for your simulation (not supported for version `1.13` and `1.12`). If this property is specified, domains that don't specify an `image` use the `default_image`.

Required: No

Type: String

Valid values:

- The URI of a repository in Amazon Elastic Container Registry (Amazon ECR) (for example, `111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-repository:latest`)

Workers

The `workers` section (required) specifies configurations for *worker groups* (groups of workers). SimSpace Weaver uses this information together with `placement_constraints` to configure the underlying infrastructure of your simulation. Only 1 worker group is currently supported.

To specify the properties for a worker group, replace *`worker-group-name`* with a name of your choice. The name must be 3-64 characters long and can contain the characters **A -Z** , **a -z** , **0 -9** , and **_ -** (hyphen). Specify the properties of the worker group after the name.

```
workers:
  worker-group-name:
    type: "sim.c5.24xlarge"
    desired: number-of-workers
```

Properties

`type`

Specifies worker type.

Required: Yes

Type: String

Valid values: `sim.c5.24xlarge`

`desired`

Specifies the desired number of workers for this worker group.

Required: Yes

Type: Integer

Valid values: 1-3. Your service quota (limit) for the number of workers for your simulations determines the maximum value of this property. For example, if your service quota is 2 then the maximum value for this property is 2. You can request an increase to your service quota. For more information, see SimSpace Weaver endpoints and quotas.

Clock

The `clock` section (required) specifies the properties of the simulation clock.

```
clock:
  tick_rate: tick-rate
```

Properties

`tick_rate`

Specifies the number of ticks per second that the clock publishes to apps.

Required: Yes

Type:

- *Version `1.14` and `1.15`:* String
- *Version `1.13` and `1.12`:* Integer

Valid values:

- *Version `1.14` and `1.15`:* `"10"` | `"15"` | `"30"` | `"unlimited"`
  - `"unlimited"`: the clock sends the next tick as soon as all apps finish their commit operations for the current tick.
- *Version `1.13` and `1.12`:* `10` | `15` | `30`

Partitioning strategies

The `partitioning_strategies` section (required) specifies the organization of partitions for a spatial domain.

Note

SimSpace Weaver only supports 1 partitioning strategy.

To specify the properties for a partitioning strategy, replace *partitioning-strategy-name* with a name of your choice. The name must be 3-64 characters long and can contain the characters **A**-**Z**, **a**-**z**, **0**-**9**, and **_** **-** (hyphen). Specify the properties of the partitioning strategy after the name.

```
partitioning_strategies:
  partitioning-strategy-name:
    topology: "Grid"
    aabb_bounds:
      x: [aabb-min-x, aabb-max-x]
      y: [aabb-min-y, aabb-max-y]
    grid_placement_groups:
      x: number-of-placement-groups-along-x-axis
      y: number-of-placement-groups-along-y-axis
```

Properties

`topology`

Specifies the topology (partition arrangement scheme) for this partitioning strategy.

Required: Yes

Type: String

Valid values: `"Grid"`

`aabb_bounds`

Specifies the bounds of the main axis-aligned bounding box (AABB) for your simulation. You specify the bounds as 2-element ordered arrays that describe the minimum and maximum values (in that order) for each axis (x and y).

Required: Conditional. This property is required (and can only be specified) if the topology is set to `"Grid"`.

Type: `Float` array (for each axis)

Valid values: `-3.4028235e38-3.4028235e38`

`grid_placement_groups`

Specifies the number of placement groups along each axis (x and y) in a grid topology. A placement group is a collection of partitions (in the same domain) that are spatially adjacent.

Required: Conditional. This property is required (and can only be specified) if the topology is set to `"Grid"`. If you don't specify a placement groups configuration, SimSpace Weaver will calculate one for you. Any domain that uses a partitioning strategy without a placement groups configuration must specify a `grid_partition` (see Spatial domain partitioning strategy).

Type: Integer (for each axis)

Valid values: `1-20`. We recommend that x * y is equal to the desired number of workers. Otherwise, SimSpace Weaver will attempt to balance your placement groups across the available workers.

Domains

The `domains` section (required) specifies the properties for each of your domains. All simulations must have at least one section for a spatial domain. You can create multiple sections for additional domains. Each type of domain has its own configuration format.

Important

Versions `1.13` and `1.12` don't support multiple spatial domains.

Important

SimSpace Weaver supports up to 5 domains for each simulation. This includes all spatial, custom, and service domains.

```
domains:
  domain-name:
    domain-configuration
  domain-name:
    domain-configuration
  ...
```

Domain configuration

- Spatial domain configuration
- Custom domain configuration

- Service domain configuration

Spatial domain configuration

To specify the properties for a spatial domain, replace *spatial-domain-name* with a name of your choice. The name must be 3-64 characters long and can contain the characters **A** -**Z** , **a** -**z** , **0** -**9** , and **_** **-** (hyphen). Specify the properties of the spatial domain after the name.

```
spatial-domain-name:
  launch_apps_by_partitioning_strategy:
    partitioning_strategy: "partitioning-strategy-name"
    grid_partition:
      x: number-of-partitions-along-x-axis
      y: number-of-partitions-along-y-axis
  app_config:
    package: "app-package-s3-uri"
    launch_command: ["app-launch-command", "parameter1", ...]
    required_resource_units:
      compute: app-resource-units
  image: "ecr-repository-uri"
```

Spatial domain partitioning strategy

The `launch_apps_by_partitioning_strategy` section (required) specifies the partitioning strategy and dimensions (in number of partitions) of the simulation space.

```
  launch_apps_by_partitioning_strategy:
    partitioning_strategy: "partitioning-strategy-name"
    grid_partition:
      x: number-of-partitions-along-x-axis
      y: number-of-partitions-along-y-axis
```

Properties

`partitioning_strategy`

Specifies the partitioning strategy for this spatial domain.

Required: Yes

Type: String

Valid values: The value of this property must match the name of a partitioning strategy defined in the `partitioning_strategies` section. For more information, see Partitioning strategies.

`grid_partition`

Specifies the number of partitions along each axis (x and y) in a grid topology. These dimensions describe the total simulation space for this domain.

Required: Conditional. This property can only be specified if the topology is set to `"Grid"`. This property depends on the `grid_placement_groups` property of the specified partitioning strategy for this domain:

- This property is required if this domain's partitioning strategy doesn't specify a `grid_placement_groups` configuration.
- If there is a `grid_placement_groups` configuration but you don't specify `grid_partition`, then SimSpace Weaver will use the same dimensions as the `grid_placment_groups` configuration.
- If you specify both `grid_placement_groups` and `grid_partition`, the dimensions of `grid_partition` must be multiples of the dimensions of `grid_placement_groups` (for example, if your `grid_placement_groups` dimensions are 2x2, then some valid dimensions for `grid_partition` are 2x2, 4x4, 6x6, 8x8, 10x10).

Type: Integer (for each axis)

Valid values: `1-20`

Spatial app configuration

The `app_config` section (required) specifies the package, launch configuration, and resource requirements for apps in this domain.

```
app_config:
  package: "app-package-s3-uri"
  launch_command: ["app-launch-command", "parameter1", ...]
  required_resource_units:
    compute: app-resource-units
```

Properties

`package`

Specifies the package (zip file) that contains the app executable/binary. The package must be stored in an Amazon S3 bucket. Only zip file format is supported.

Required: Yes

Type: String

Valid values: The Amazon S3 URI of the package in an Amazon S3 bucket. For example, `s3://example-bucket/MySpatialApp.zip`.

`launch_command`

Specifies the executable/binary file name and command-line parameters to launch the app. Each command-line string token is an element in the array.

Required: Yes

Type: String array

`required_resource_units`

Specifies the number of resource units that SimSpace Weaver should allocate to each instance of this app. A *resource unit* is a fixed amount of virtual central processing units (vCPUs) and random-access memory (RAM) on a worker. For more information about resource units, see Endpoints and service quotas. The `compute` property specifies a resource unit allocation for the `compute` family of workers, and is currently the only valid type of allocation.

Required: Yes

Type: Integer

Valid values: 1-4

Custom container image

The `image` property (optional) specifies the location of a container image that SimSpace Weaver uses to run apps in this domain (not supported in versions `1.13` and `1.12`). Provide the URI to a repository in Amazon Elastic Container Registry (Amazon ECR) that contains the image. If this property isn't specified but the `default_image` is specified in the top level `simulation_properties` section, apps in this domain use the `default_image`. For more information, see Custom containers.

```
    image: "ecr-repository-uri"
```

Properties

`image`

Specifies the location of a container image to run apps in this domain.

Required: No

Type: String

Valid values:

- The URI of a repository in Amazon Elastic Container Registry (Amazon ECR) (for example, `111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-repository:latest`)

Custom domain configuration

To specify the properties for a custom domain, replace *custom-domain-name* with a name of your choice. The name must be 3-64 characters long and can contain the characters **A -Z** , **a -z** , **0 -9** , and **_ -** (hyphen). Specify the properties of the custom domain after the name. Repeat this process for each custom domain.

```
  custom-domain-name:
    launch_apps_via_start_app_call: {}
    app_config:
      package: "app-package-s3-uri"
      launch_command: ["app-launch-command", "parameter1", ...]
      required_resource_units:
        compute: app-resource-units
      endpoint_config:
        ingress_ports: [port1, port2, ...]
    image: "ecr-repository-uri"
```

Properties

`launch_apps_via_start_app_call`

This property is required to launch your custom apps using the **StartApp** API.

Required: Yes

Type: N/A

Valid values: `{}`

Custom app configuration

The `app_config section` (required) specifies the package, launch configuration, resource requirements, and network ports for apps in this custom domain.

```
app_config:
  package: "app-package-s3-uri"
  launch_command: ["app-launch-command", "parameter1", ...]
  required_resource_units:
    compute: app-resource-units
  endpoint_config:
    ingress_ports: [port1, port2, ...]
```

Properties

`package`

Specifies the package (zip file) that contains the app executable/binary. The package must be stored in an Amazon S3 bucket. Only zip file format is supported.

Required: Yes

Type: String

Valid values: The Amazon S3 URI of the package in an Amazon S3 bucket. For example, `s3://example-bucket/MyCustomApp.zip`.

`launch_command`

Specifies the executable/binary file name and command-line parameters to launch the app. Each command-line string token is an element in the array.

Required: Yes

Type: String array

`required_resource_units`

Specifies the number of resource units that SimSpace Weaver should allocate to each instance of this app. A *resource unit* is a fixed amount of virtual central processing units (vCPUs) and random-access memory (RAM) on a worker. For more information about resource units, see Endpoints and service quotas. The `compute` property specifies a resource unit allocation for the `compute` family of workers, and is currently the only valid type of allocation.

Required: Yes

Type: Integer

Valid values: `1-4`

`endpoint_config`

Specifies the network endpoints for apps in this domain. The value of `ingress_ports` specifies the ports that your custom apps bind to for incoming client connections. SimSpace Weaver maps dynamically allocated ports to your specified ingress ports. Ingress ports are both TCP and UDP. Use the **DescribeApp** API to find the actual port number to connect your clients.

Required: No. If you don't specify endpoint configuration then your custom apps in this domain won't have network endpoints.

Type: Integer array

Valid values: `1024-49152`. Values must be unique.

Custom container image

The `image` property (optional) specifies the location of a container image that SimSpace Weaver uses to run apps in this domain (not supported in versions `1.13` and `1.12`). Provide the URI to a repository in Amazon Elastic Container Registry (Amazon ECR) that contains the image. If this property isn't specified but the `default_image` is specified in the top level `simulation_properties` section, apps in this domain use the `default_image`. For more information, see Custom containers.

```
image: "ecr-repository-uri"
```

Properties

`image`

Specifies the location of a container image to run apps in this domain.

Required: No

Type: String

Valid values:

- The URI of a repository in Amazon Elastic Container Registry (Amazon ECR) (for example, `111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-repository:latest`)

Service domain configuration

To specify the properties for a service domain, replace *`service-domain-name`* with a name of your choice. The name must be 3-64 characters long and can contain the characters **A -Z** , **a -z** , **0 -9** , and **_ -** (hyphen). Specify the properties of the service domain after the name. Repeat this process for each service domain.

```
service-domain-name:
  launch_apps_per_worker:
    count: number-of-apps-to-launch
  app_config:
    package: "app-package-s3-uri"
    launch_command: ["app-launch-command", "parameter1", ...]
    required_resource_units:
      compute: app-resource-units
    endpoint_config:
      ingress_ports: [port1, port2, ...]
  image: "ecr-repository-uri"
```

Launch apps per worker

The `launch_apps_per_worker` section (required) indicates that this is a service domain configuration, and specifies the number of service apps to launch per worker.

```
  launch_apps_per_worker:
    count: number-of-apps-to-launch
```

Properties

`count`

This property specifies the number of service apps to launch per worker.

Required: Yes

Type: Integer

Valid values: {} | 1 | 2. A value of {} specifies the default value of 1.

Service app configuration

The `app_config section` (required) specifies the package, launch configuration, resource requirements, and network ports for apps in this service domain.

```
app_config:
  package: "app-package-s3-uri"
  launch_command: ["app-launch-command", "parameter1", ...]
  required_resource_units:
    compute: app-resource-units
  endpoint_config:
    ingress_ports: [port1, port2, ...]
```

Properties

`package`

Specifies the package (zip file) that contains the app executable/binary. The package must be stored in an Amazon S3 bucket. Only zip file format is supported.

Required: Yes

Type: String

Valid values: The Amazon S3 URI of the package in an Amazon S3 bucket. For example, `s3://example-bucket/MyServiceApp.zip`.

`launch_command`

Specifies the executable/binary file name and command-line parameters to launch the app. Each command-line string token is an element in the array.

Required: Yes

Type: String array

`required_resource_units`

Specifies the number of resource units that SimSpace Weaver should allocate to each instance of this app. A *resource unit* is a fixed amount of virtual central processing units (vCPUs) and random-access memory (RAM) on a worker. For more information about resource units, see Endpoints and service quotas. The `compute` property specifies a resource unit allocation for the `compute` family of workers, and is currently the only valid type of allocation.

Required: Yes

Type: Integer

Valid values: `1-4`

`endpoint_config`

Specifies the network endpoints for apps in this domain. The value of `ingress_ports` specifies the ports that your service apps bind to for incoming client connections. SimSpace Weaver maps dynamically allocated ports to your specified ingress ports. Ingress ports are both TCP and UDP. Use the **DescribeApp** API to find the actual port number to connect your clients.

Required: No. If you don't specify endpoint configuration then your service apps in this domain won't have network endpoints.

Type: Integer array

Valid values: `1024-49152`. Values must be unique.

Custom container image

The `image` property (optional) specifies the location of a container image that SimSpace Weaver uses to run apps in this domain (not supported in versions `1.13` and `1.12`). Provide the URI to a repository in Amazon Elastic Container Registry (Amazon ECR) that contains the image. If this property isn't specified but the `default_image` is specified in the top level `simulation_properties` section, apps in this domain use the `default_image`. For more information, see Custom containers.

```
    image: "ecr-repository-uri"
```

Properties

`image`

Specifies the location of a container image to run apps in this domain.

Required: No

Type: String

Valid values:

- The URI of a repository in Amazon Elastic Container Registry (Amazon ECR) (for example, `111122223333.dkr.ecr.us-west-2.amazonaws.com/my-ecr-repository:latest`)

Placement constraints

The `placement_constraints` section (optional) specifies which spatial domains SimSpace Weaver should place together on the same workers. For more information, see Configuring spatial domains.

Important

Versions `1.13` and `1.12` don't support `placement_constraints`.

```
placement_constraints:
  - placed_together: ["spatial-domain-name", "spatial-domain-name", ...]
    on_workers: ["worker-group-name"]
```

Properties

`placed_together`

Specifies the spatial domains that SimSpace Weaver should place together.

Required: Yes

Type: String array

Valid values: Names of spatial domains specified in the schema `on_workers`

Specifies the worker group that that SimSpace Weaver should place the domains on.

Required: Yes

Type: 1-element string array

Valid values: Name of a worker group specified in the schema

SimSpace Weaver API references

SimSpace Weaver has 2 different sets of application programming interfaces (APIs):

- **Service APIs –** These APIs control the service and service resources, such as your simulations, clocks, and apps. They are part of the main AWS software development kit (SDK) and you can use the AWS Command Line Interface (CLI) to call them. For more information about the servcie APIs, see the SimSpace Weaver API reference.
- **App SDK APIs –** These APIs control the data within your simulation. You use them in your app code to do things like read and write entity field data, work with subscriptions, and monitor events in the simulation. For more information, see the SimSpace Weaver app SDK documentation in your app SDK folder: `*sdk-folder*\SimSpaceWeaverAppSdk\documentation`

Note

`sdk-folder` is the folder where you unzipped the SimSpaceWeaverAppSdkDistributable package.

AWS SimSpace Weaver versions

We continuously improve AWS SimSpace Weaver. You must download the latest SimSpace Weaver app SDK when we release a new version if you want to take advantage of new features and feature updates. To run an existing simulation with a newer version, you might have to update its schema and code, then start a new instance of the simulation. You don't have to upgrade, and can continue to run existing simulations with previous versions. You can check this page to see what's different between the versions. All versions are currently supported.

Important

The latest version of the AWS SimSpace Weaver User Guide only covers the latest version of the service. You can find the documentation for previous versions in the AWS SimSpace Weaver Guide Catalog, available from the main documentation landing page.

Latest version

The latest version is: 1.17.0

How to find your current version

If you created a simulation with the SimSpace Weaver app SDK, the `create-project` script downloaded a version of the SDK libraries into a subdirectory in your *`sdk-folder`*. The subdirectory that contains the SDK libraries has a name that includes the SDK version number: `SimSpaceWeaverAppSdk-`*`sdk-version`*. For example, the libraries for version 1.16.0 are in `SimSpaceWeaverAppSdk-1.16.0`.

You can also find the version of the SimSpace Weaver app SDK distributable package in the text file `app_sdk_distributable_version.txt` in your *`sdk-folder`*.

Download the latest version

Use one of the following links to download the latest version.

- Complete app SDK distributable package
- Only the app SDK libraries

You can also download the complete SimSpace Weaver app SDK distributable package from the SimSpace Weaver console in the AWS Management Console. Choose **Download app SDK** from the navigation pane.

Warning

Don't use the AWS CLI to download anything that appears to be the SimSpace Weaver app SDK distributable package. Only use the download links on this page or the download link in the console. Any other download method or location isn't supported and might contain obsolete, incorrect, or malicious code.

Troubleshooting app SDK downloads

We use Amazon CloudFront (CloudFront) to distribute the app SDK .zip files. You might experience some of the following situations.

- **The downloaded package isn't the latest version**
  - If the .zip file that you downloaded doesn't contain the latest version, it's possible that the cache at your CloudFront edge location hasn't updated yet. Download again after 24 hours.
- **You get a HTTP 4xx or 5xx error using a download link**
  - Try again after 24 hours. If you get the same error, use the **Feedback** link at the bottom of the SimSpace Weaver console to report the problem. Select **Report an issue** as the **Type** of feedback.
- **Your browser reports that it can't load the page**
  - You might have a local network or browser configuration problem. Verify that you can load other pages. Clear your browser cache and try again. Make sure that you don't have firewall rules that might block the download URL.
- **You get an error when you try to save the file**
  - Check your local file system permissions to make sure that you have the correct permissions to save the file.
- **Your browser displays AccessDenied**
  - If you manually entered the URL into your browser, check that it is correct. If you used a download link, make sure something didn't interfere with the URL in your browser; use the link again.

Install the latest version

To install the latest version

1. Download the latest version.
2. Unzip the SimSpaceWeaverAppSdkDistributable.zip to a folder.
3. Run `python setup.py` from the unzipped latest version SimSpace Weaver app SDK folder.
4. Use the unzipped latest version SimSpace Weaver app SDK folder instead of the previous version.

Service versions

Version	Notes	Release date	Documentation	App SDK download
1.17.0	**Major changes to the SimSpace Weaver app SDK distributable package** • We replaced the Windows batch and Linux Bash scripts with Python-based scripts. Python 3.9 is therefore now required to use the scripts and samples, even if you don't use (or intend to use) the Python SDK.	April 17, 2024	This guide	• Complete package • Libraries only See Troublesh ooting.

Version	Notes	Release date	Documentation	App SDK download
	• This release increases support for Amazon Linux 2. • We fixed several bugs in SimSpace Weaver Local. For more informati on, see the release notes. **Bug fix** • We fixed a bug causing entities to become unowned if they didn't complete their transfer between remote workers.			

Version	Notes	Release date	Documentation	App SDK download
1.16.0	**New feature:** • You can now use messaging APIs in the SimSpace Weaver app SDK to send and receive messages between your apps. This feature is available for C++, Python, and the Unity and Unreal Engine 5 integrations.	February 12, 2024	See the AWS SimSpace Weaver Guide Catalog.	• Complete package • Libraries only See Troubleshooting.

Version	Notes	Release date	Documentation	App SDK download
1.15.3	**SimSpace Weaver Local update:** • We changed SimSpace Weaver Local to more closely align it with development for the AWS Cloud. These changes impact C++, Python, Unity, and Unreal Engine projects and workflows for SimSpace Weaver Local.	December 4, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download
1.15.2	**App SDK distribut able package update:** • We updated the `Dockerfile` to use the specific required version of cmake. Docker container builds might fail without this change.	November 2, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download

Version	Notes	Release date	Documentation	App SDK download
1.15.1	**Feature update:** • **Python SDK:** This release fixes an issue that caused Python-based simulations to fail in the AWS Cloud.	September 22, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download
1.15.0	**New feature:** • **Python SDK:** You can now develop your simulations in Python. The SimSpace Weaver app SDK distributable package includes a template for a sample Python project and its Python view client.	August 31, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download

Version	Notes	Release date	Documentation	App SDK download
1.14.0	**New features:** • **Custom containers:** Create your own Amazon Linux 2 (AL2) based container image, store it in Amazon Elastic Container Registry (Amazon ECR), and use it to run your SimSpace Weaver apps in the AWS Cloud. • **Multiple spatial domains:** Create more than 1 spatial domain in a simulatio n. Separate simulation logic instead of combining it all into a single spatial app. Allocate different resources to spatial domains based on their requirements.	July 26, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download

Version	Notes	Release date	Documentation	App SDK download
	• **Unlimited tick rate:** Enable your simulation to run as fast as your code can execute. Set your simulation's clock so that it sends the next tick as soon as all apps finish their commit operations for the current tick. **SimSpace Weaver app SDK:** • The value of `tick_rate` is now a string. The value must include double quotes ("). The tick rate for earlier versions is still an integer.			

Version	Notes	Release date	Documentation	App SDK download
1.13.1	**SimSpace Weaver app SDK:** • Feature update: Project creation now works correctly with the `PathfindingSampleUnreal` template.	June 7, 2023	See the [AWS SimSpace Weaver Guide Catalog]().	Not available for download

Version	Notes	Release date	Documentation	App SDK download
1.13.0	**SimSpace Weaver service APIs:** • • New `CreateSnapshot` action • Changes to the `StartSimulation` action: • Added a `SnapshotS3Location` parameter to start from a snapshot. • The `SchemaS3Location` parameter is now optional. • Changes to `DescribeSimulation` output: • `SchemaError` is deprecated.	April 29, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download

Version	Notes	Release date	Documentation	App SDK download
	<ul><li><ul><li>Added a `StartError` field.</li><li>Added a `SnapshotS3Location` field.</li><li>Added a `SNAPSHOT_IN_PROGRESS` simulation status.</li></ul></li><li>New [`S3Destination`] data type</li></ul> **SimSpace Weaver console:** <ul><li>New functionality to create snapshots.</li><li>New functionality to start a simulation from a snapshot.</li></ul>			

Version	Notes	Release date	Documentation	App SDK download

	SimSpace Weaver app SDK:
	
	• New scripts to support snapshots
	  ◦ `create-snapshot-` *`proj`* *`name`* `.bat`
	  ◦ `start-from-snapshot-` *`project-name`* `.bat`
	  ◦ `quick-start-from-snapshot-` *`pro`* *`name`* `-cli.bat`
	  ◦ `list-snapshots-` *`proje`* *`name`* `.bat`
	• Projects now use a single Amazon S3 bucket per project: weaver-*`lowercas`* *`-project-name`* `-`*`account-number`* `-`*`region`*			

Version	Notes	Release date	Documentation	App SDK download

1.12.3	**SimSpace Weaver app SDK:**
	
	• The following scripts now support the `--maximum-duration` parameter:
	  ◦ `quick-start-`*`project-name`*`-cli.bat`
	  ◦ `quick-start-`*`project-name`*`-cli.sh`
	  ◦ `start-simulation-`*`pro`* *`name`*`.bat`
	  ◦ `start-simulation-`*`pro`* *`name`*`.sh`
	  ◦ `run-`*`project-name`*`.bat`
	  ◦ `run-`*`project-name`*`.sh`
		March 27, 2023	See the [AWS SimSpace Weaver Guide Catalog]().	Not available for download

Version	Notes	Release date	Documentation	App SDK download
1.12.2	**SimSpace Weaver app SDK:** • Bug fix: `docker-create-image.bat` now runs correctly.	March 1, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download

Version	Notes	Release date	Documentation	App SDK download
1.12.1	**SimSpace Weaver app SDK:** • The scripts now accept an AWS CLI profile to use for AWS authentication. • The scripts now support AWS IAM Identity Center for AWS authentication. **SimSpace Weaver Local:** • Bug fix: `Api::BeginUpdateWillBlock` now correctly returns `true` if all of the spatial apps have not joined the simulation.	February 28, 2023	See the AWS SimSpace Weaver Guide Catalog.	Not available for download
1.12.0	Release for general availability (GA)	November 29, 2022	See the AWS SimSpace Weaver Guide Catalog.	Not available for download

AWS SimSpace Weaver version 1.17.0

This release is an overhaul of the SimSpace Weaver app SDK distributable package. We replaced outdated Windows batch and Linux Bash scripts with Python-based scripts.

Important

Python 3.9 is now a requirement to use the scripts and samples, not just for the Python SDK.

Contents

Major changes for 1.17.0

- Simplified project creation
  - After running `setup.py`, you can create your own project by simply copy-pasting a sample.
- 1-click samples
  - The distribution zip file now contains ready-to-use samples that work after setting up the distribution.
- Each SDK now exists in its own directory: `cpp`, `python`, `unreal`, and `unity`. You might have to update your paths depending on which SDK you use.
- Improvements to helper scripts.
  - Scripts now contain multiple AWS CLI options to maximize their flexibility.
  - Integrated console client launch and connection as part of quick-start.
  - Improved console output.
  - Unreal and Unity sample building now works with `quick-start`, no more manual steps required.
  - SimSpace Weaver Local now works by just calling `quick-start`, no more manual building and launching.

  - SimSpace Weaver Local `quick-start` has integrated support for logging app output.
  - SimSpace Weaver Local can now be launched in a non-GUI environment, such as in an ssh session.
  - The "custom container" feature is now integrated into the `quick-start` script.
  - Increased Amazon Linux 2 (AL2) support: script workflows for Windows and AL2 are now comparable. Previously, AL2 projects required more manual steps and SimSpace Weaver Local wasn't supported for AL2.
- Unreal Engine and Unity plugins are now included as part of the SimSpace Weaver app SDK distributable package.
- Bug fixes for SimSpace Weaver Local
  - Fixed a bug where entities could be assigned the same entity ID.
  - Fixed a bug where two partitions could be assigned the same partition ID.
  - Fixed a bug related to apps attempt to write to entities they did not own.
  - Resolved a memory leak issue.

Update a project to 1.17.0

1. Setup the 1.17.0 distribution: Go through the setup procedures again because we changed them for 1.17.0. For more information, see Setting up for SimSpace Weaver.
2. Each Weaver App SDK now exists in it’s own directory. Update your build paths to reflect this.

   a. C++ directory: `SimSpaceWeaverAppSdk/cpp`

      - The SimSpace Weaver C++ app SDK now uses a `FindSimSpaceWeaverAppSdk.cmake` file. This file sets up a `weaver` target that gets linked to, and includes important bug fixes when building for Weaver in the AWS Cloud. You should use this instead of linking to the binaries directly.

   b. Python directory: `SimSpaceWeaverAppSdk/python`

   c. Unity plugin: `SimSpaceWeaverAppSdk/unity`

   d. Unreal Engine plugin: `SimSpaceWeaverAppSdk/unreal`
3. The previous `tools` scripts will not work with the new SimSpace Weaver distribution. To use the new `tools` scripts with your project:

   a. Delete your old `tools/windows`, `tools/linux`, and `tools/local` directories.

b. Copy the `tools` directory of a sample project that uses the same SimSpace Weaver app SDK as your project. Be sure you have run `setup.py` ***before*** you copy this directory.

> **Important**
>
> The tools scripts are only guaranteed to work with the sample projects. You might have to edit these scripts, especially the `build.py` script, to work with your project. Any edits will be unique to your project therefore we can't provide any guidance.

Frequently asked questions about version 1.17.0

Do I have to update to version 1.17.0?

This isn't a required update because there are no changes to the SimSpace Weaver API or SimSpace Weaver app SDK. You must update to 1.17.0 if you want to use the 1.17.0 SimSpace Weaver Local, which contains several bug fixes.

What is the minimum Python version required?

Python 3.9 is the minimum version.

What is the minimum CMake version required?

CMake version 3.13 is the minimum.

What is the minimum version of Unreal Engine required?

Unreal Engine 5.0 is the minimum.

What is the minimum version of Unity required?

Unity version 2022.3.19.F1 is the minimum.

AWS SimSpace Weaver version 1.15.1

This release is a **required** update for the Python SDK that was originally released in SimSpace Weaver version 1.15.0. It fixes a version mismatch issue that caused Python-based simulations to fail in the AWS Cloud. Use this version instead of 1.15.0.

Update an existing Python project to 1.15.1

If you have an existing Python project that you created with the version 1.15.0 Python SDK, you must perform the following steps to update it to 1.15.1 so that it can run in the AWS Cloud.

Instead of following this procedure, you can also create a new Python project with the 1.15.1 Python SDK and move your custom code to the new project.

To update a 1.15.0 Python project to 1.15.1

1. Go to your Python project's folder.
2. In `src/PythonBubblesSample/bin/run-python` change the following line:

   ```
   export PYTHONPATH=$PYTHONPATH:/roapp/lib
   ```

   To the following:

   ```
   export PYTHONPATH=$PYTHONPATH:$LD_LIBRARY_PATH:/roapp/lib
   ```

3. In `CMakeLists.txt` delete the following lines:

   -
     ```
     file(COPY "${SDK_PATH}/libweaver_app_sdk_python_v1_$ENV{PYTHON_VERSION}.so"
      DESTINATION "${ZIP_FILES_DIR}/lib/weaver_app_sdk_v1")
     ```

   -
     ```
     file(RENAME "${ZIP_FILES_DIR}/lib/weaver_app_sdk_v1/libweaver_app_sdk_python_v1_
     $ENV{PYTHON_VERSION}.so" "${ZIP_FILES_DIR}/lib/weaver_app_sdk_v1/
     libweaver_app_sdk_python_v1.so")
     ```

   -
     ```
     message("    * COPYING WEAVER PYTHON SDK TO BUILD DIR ${ZIP_FILES_DIR}....")
     ```

   -
     ```
     file(COPY ${SDK_DIR} DESTINATION ${ZIP_FILES_DIR}/lib/weaver_app_sdk_v1)
     ```

Troubleshooting for version 1.15.1

After updating a 1.15.0 Python simulation, it fails to start in the AWS Cloud

Symptoms: After approximately 5-10 minutes after you start the simulation, the simulation management log reports an `internal error` and the simulation status is `FAILED`.

This can happen if a library file from the 1.15.0 Python SDK is included in an app zip file. Make sure that you completed the steps to update your project, and make sure that `libweaver_app_sdk_python_v1.so` isn't in your zip files or referenced in any way.

Frequently asked questions about version 1.15.1

Does this release affect anything other than the Python SDK?

No.

Do I have to update to version 1.15.1?

You don't have to update to 1.15.1 if you don't intend to use Python for your spatial apps. If you updated to 1.15.0, your Python-based simulations won't run in the AWS Cloud. We recommend that you update to 1.15.1 if you use 1.15.0.

What is `$LD_LIBRARY_PATH`?

It's the location of the Python SDK when your simulation runs in the AWS Cloud. This is new for 1.15.1. We made this change to avoid Python version problems in the future. Linking to that directory is functionally the same as linking to `libweaver_app_sdk_python_v1.so` in 1.15.0.

Document history for AWS SimSpace Weaver

The following table describes important changes to the SimSpace Weaver documentation.

Date	Change	Documentation updates	API versions updated
April 17, 2024	Updated content	Updated throughout the user guide for the version 1.17.0 release. Major changes to the Setting up chapter and Getting started tutorials. See the release notes for more information.	N/A
February 12, 2024	Updated content	Updated the AWS SimSpace Weaver versions chapter for the version 1.16.0 release.	N/A
February 12, 2024	New content	Added the Messaging section as part of the version 1.16.0 release. This section describes the messaging APIs added to the SimSpace Weaver app SDK. You can use these APIs to send and receive messages between your apps.	N/A
February 12, 2024	Updated content	Updated the SimSpace Weaver simulation schema reference chapter for version 1.16.0.	N/A

Date	Change	Documentation updates	API versions updated
February 12, 2024	Updated content	Added service quotas for messaging to the SimSpace Weaver endpoints and quotas chapter.	N/A
February 12, 2024	New guides	Split the content for versions before 1.16.0 into a separate guide. Added the AWS SimSpace Weaver Guide Catalog (available from the main documentation landing page) to access guides for previous versions.	N/A
December 4, 2023	Updated content	Updated the AWS SimSpace Weaver versions chapter for the version 1.15.3 release.	N/A
December 4, 2023	Updated content	Updated the AWS SimSpace Weaver versions chapter to include installation instructions for the latest version.	N/A
December 4, 2023	Updated content	Updated the Service quotas for SimSpace Weaver Local.	N/A

Date	Change	Documentation updates	API versions updated
December 4, 2023	New and updated content	Restructured the Local development in SimSpace Weaver section and added a new page that describes the differences for SimSpace Weaver Local introduced in version 1.15.3.	N/A
November 7, 2023	Updated content	Updated the instructions to set up for Docker and WSL to use the direct download link/URL for the app SDK. For more information, see Set up your local environment for SimSpace Weaver.	N/A
November 2, 2023	Updated content	Updated the service versions page for the 1.15.2 release. For more information, see Service versions.	N/A

Date	Change	Documentation updates	API versions updated
October 23, 2023	Updated content	Updated the service versions page with new instructions to download the app SDK distributable package. Customers should now only use one of our approved direct download links and not use the AWS CLI to download the app SDK distributable package. For more information, see Download the latest version.	N/A
September 22, 2023	New content	Added a version notes page with update instructions for the 1.15.1 release. For more information, see AWS SimSpace Weaver version 1.15.1.	N/A
September 10, 2023	New content	Added a troubleshooting section for situations where the AWS CLI doesn't recognize SimSpace Weaver. For more information, see The AWS CLI doesn't recognize `simspaceweaver` .	N/A

Date	Change	Documentation updates	API versions updated
September 10, 2023	Updated content	Updated installation instructions for the AWS CLI in WSL. For more information, see Set up the SimSpace Weaver distribution package for Amazon Linux 2 (AL2) in Windows Subsystem for Linux (WSL).	N/A
September 7, 2023	API update	BucketName and ObjectKey are now required for the S3Location data type. BucketName is now required for the S3Destination data type.	AWS SDK: 2023-09-07
August 31, 2023	New content	Added a new section for release 1.15.0: Working with Python.	N/A
August 15, 2023	Updated content	Updated download instructions in AWS SimSpace Weaver versions to only list official SimSpace Weaver Amazon S3 buckets. Other download locations are not controlled by AWS and might contain malicious code.	N/A

Date	Change	Documentation updates	API versions updated
July 26, 2023	Updated content	Updated Clock.	N/A
July 26, 2023	Updated content	Updated Configuring spatial domains.	N/A
July 26, 2023	New content	Added a new section: Custom containers.	N/A
July 26, 2023	Updated content	Updated AWS SimSpace Weaver versions for release 1.14.0.	N/A
July 6, 2023	New content	Added a new section: PathfindingSample console client fails to connect.	N/A
June 7, 2023	Updated content	Updated AWS SimSpace Weaver versions for release 1.13.1.	N/A
May 15, 2023	New content	Added a new section: Using snapshots with AWS CloudFormation.	N/A
April 29, 2023	New content	Added content for release 1.13.0. For more information, see AWS SimSpace Weaver versions.	AWS SDK: 2023-04-28

Date	Change	Documentation updates	API versions updated
March 27, 2023	New content	Added a section that describes the maximum duration of simulatio ns. Added notes in the tutorials for release 1.12.3, which added support for the `--maximum-duration` parameter to SimSpace Weaver app SDK scripts.	N/A
March 9, 2023	Changed content	Clarified that we only provide instructions for Docker on Windows and for Windows Subsystem for Linux (WSL), and that WSL (and any other Linux environment) is unsupported.	N/A
February 28, 2023	New content	Added a chapter that describes SimSpace Weaver versions.	N/A
February 28, 2023	Changed content	Changed content about authentication to include the use of AWS IAM Identity Center and named profiles for the AWS Command Line Interface (AWS CLI).	N/A

Date	Change	Documentation updates	API versions updated
February 17, 2023	New content	Added a section about managing your resources with AWS CloudForm ation.	N/A
January 23, 2023	New content	Added instructions for debugging local simulations.	N/A
November 29, 2022	Service launch	Released the User Guide and API Reference for SimSpace Weaver.	AWS SDK: 2022-11-29

Glossary

This glossary defines terms that are specific to AWS SimSpace Weaver.

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.

A

app	Executable code (also called *binaries*) that you create. The term *app* can refer to the code or a running instance of that code. An app encapsulates simulation behavior. Apps create, delete, read, and update entities.
app SDK	A software development kit (SDK) that you use to integrate an app with SimSpace Weaver. The SDK provides APIs for reading and writing entity data and tracking simulation time. For more information, see SimSpace Weaver app SDK.

C

client	Processes (or their definitions) that exist outside of SimSpace Weaver and interact with the simulation through a custom app or service app. You can use a client to view or change the simulation state.
clock	An abstraction of SimSpace Weaver's internal scheduling processes. The clock publishes ticks to apps to maintain time synchronization. Each simulation has its own clock.
clock rate	The number of ticks per second that the clock publishes to apps. For more information about supported clock rates, see SimSpace Weaver endpoints and quotas.
clock tick rate	See clock rate.
compute resource unit	A unit of compute resources (processor and memory) on a worker. A single instance of an app is normally allocated 1 compute resource unit. You can allocate more than 1 compute resource unit for each app.
custom app	A type of app that you use to read and interact with the state of the simulation. Custom apps can create entities in the simulation but don't own them. When a custom app creates an entity, it must transfer the

	entity to the spatial domain. You control the lifecycle of a custom app using the app APIs. For more information about the SimSpace Weaver APIs, see SimSpace Weaver API references.
custom domain	A domain that contains custom apps.
custom partition	The partition of a custom app.

D

deadline	An actual time by which an operation (such as processing for a tick) should be complete.
domain	A group of app instances that run the same executable code (app binary) and have the same launch options.

E

endpoint (service)	A fully-qualified domain name (FQDN) that programs (such as the AWS Command Line Interface) use to connect to the SimSpace Weaver service.
endpoint (simulation)	An IP address and port number that clients use to connect to connect to a simulation. You can configure endpoints on custom apps and service apps.
entity	Customer data objects (or their definitions). Entities can be static (remain in one location) or dynamic (move through the simulation space). For example, people and buildings in a simulation.

I

index (simulation)	A description of the spatial properties of a simulation, including its spatial boundaries and coordinate system.

L

lifecycle (of an app)	A description of the expected logical steps that an app goes through during a simulation. Lifecycles are either *managed* (SimSpace Weaver starts and stops the app) or *unmanaged* (you start and stop the app).

load (entity field data)	Read entity field data from the State Fabric.

P

partition	A segment of shared memory on a worker. Each partition contains a discrete subset of entities within a domain. Each app has an assigned partition. An app owns all of the entities in its partition. When an app creates an entity, it creates it in its partition. When entities move from one partition to another partition, ownership transfers from the source partition's app to the destination partition's app.

R

resource unit	See ???.

S

schema	A YAML or JSON document that describes the configuration of a simulation. SimSpace Weaver uses a schema to create a simulation resource.
service app	A type of app that you use to read and interact with the state of the simulation. Service apps can create entities in the simulation but must transfer them to the spatial domain. SimSpace Weaver manages the lifecycle of a service app, and starts 1 (or more, as specified in your simulation schema) on each worker in your simulation.
service domain	A domain that contains service apps.
service partition	The partition of a service app.
simulation (resource)	An abstraction of a compute cluster that runs a simulated virtual space. You can have multiple simulations. You configure a simulation using a schema.
spatial app	A type of app that encapsulates the core simulation logic. Each spatial app owns 1 (and only 1) partition.
spatial domain	A domain that contains spatial apps.

spatial partition	The partition of a spatial app.
State Fabric	SimSpace Weaver's in-memory database. The State Fabric stores the state of simulations, including entities and internal SimSpace Weaver data.
store (entity field data)	Write entity field data to the State Fabric.
subscription	A long-running request for a specific app instance to receive data from a subscription area. The subscribing app uses a subscription to discover changes to entities inside the subscription area.
subscription area	A 2-dimensional region of the simulation space. A subscription refers to a subscription area. A subscription area can span more than 1 partition, and also include parts of partitions. A subscription area is continuous within its defined bounds.

T

tick	A discrete value for time (either wall-clock time or simulation time). Apps can iterate faster than the tick duration, but are expected to write specified ticks within specific deadlines. All operations for all apps for a given tick must complete before the next tick can start.
tick rate	See clock rate.
time (actual)	The current time from the perspective of reality. SimSpace Weaver uses a 64-bit POSIX timestamp which is the number of nanoseconds since the Unix epoch (January 1, 1970, 00:00:00 UTC).
time (simulation)	The current time from the perspective of the simulation. SimSpace Weaver uses a 64-bit integer logical tick counter, which might not directly correspond to the actual time.

W

worker	An Amazon Elastic Compute Cloud (Amazon EC2) instance that runs simulation code.