
Amazon AppStream

Developer Guide



Amazon AppStream: Developer Guide

Copyright © 2014 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

The following are trademarks of Amazon Web Services, Inc.: Amazon, Amazon Web Services Design, AWS, Amazon CloudFront, Cloudfront, Amazon DevPay, DynamoDB, ElastiCache, Amazon EC2, Amazon Elastic Compute Cloud, Amazon Glacier, Kindle, Kindle Fire, AWS Marketplace Design, Mechanical Turk, Amazon Redshift, Amazon Route 53, Amazon S3, Amazon VPC. In addition, Amazon.com graphics, logos, page headers, button icons, scripts, and service names are trademarks, or trade dress of Amazon in the U.S. and/or other countries. Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon.

All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon AppStream?	1
Advantages of Streaming Your Application	1
What Can You Do with Amazon AppStream?	2
How Does Amazon AppStream Work?	2
Amazon AppStream Components	3
Architectural Overview of Amazon AppStream	4
Amazon AppStream Application Lifecycle	6
Can My Application Run on Amazon AppStream?	6
Supported Operating Systems	7
Hardware Specifications	7
Video Input Specifications	7
Audio Specifications	7
Bandwidth Requirements	7
Persistent Data	8
User Input	8
Regions	8
Tools for Amazon AppStream	8
Amazon AppStream SDK	8
Amazon AppStream SDK for Java	9
Amazon AppStream Console	9
Downloads	10
Amazon AppStream SDK	10
Amazon AppStream SDK for Java	10
Other Files to Download	10
Get Started	12
Service Requirements	14
Sign Up for AWS	14
Option 1: Preview Amazon AppStream	14
Step 1: Create a key pair	15
Step 2: Create the standalone mode	16
Step 3: Stream	18
Option 2: Preview Your Application	19
Step 1: Integrate the SDK	20
Step 2: Create a key pair	20
Step 3: Create the standalone mode	21
Step 4: Copy your application	25
Step 5: Stream	26
Option 3: Deploy a Streaming Application	27
Step 1: Build	28
Step 2: Deploy	28
Step 3: Stream	48
Where to Go Next	50
Build an Application	51
Build a Streaming Application	51
Design Considerations	52
Add Streaming to Your Application	52
Test Your Streaming Application	65
Build an Installer	73
Build an Entitlement Service	73
Design Considerations	74
Build the Entitlement Web Service	74
Publish Your Entitlement Service	79
Sample Entitlement Request and Response	80
Build a Client	80
Design Considerations	81

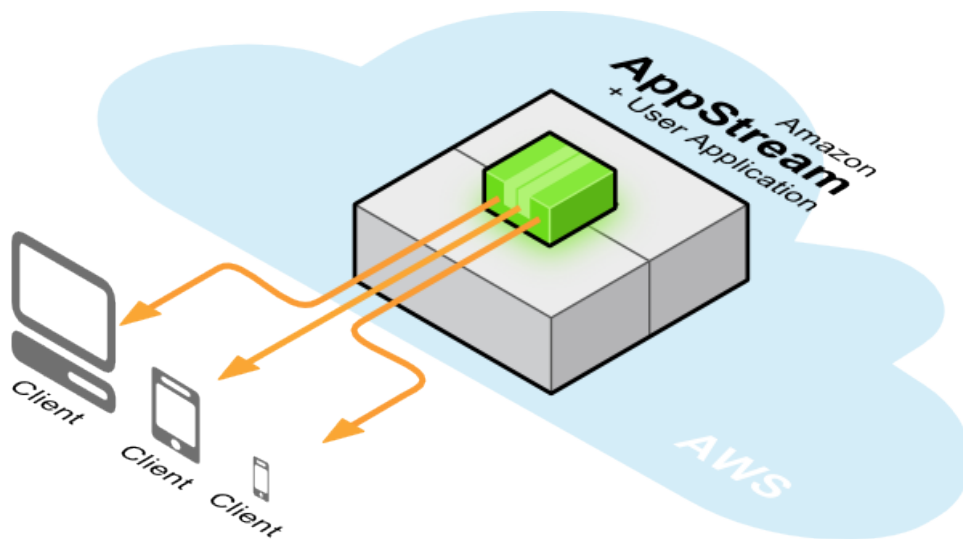
Build a Client for Android	81
Build a Client for iOS	89
Build a Client for OS X	101
Build a Client for Windows	112
Codec and Open Source Licensing	121
Deploy Your Application	123
Prerequisites	123
Upload the Application Installer to Amazon Simple Storage Service	123
Create a Pre-signed URL	124
Deploy Your Streaming Application	126
Manage Your Application	135
View All Applications	135
View Application Summary	136
Edit an Application	137
Clone an Application	140
Archive an Application	145
Enable Logging on an Application	147
AppStream Log Names	148
Default Amazon AppStream Logs	148
Custom Amazon AppStream Logs	150
.....	150
Enabling Amazon AppStream Logging Programmatically	151
Increase Your Service Limits	152
Security Considerations	154
Controlling Access Using IAM	154
Example IAM User Policies for Amazon AppStream	155
Security Best Practices	157
Versioning	157
Multi-Factor Authentication	157
Key Rotation	158
Use A Strong Password For Remote Management	158
Restrict Access to Your Streaming Application	158
Troubleshooting Amazon AppStream	159
Deployment Problems	159
Is Your Installer Corrupted?	159
Has Your Pre-Signed URL Expired?	159
Does Your Pre-Signed URL Use HTTP Protocol?	159
Streaming Problems	160
Error Codes	160
APPLICATION_DELETION_FAILED	160
APPLICATION_INSTALLATION_FAILED	160
APPLICATION_LAUNCH_FAILED	161
APPLICATION_INSTALLATION_NOT_SILENT	161
APPLICATION_INSTALLATION_TIMED_OUT	161
APPLICATION_LAUNCH_TIMED_OUT	161
APPLICATION_RUNTIME_FAILURE	162
INTERNAL_FAILURE	162
SDK_VERSION_DETECTION_FAILED	162
S3_URL_INVALID	162
Amazon AppStream REST API	164
Hypertext Application Language	164
Making HTTP Requests	165
Limits on Request Rates	165
HTTP Header Contents	165
HTTP Request Body	166
HTTP Responses	166
Signing Requests	168
Handling Errors	169

API Error Codes (Client and Server Errors)	169
Catching Errors	171
Error Retries and Exponential Backoff	172
Resources	172
AppStream	172
Applications	173
Application	174
Application Errors	176
Application Error	177
Application Status	177
Sessions	179
Session	179
Session Status	180
Link Relations	181
appstream:applications	181
application:by-id	181
application:create	181
application:update	183
application:archive	184
application:reactivate	184
application:delete	185
application:status	185
application:errors	185
application:sessions	185
session:by-id	185
session:entitle	185
session:status	186
session:terminate	186
Common Link Relations	187
Product Updates	188
Release for June 20, 2014 (Latest)	188
Up to 20% performance improvement on 64-bit iOS devices	188
Simplified sample streaming application code	188
Remote debugging in standalone mode	189
Other Updates	189
Release for May 30, 2014	189
Utilization Log is in Coordinated Universal Time (UTC)	189
Utilization Log Contains GPU Metrics	189
Simplified the Application Resource Properties	189
Other Updates	189
Release for May 9, 2014	189
Logging for Utilization Metrics	190
YUV444 Color Subsampling Support	190
Other Updates	190
Release for April 22, 2014	190
Standard and User-defined Logging	190
Updated OpenSSL Version	190
Other Updates	190
Release for March 28, 2014	191
New Billing Practice	191
SDK Version Detected from the Files instead of the Registry	191
Other Updates	191
Release for March 7, 2014	191
Amazon AppStream Is Available to Anyone with an AWS Account	191
Improved Deployment Error Messages	191
Example Client Application for Mac OS X	191
Increase Your Service Limits	191
Other Updates	192

Release for February 14, 2014	192
Improved Deployment Experience	192
Silent Installer Requirement	192
Other Updates	192
Release for January 24, 2014	192
Single Download for the Amazon AppStream SDK	192
Improved Console Load Time	193
Support for Android Hardware Decoding	193
Other Updates	193
Document History	194

What is Amazon AppStream?

The Amazon AppStream web service deploys your application on Amazon Web Services (AWS) infrastructure and streams input and output between your application and devices such as personal computers, tablets, and mobile phones. Your application's processing occurs in the cloud, so it can scale to handle vast computational loads. Devices need only display output and return user input, so the client application on the device can be lightweight in file size and processing requirements.



In addition to converting existing applications into interactive applications, you can use Amazon AppStream to invent new types of applications optimized for a user interface that runs locally on a device while processing occurs in the cloud. You can also create a hybrid scenario, in which part of your application runs on AWS, and part of it runs natively on end-user devices.

Amazon AppStream currently supports interactive streaming of applications that run on Microsoft Windows Server 2008, and client applications running on Kindle/FireOS, Android, Apple iOS and OS X, and Microsoft Windows.

Advantages of Streaming Your Application

Interactively streaming your application from the cloud provides several benefits:

- **Remove Device Constraints**—Applications requiring powerful hardware can run in the cloud and be interactively streamed to low-end devices.
- **Multidevice Support**—With the separation of computation and user interface that Amazon AppStream provides, you can write your application once, and create a client application for each end-user device your application supports. When a new device is released, you simply release a new client application that supports the new device. Your customers will also be able to run your application from multiple devices, such as a Microsoft Windows laptop at work, a Kindle Fire tablet at home, and an iPhone mobile phone while they ride the bus.
- **Immediately Available**—By interactively streaming your application your end-users can start using your application or game immediately, without having to download a large file and install a native application. Applications intended to run natively on devices can use Amazon AppStream to stream a version of their application to clients while the download and install runs in the background processing of the end-user device.
- **Easy Updates**—You update your application by simply providing a new version of your application to Amazon AppStream. If the client display and user input logic is unchanged, that's all you need to do to immediately upgrade all of your users without any action on their part.
- **Improve Security**—Hosting your application on Amazon AppStream keeps your application's executable files out of the hands of malicious users who could decompile or redistribute your code. You also benefit from the on-site security protecting AWS data centers.

What Can You Do with Amazon AppStream?

Hosting your application on AWS infrastructure means your application can scale to meet demand, removing hardware limitations. All this potential power makes new types of applications possible:

- **Test drive applications and games**—You can use Amazon AppStream to run a complex application or high-resolution game remotely using a lightweight (5 MB) client application so users can determine whether to purchase the native version. If they decide to purchase the native version, they can continue to use the application remotely while the native version is downloaded in the background.
- **Games direct to television**—Because your client is lightweight, it can run on a smart TV. Consumer electronics manufacturers can use Amazon AppStream to stream leading games to enhanced television sets that accept input from a controller, creating a streaming games ecosystem.
- **Computer-assisted design (CAD) software**—Resource-intensive applications such as CAD programs and video rendering currently need to be run on high-powered local machines. By streaming these applications from Amazon AppStream, you make it possible for your users to interact with your software from low-cost and lightweight devices, removing a barrier to adoption.
- **Live Video rendering**—Video production teams shoot a scene from many different angles and locations and then deliver the footage to a centralized location to be rendered and merged. Directors often wait until the video is processed in order to determine whether the scene is acceptable. Using a video rendering application hosted on Amazon AppStream, you can broadcast multiple video feeds to the application, render the video in real-time, and stream all the angles and shots back to the director for evaluation, speeding up the filming process.
- **Hybrid applications**—You can stream part of your application from Amazon AppStream and run part of your application natively. This gives you the best of both worlds: run computation-intensive but latency-tolerant components on Amazon AppStream and run only those components that require real-time responsiveness natively on the device.

How Does Amazon AppStream Work?

Amazon AppStream provides a framework for you to host an application on AWS infrastructure and stream the input and output of the application to clients running on consumer devices such as PCs, mobile phones, and tablets.

In building your product, you provide the logic for the application, the client, and user authentication and authorization. This gives you the flexibility to create an end-to-end solution specifically tailored to your business and customer requirements.

The following topics describe the components that make up Amazon AppStream and explain how they work together to provide a high-definition, responsive experience for your users.

Topics

- [Amazon AppStream Components](#) (p. 3)
- [Architectural Overview of Amazon AppStream](#) (p. 4)
- [Amazon AppStream Application Lifecycle](#) (p. 6)

Amazon AppStream Components

Streaming an application from Amazon AppStream involves several components working together, some are AWS products, and others you supply.

Topics

- [Amazon AppStream Host](#) (p. 3)
- [Application](#) (p. 3)
- [Clients](#) (p. 3)
- [Entitlement Service](#) (p. 4)

Amazon AppStream Host

Amazon AppStream hosts your application on EC2 instances. Each host runs on a very large instance type called a GPU instance. Each GPU instance provides large amounts of parallel processing power. For more information, see [GPU Instances](#) in the *Amazon Elastic Compute Cloud User Guide*.

Application

The *application* is the code that you plan to host on Amazon AppStream. This can be a pre-existing application that you modify or a new application that you design specifically to work with Amazon AppStream. To stream content from Amazon AppStream, your application calls `XStxServerAPI` functions to send audio and video to clients and listens to `XStxServerAPI` callback functions to receive input from clients. These functions are described in the Amazon AppStream SDK.

The Amazon AppStream SDK provides a sample implementation of an application that runs on Microsoft Windows Server 2008 or later. You can use it as a guide to making an application compatible with Amazon AppStream. For more information, see [Build an Amazon AppStream Application](#) (p. 51).

Clients

Clients are lightweight applications that run on consumer devices. They decode the audio and video output of your application and display it on the device. They also encode user input from the device and return it to the application. Thus, they provide a fully interactive experience to your users.

Each device type requires a client written for that platform. For example, if you want your customers to be able to access your application from both iPhone mobile phones and Android tablets, you would provide two clients, one for iOS and one for Android. Both clients access the same application. To support new device types, simply create a new client for that device. You do not have to change the application to support a new client.

To interact with applications streaming from Amazon AppStream, the clients call `XStxClientAPI` functions and listen to `XStxClientAPI` callback functions. These functions are available in the Amazon AppStream SDK.

The client SDKs provides sample implementations of clients that run on Microsoft Windows PCs, iPhone mobile phones, and Android tablets. Depending on your use case, you may be able to use these sample implementations with your application without modification. If your application requires additional functionality, you can use the sample implementations as a starting place to create your own clients. For more information, see [Build a Client \(p. 80\)](#).

Entitlement Service

An *entitlement service* authenticates and authorizes users. It is the gatekeeper between clients and your application, ensuring that only those clients entitled to access your application can do so. Your entitlement service may authenticate users in a variety of ways: by comparing user login credentials to a list of subscribers in a database, by using an external login service such as [Login with Amazon](#), or by simply authenticating all clients.

After the entitlement service successfully authenticates a user, it calls the Amazon AppStream service to create a new streaming session for the client. To do so, it can either call the [Amazon AppStream REST API \(p. 164\)](#) directly or use the Java interface provided in the Amazon AppStream SDK. The Amazon AppStream SDK is the quickest way to create an entitlement service because it provides the functionality for signing the requests your entitlement service sends to Amazon AppStream.

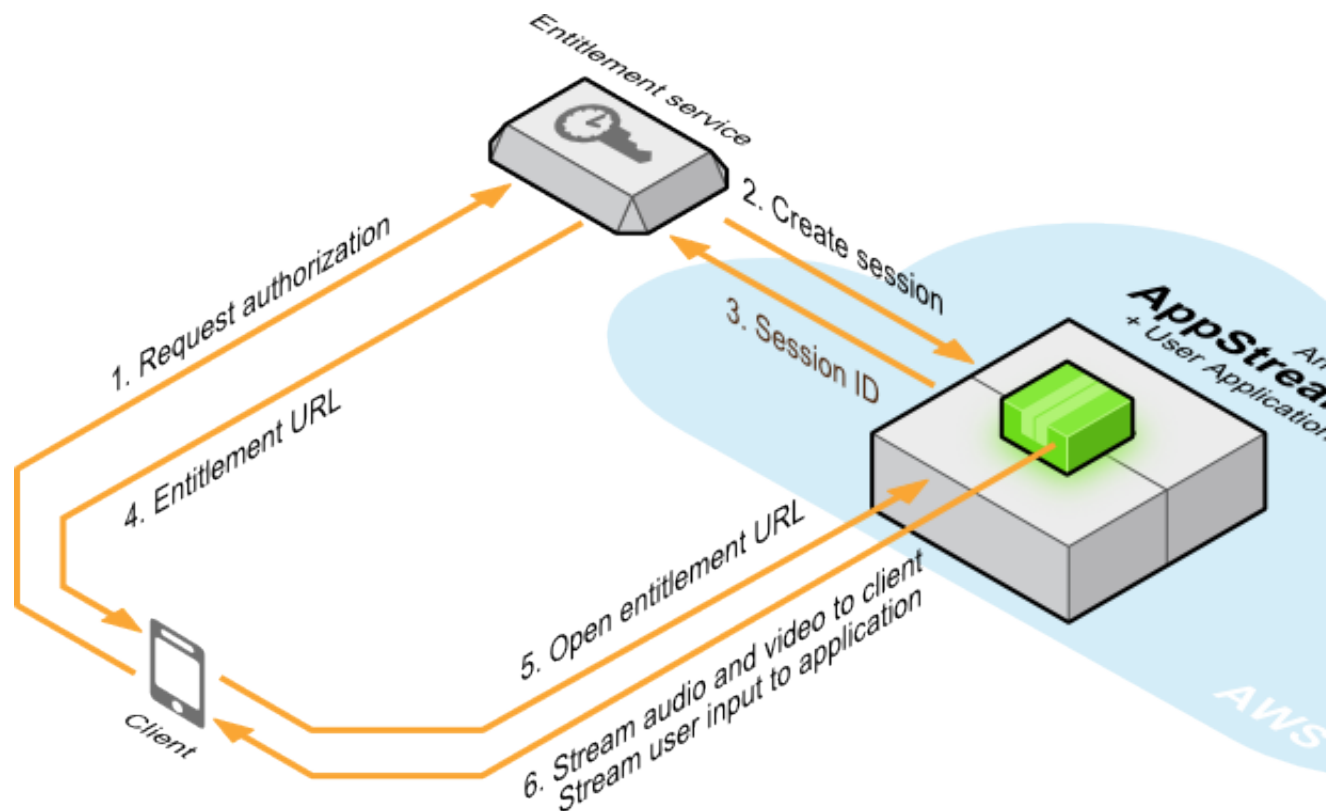
After the entitlement service creates a new session, it returns the session identifier to the authorized client as an entitlement URL. The client then uses the entitlement URL to connect to the application.

Amazon AppStream provides a sample implementation of an entitlement service written in Java, as well as an AWS CloudFormation template that you can use to deploy the sample entitlement service on AWS. You can use the sample entitlement service during development and testing of your product.

For your production release, you'll probably want to create your own entitlement service. The sample entitlement service is a great place to start when writing your own service. You can download the sample entitlement service from the links in [Downloads \(p. 10\)](#).

Architectural Overview of Amazon AppStream

The following diagram illustrates how the various components of an Amazon AppStream your product work together. For information about the individual components, see [Amazon AppStream Components \(p. 3\)](#).



How a Client Connects to the Application

In order to connect to an application hosted on Amazon AppStream, the client needs the service to create a new streaming session, and the client needs credentials to access that session. The authentication and authorization of clients to access applications is handled by an entitlement service, which handles the negotiation between client and Amazon AppStream.

The steps of authorizing a client to access an application are as follows:

1. **Request authorization**—The client application calls the entitlement service and requests authorization for a set of user credentials.
2. **Create a session**—If the entitlement service successfully authenticates the user credentials and verifies they are authorized to access the application, the entitlement service calls the Amazon AppStream service to create a new client session.
3. **Return session ID**—The Amazon AppStream service creates a new client session and returns the session identifier to the entitlement service.
4. **Return entitlement URL**—The entitlement service uses the session identifier returned by the Amazon AppStream service to create an entitlement URL, which it returns to the client.
5. **Open entitlement URL**—The client redeems its entitlement to access the application by opening the entitlement URL. When it does, the Amazon AppStream service redirects the client to the IP address of the Amazon AppStream host hosting the application.
6. **Stream output and receive user input**—Using the `XStxServerAPI` and `XStxClientAPI` interfaces, the application streams audio and video to the client, and the client sends user input to the application through callback functions. The Amazon AppStream service manages the connection for best performance given the current network conditions.

Amazon AppStream Application Lifecycle

When you deploy and manage your application on Amazon AppStream it goes through a series of states. For more information, see [Deploy Your Streaming Application to Amazon AppStream \(p. 123\)](#) and [Manage Your Application \(p. 135\)](#).

Building

During the Building state, Amazon AppStream deploys your application. This has several substates, which are displayed in the console.

- **Preparing environment**—Amazon AppStream allocates the IT infrastructure required to host your application.
- **Copying application**—Amazon AppStream copies the installer for your application from Amazon S3 to the host. A presigned URL that you provide gives the installer access to the content stored in your AWS account. The installer installs your application and its dependency files.
- **Installing application**—Amazon AppStream calls the installer, including any command-line parameters you provided, to install your application on the host.
- **Creating AMI**—Amazon AppStream creates an Amazon Machine Image (AMI) of the host with your application installed. For more information, see [Amazon Machine Images \(AMI\)](#) in the *Amazon Elastic Compute Cloud User Guide*.

Active

During the Active state, your application is ready to accept user connections. For the limited release, this state corresponds to **Pre-production** in the console.

Archiving

The Archiving state begins after you archive your application but before it is fully archived. This state corresponds to **Pre-production (archive pending)** in the console. In this state, the application continues to stream to existing client connections, but no longer accepts new client connections. When all client connections have concluded, Amazon AppStream moves the application into the **Archived** state.

Archived

Once it is fully archived, your application no longer accepts client connections. This state corresponds to **Archived** in the console.

Error

A problem during deployment puts your application into the Error state.

Can My Application Run on Amazon AppStream?

Many existing applications can be adapted for Amazon AppStream streaming simply by changing how the application handles input and output. You just update the application to route that data through Amazon AppStream APIs instead of a local machine. For more information, see [Build an Amazon AppStream Application \(p. 51\)](#).

You also need to consider whether your application can run on the infrastructure provided by Amazon AppStream. The following topics detail the specifics of the virtualized hardware provided by Amazon AppStream and other requirements for Amazon AppStream streaming.

Supported Operating Systems

The application must be able to run on the Microsoft Windows Server 2008 or later operating system. Windows Server 2008 is a 64-bit operating system; 32-bit applications are supported through the WoW64 extensions. If your application has other dependencies, such as the .NET Framework, you can include them as part of your application installer.

Hardware Specifications

The Amazon AppStream servers that Amazon AppStream uses to host applications are GPU instances provided by Amazon Elastic Compute Cloud (Amazon EC2). The GPU instances have the following virtualized hardware. For more information, see [GPU Instances](#) in the *Amazon Elastic Compute Cloud User Guide*.

- CPU: 10 EC2 compute units (8 virtual cores at 2.5 GHz each)
- RAM: 15 GiB
- Instance storage: 50 GiB
- GPU: 1 * NVIDIA GK104 GPU with NVIDIA GRID K520
- GPU memory: 4 GiB
- I/O performance: High (we recommend 2 Gbps/instance)
- EBS-optimized: Yes (500 Mbps)
- 64-bit platform: Windows or Amazon Linux
- HVM only

Video Input Specifications

Amazon AppStream accepts YUV 420 video input from the application and outputs YUV 420 to the client. Amazon AppStream does not perform any color conversion internally.

Audio Specifications

Your application can push audio to the Amazon AppStream library or it can make use of the automatic audio capture feature that Amazon AppStream provides. If your application uses automatic audio capture, the application simply writes audio as a normal Microsoft Windows application would. If the application pushes audio to the Amazon AppStream library, observe the following audio specifications:

- 48000Hz sampling rate
- 2 interleaved channels
- 16 bit signed

Bandwidth Requirements

When accessing an application hosted on Amazon AppStream, the client must be continuously connected to the Internet with a minimum bandwidth of 3 Mbps.

Amazon AppStream recommends at least 3 Mbps for streaming video at 720 pixels at 30 frames per second (720p30). When more bandwidth is available, Amazon AppStream allows the encoding rate to

go as high as .2 bits per pixel, which at 720p30 is about 5.53 Mbps. When less bandwidth is available, Amazon AppStream allows the encoding rate to go as low as .02 bits per pixel, which at 720p30 is about 553 Kbps. Amazon AppStream adapts the video bit rate based on available bandwidth. If Amazon AppStream measures the available bandwidth at 3 Mbps, for example, Amazon AppStream sets the encoding bit rate to meet that constraint.

Persistent Data

Because Amazon AppStream hosts your application on Amazon AppStream hosts, any data stored on the server is lost when the client session ends. If your application needs to persist data between client sessions your application should record the data in a persistent store such as Amazon S3, Amazon RDS, or DynamoDB.

User Input

You can stream a variety of user inputs from the client to your application:

- **Keyboard**—transmits keyboard data from the client to the application.
- **Mouse**—transmits mouse move and mouse click data from the client to the application.
- **Touch**—transmits multi-touch and gesture data from the client to the application.
- **Raw input**—transmits a raw stream of bytes from the client to the application. You can use this to transmit user data that does not fit the keyboard, mouse, or touch models. For example, accelerometer data.

Regions

Amazon Web Services run on servers in data centers around the world. These are organized by geographical region. When you launch an application on Amazon AppStream, you must specify which region to launch it into. You might choose a region to reduce latency, minimize costs, or address regulatory requirements. For the list of regions and endpoints supported by Amazon AppStream, go to [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

Amazon AppStream currently supports only the US East (Virginia) region.

Tools for Amazon AppStream

Amazon AppStream provides several tools to simplify the process of developing application solutions.

Amazon AppStream SDK

The Amazon AppStream SDK simplifies the process of adding streaming to your application and makes it easier to build clients for Windows devices. It provides C header files and libraries with the functionality needed to stream your application from Amazon AppStream; as well as receive the streamed content in a client. The Amazon AppStream SDK; includes the source code for a sample application and a client as well as a pre-compiled client file that you can use to connect to an application. You can use these sample implementations to test streaming an application from Amazon AppStream without writing any code. You can obtain this SDK from [Downloads \(p. 10\)](#).

Amazon AppStream SDK for Java

The Amazon AppStream SDK for Java includes functions you can call to interact with the Amazon AppStream service. These functions wrap the [Amazon AppStream REST API \(p. 164\)](#) and handle details like signing the requests sent to the Amazon AppStream service. The most common use of the Amazon AppStream SDK for Java is to write an entitlement service that authorizes user access to your applications. Your entitlement service calls into Amazon AppStream to create new client sessions. You can also use the Amazon AppStream SDK for Java functions to automate Amazon AppStream. For example, you could write an application to bulk-add applications.

Amazon AppStream Console

The Amazon AppStream console is a graphical interface that you can use to add and manage applications. With it, you fill out web forms to specify the details of streaming applications and view the details of existing applications. The console is available online at <https://console.aws.amazon.com/appstream/>.

Downloads

Amazon AppStream SDK

The Amazon AppStream SDK provides tools for developing Amazon AppStream-compatible Windows streaming applications as well as client applications for FireOS, Android, iOS, Mac OS X, and Windows. You will need the SDK to complete [Get Started \(p. 12\)](#).

To download the Amazon AppStream SDK, go to https://s3.amazonaws.com/appstream-sdk/AppStreamSDK_1.5.0.51.zip.

The SDK also contains a compiled version of a Windows client application that can connect to an Amazon AppStream streaming application. To use the sample client application, you need to do the following:

1. Download the [DirectX Software Development Kit](#) from the Microsoft Download Center and install it.
2. Download the [Visual C++ Redistributable for Visual Studio 2012](#) from the Microsoft Download Center and install it.

Amazon AppStream SDK for Java

The Amazon AppStream SDK for Java includes functions you can call to interact with Amazon AppStream. These functions wrap the [Amazon AppStream REST API \(p. 164\)](#) and handle details like signing the requests sent to Amazon AppStream. The most common use of the Amazon AppStream SDK for Java is to write an entitlement service that authorizes user access to your streaming applications. and to automate Amazon AppStream. applications.

To download the Amazon AppStream SDK for Java, go to <https://github.com/aws-labs/aws-appstream-sdk-java/>.

Other Files to Download

In addition to the Amazon AppStream SDK, you can download the following files:

- For an AWS CloudFormation template to use Amazon AppStream standalone mode, go to <https://s3.amazonaws.com/appstream-public/AppStreamDeveloper.template>.

- For an Amazon AppStream entitlement service template, go to <https://s3.amazonaws.com/appstream-sdk/appstreamEntitlementService.template>.
- For the Amazon AppStream entitlement service sample, go to <https://s3.amazonaws.com/appstream-sdk/sample-entitlement-service.jar>.
- For the source code for the sample entitlement service, go to <https://s3.amazonaws.com/appstream-sdk/sample-entitlement-service-src.zip>.

Get Started

Amazon AppStream deploys interactive applications on AWS infrastructure, and streams input and output between the application and clients running on end-user devices such as personal computers, tablets, and mobile phones.

Using Amazon AppStream involves five basic steps:

1. Add streaming functionality to a new or existing application using the server APIs provided in the Amazon AppStream SDK.
2. Write a client application for each platform or device your streaming application will support. You do this using the client APIs provided in the Amazon AppStream SDK.
3. Create an entitlement service that authenticates clients and authorizes them to connect to your application. This gives you the ability to monetize your application through payment models such as user subscriptions or pay-to-play. The entitlement service interacts with Amazon AppStream by calling the Java wrapper classes provided by the Amazon AppStream SDK or by calling the REST API directly.
4. Add your application to Amazon AppStream using the AWS Management Console or programmatically by using the REST APIs.
5. Distribute your client application to users. You can do this directly or by submitting your client to a distribution site such as the Amazon AppStore.

In the following sections, you'll learn how to get set up to use Amazon AppStream. First, we'll show you the requirements to use Amazon AppStream. Then we'll show you where to sign up for an AWS account

and show you where to download the Amazon AppStream SDK and install the dependency files required by the client application. Finally, we'll show you the following options to try out Amazon AppStream.

For more information, see the following topics:

• [Getting started with Amazon AppStream](#)

• [Quickly see a sample streaming application on a single client application without writing any code. You can only stream to a single user.](#)

• [See your application streamed through Amazon AppStream. You'll need to add streaming to your application by integrating the Amazon AppStream SDK and then use the Amazon AppStream Service Simulator to test your streaming application. You can only stream to a single user.](#)

• [Do all the steps to deploy a sample streaming application to Amazon AppStream and stream to multiple client applications. You'll build a sample streaming application, deploy that streaming application to Amazon AppStream, deploy the sample entitlement service, and then stream to multiple users.](#)

• [Getting started with Amazon AppStream](#)

Topics

- [Service Requirements \(p. 14\)](#)
- [Sign Up for AWS \(p. 14\)](#)
- [Option 1: Preview Amazon AppStream \(p. 14\)](#)
- [Option 2: Preview Your Application \(p. 19\)](#)
- [Option 3: Deploy a Streaming Application \(p. 27\)](#)
- [Where to Go Next \(p. 50\)](#)

Service Requirements

The streaming application must run in the 64-bit version of Microsoft Server 2008 R2 or earlier.

Client applications must run in one of the following operating systems:

- Android 2.3 (Gingerbread) or later
- Apple iOS 7.0 or later
- Mac OS X Mountain Lion (10.8.5) or later
- Microsoft Windows 7 or later

Sign Up for AWS

You need an AWS account in order to use Amazon AppStream.

If you do not have an AWS account, use the following procedure to create one.

To sign up for AWS

1. Go to <http://aws.amazon.com> and click **Sign Up**.
2. Follow the on-screen instructions.

AWS notifies you by email when your account is active and available for you to use.

Option 1: Preview the Sample Streaming Application on Amazon AppStream Standalone Mode

The Amazon AppStream standalone mode uses an AWS CloudFormation template to create an Amazon EC2 instance in your account and then run a streaming application from that instance. The Amazon AppStream standalone mode is recommended for debugging your streaming application during development. You can use an example streaming application provided by Amazon AppStream in standalone mode to check whether your Amazon EC2 instance can stream a streaming application. After testing your EC2 instance, you can then use your own streaming application in standalone mode. To connect to the streaming application, use the precompiled Windows client application in the Amazon AppStream SDK to connect to a streaming application in standalone mode. All the files you need to complete this section are available in [Downloads \(p. 10\)](#).

Note

You may incur charges when you use Amazon AppStream standalone mode. Amazon AppStream standalone mode uses your own EC2 instance. Use the [Simple Monthly Calculator](#) to estimate your monthly cost. For more information, see [Amazon EC2 Pricing](#).

To complete this section, you will need the following:

- A computer running Microsoft Windows that is connected to the Internet. You will use this computer to administer the Amazon EC2 instance and install your application on the EC2 instance.
- A device or computer that is connected to the Internet. You will stream your application from your Amazon EC2 instance to this device or computer.

- Android 2.3 (Gingerbread) or later
- Apple iOS 7.0 or later
- Mac OS X Mountain Lion (10.8.5) or later
- Microsoft Windows 7 or later

In this section, you will do the following:

Topics

- [Step 1: Create a key pair \(p. 15\)](#)
- [Step 2: Create the standalone mode \(p. 16\)](#)
- [Step 3: Stream \(p. 18\)](#)

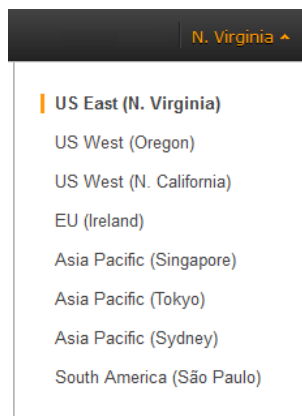
Step 1: Create a key pair

To begin, you will create a key pair in Amazon EC2, which you will use as a parameter for the AWS CloudFormation template. The key pair is required to create the Amazon AppStream standalone more.

If you already have an existing key pair, you can skip to the next section and use your existing key pair.

To create a key pair

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select **US East (N. Virginia)**.



3. In the navigation pane, click **Key Pairs**.
4. Click **Create Key Pair**.
5. For **Key pair name** enter **AppStreamKeyPair** and then click **Create**.
6. The private key file (`AppStreamKeyPair.pem`) is automatically downloaded by your browser. Save the private key file in a safe place.

Important

This is the only chance for you to save the private key file. You will need this file to connect to the instance you will create in the next step.

Your new key pair appears in the Amazon EC2 console. In the next step, you will use the key pair to create the standalone mode.

Step 2: Create the standalone mode

Now that you have your key pair, you need to set up the standalone mode for the example streaming application. On successful creation, your account will have a running instance with the example streaming application.

1. Open the [AWS CloudFormation console](#).
2. From the navigation bar, select the same region where you created the key pair.

Important

Do not select a region that is different from your key pair was created.

3. Click **Create Stack** or **Create New Stack**.
4. For **Name**, type **MyAppStreamStack**.
5. Select **Provide an S3 URL** to template and type or paste `https://s3.amazonaws.com/appstream-public/AppStreamDeveloper.template` and then click **Next**.
6. In **Specify Parameters**, do the following:
 - For **KeyPairName**, type **AppStreamKeyPair**, or if you have it, your existing key pair in the same region.
 - For **MicrosoftVirtualAudioDrivers**, type **Yes**.
 - For **NvidiaGRIDDrivers**, type **Yes**.
 - Click **Next**.
7. In **Options**, do the following:
 - For **Key**, type **Name**.
 - For **Value**, type **MyAppStreamStack**.
 - Click **Next**.
8. In **Review**, check that the parameters you entered are correct and then click **Create**.

Review

Template

Name	MyAppStreamStack
Template URL	https://s3.amazonaws.com/appstream-public/AppStreamDeveloper.template
Description	AppStream Stand-Alone is subject to the AWS Customer Agreement and AppStream Service Terms. AppStream Stand-Alone may only be used for development and testing; you may not host connections from end users on your AppStream Stand-Alone Instance. Do not download, transmit or otherwise distribute AppStream materials provided with the AppStream Stand-Alone instance.
Estimate cost	Link is not available

Parameters

ApplicationPresignedS3URL	https://s3.amazonaws.com/appstream-
AppStreamSDKVersion	1.5
DeveloperPassword
InstallerParameters	
KeyPairName	AppStreamKeyPair
LauncherParameters	
LauncherPath	C:\app\XStxDirectXServer\XStxDirectXServer.exe
MicrosoftVirtualAudioDrivers	Y
NvidiaGRIDdrivers	Y
TightVNC	
Create IAM resources	False

Options

Tags

Name	MyAppStreamStack
-------------	------------------

Advanced

Notification	
Timeout	none
Rollback on failure	Yes

[Cancel](#) [Previous](#) [Create](#)

Creating a stack can take 15 to 20 minutes to complete. While waiting for your stack to complete, you can download the Amazon AppStream SDK, which contains the pre-built client applications that you will use to stream your application. To download the Amazon AppStream SDK, go to https://s3.amazonaws.com/appstream-sdk/AppStreamSDK_1.5.0.51.zip. Extract the files to a computer that is accessible to your device or to other computers that will use the pre-built client application.

You can also get the public IP address of your instance. You'll need the public IP address to stream the sample streaming application to your client application.

To get the public IP address of your instance

1. Open the [Amazon EC2 console](#).
2. In the navigation pane, select **Instances**. Select your instance, and click **Connect**.
3. In the **Connect To Your Instance** dialog box, record the public IP address.

After the status of your stack is **CREATE_COMPLETE** and recorded the public IP address, you can stream the sample server application to your device or computer running the pre-built client application.

Step 3: Stream the streaming application to your device

Now that you have your EC2 instance running your application, you are ready to stream your application to a device. Your device needs to connect to the EC2 instance through the pre-built client application from the Amazon AppStream SDK.

To stream using the Android client

Consult your device documentation if you need more information about copying and installing an application on your device.

1. Install a file manager app on your device.
2. Allow your device to install application from unknown sources other than the Google Play store.
3. Copy `<SDK_dir>\precompiled_samples\android\AppStreamExampleClient.apk` and install it on your device.
4. Start the **Amazon AppStream Example Client**.
5. Select **AppStream StandAlone Mode** and type the public IP address of your EC2 instance.
6. Click **Connect**.

After the client application starts, you can press keys to do the following:

Keystroke	Description
A, D, W, S, Q, E	Change the camera angle and/or move of the triangle.
J, K	Change the background color.
I	Add an extra triangle in random locations centered around the world origin.
O	Remove an extra triangle.

To stream using the Mac OS X client

1. Uncompress the `<SDK_dir>\precompiled_samples\osx\AppStreamOSXClientPrecompiled.zip` file.
2. Control-click `<uncompressed_dir>\build\AppStreamSample\AppStream.app` and select **Open** to start the client application.
3. Select **AppStream Standalone Mode** and type the public IP address of your EC2 instance.
4. Click **Connect**.

After the client application starts, you can press keys to do the following:

Keystroke	Description
A, D, W, S, Q, E	Change the camera angle and/or move of the triangle.
J, K	Change the background color.
I	Add an extra triangle in random locations centered around the world origin.
O	Remove an extra triangle.

To stream using the Windows client

1. Install the following dependency files:
 - Download the [DirectX Software Development Kit](#) from the Microsoft Download Center and install it.
 - Download the [Visual C++ Redistributable for Visual Studio 2012](#) from the Microsoft Download Center and install it.
2. Open a command prompt and go the directory where you downloaded the Amazon AppStream SDK.
3. In the <SDK_dir>\precompiled_samples\windows\x64 directory, run the following command:

```
quickRun64.bat <Public IP Address>
```

Where <Public IP Address> is the IP address you copied in the previous step.

Note

If this batch file fails to run, try the batch file in
<SDK_dir>\precompiled_samples\windows\x86 directory.

After the client application starts, you can press keys to do the following:

Keystroke	Description
A, D, W, S, Q, E	Change the camera angle and/or move of the triangle.
J, K	Change the background color.
I	Add an extra triangle in random locations centered around the world origin.
O	Remove an extra triangle.

Option 2: Preview Your Application on Amazon AppStream Standalone Mode

The Amazon AppStream standalone mode is a quick way to see how your application converted to a streaming application works in Amazon AppStream. In this mode, you will use the Amazon AppStream Service simulator to stream your streaming application to a client application.

To complete this section, you will need the following:

- A computer running Microsoft Windows that is connected to the Internet. You will use this computer to administer the Amazon EC2 instance and install your application on the EC2 instance.
- A device or computer that is connected to the Internet. You will stream your application from your Amazon EC2 instance to this device or computer.
 - Android 2.3 (Gingerbread) or later
 - Apple iOS 7.0 or later
 - Mac OS X Mountain Lion (10.8.5) or later
 - Microsoft Windows 7 or later
- The Amazon AppStream SDK, which contains the libraries that you will integrate into your application and the pre-compiled client applications that you will use to connect to your streaming application. You can download the SDK from [Downloads \(p. 10\)](#).
- The source code to your application. You will add streaming to your application by integrating the SDK.

Note

You may incur charges when you use Amazon AppStream standalone mode. Amazon AppStream standalone mode uses your own EC2 instance. Use the [Simple Monthly Calculator](#) to estimate your monthly cost. For more information, see [Amazon EC2 Pricing](#).

In this section, you will do the following:

Topics

- [Step 1: Integrate the SDK \(p. 20\)](#)
- [Step 2: Create a key pair \(p. 20\)](#)
- [Step 3: Create the standalone mode \(p. 21\)](#)
- [Step 4: Copy your application \(p. 25\)](#)
- [Step 5: Stream \(p. 26\)](#)

Step 1: Integrate the Amazon AppStream SDK into Your Application

You begin by adding streaming functionality by integrating the Amazon AppStream SDK into your application. [Build a Streaming Application \(p. 51\)](#) describes how to add streaming to your application. The section includes excerpts from the source code of the sample streaming application included in the SDK in the `<SDK_dir>\example_src` directory. If you want to use the sample streaming application, you can compile the source code into an application. The Amazon AppStream SDK documentation provides the requirements and instructions to compile the source in the `<SDK_dir>\doc\html` directory.

To download the Amazon AppStream SDK

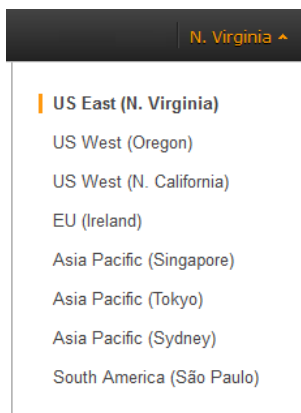
- Go to https://s3.amazonaws.com/appstream-sdk/AppStreamSDK_1.5.0.51.zip. Extract the files to a computer that is accessible to your device or to other computers that will use the pre-built client application.

Step 2: Create a Key Pair

In this step, you will create a key pair in Amazon EC2. The key pair is required to create the Amazon AppStream QuickDeploy standalone mode. If you already have an existing key pair, you can skip this step and proceed to [Step 3: Create the Amazon AppStream standalone mode \(p. 21\)](#).

To create a key pair

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select **US East (N. Virginia)**.



3. In the navigation pane, click **Key Pairs**.
4. Click **Create Key Pair**.
5. For **Key pair name** enter `AppStreamKeyPair` and then click **Create**.
6. The private key file (`AppStreamKeyPair.pem`) is automatically downloaded by your browser. Save the private key file in a safe place.

Important

This is the only chance for you to save the private key file. You will need this file to connect to the instance you will create in the next step.

Your new key pair appears in the Amazon EC2 console. In the next step, you will create a stack in AWS CloudFormation that is the Amazon AppStream QuickDeploy standalone mode.

Step 3: Create the Amazon AppStream standalone mode

Now that you have your key pair, you need to set up the Amazon AppStream standalone through the AWS CloudFormation console. On successful creation, your account will have an running Amazon EC2 instance where you can install your application.

In this step, you will do the following:

- Create the Amazon AppStream standalone mode.
- Get the password to the administrator account which you may need to install your application.
- Get the public IP address of your instance which you will need to stream your application.
- Get the remote desktop file to administer your instance.

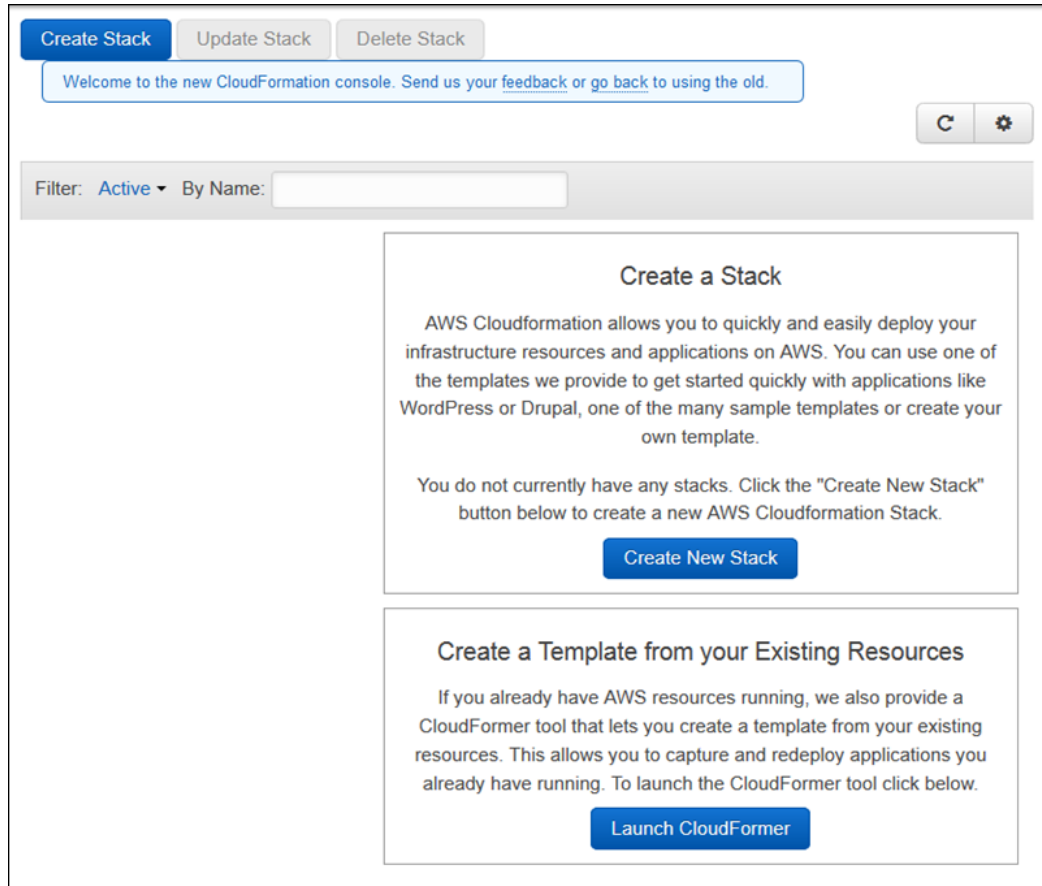
To create the Amazon AppStream standalone mode

1. Open the [AWS CloudFormation console](#).
2. From the navigation bar, select the same region where you created a new key pair or are using an existing key pair.

Important

Do not select a region that is different from your key pair was created.

3. Click **Create Stack** or **Create New Stack**.



4. For **Name**, type **MyStandaloneStack**.
5. Select **Provide an S3 URL** to template and type or paste <https://s3.amazonaws.com/appstream-public/AppStreamDeveloper.template> and then click **Next**.

Select Template

Specify a stack name and then select the template that describes the stack that you want to create.

Stack

An AWS CloudFormation stack is a collection of related resources that you provision and update as a single unit

Name

Template

A template is a JSON-formatted text file that describes your stack's resources and their properties. AWS CloudFormation stores the stack's template in an Amazon S3 bucket. [Learn more.](#)

Source ☒ Select a sample template

☐ Upload a template to Amazon S3

No file chosen

☐ Specify an Amazon S3 template URL

6. In **Specify Parameters**, do the following:
 - For **DeveloperPassword**, type a password that you will use to log into the developer account. The password must be alphanumeric that is between 8-20 characters with at least one number and one letter and does not contain any spaces or special characters.
 - For **KeyPairName**, type **AppStreamKeyPair**, or if you have it, your existing key pair in the same region.
 - For **MicrosoftVirtualAudioDrivers**, type **yes**.
 - For **NvidiaGRIDDrivers**, type **yes**.
 - Click **Next**.
7. In **Options**, do the following:
 - For **Key**, type **Name**.
 - For **Value**, type **MyStandaloneStack**.
 - Click **Next**.
8. In **Review**, check that the parameters you entered are correct and then click **Create**.

Review

Template

Name	MyAppStreamStack
Template URL	https://s3.amazonaws.com/appstream-public/AppStreamDeveloper.template
Description	AppStream Stand-Alone is subject to the AWS Customer Agreement and AppStream Service Terms. AppStream Stand-Alone may only be used for development and testing; you may not host connections from end users on your AppStream Stand-Alone Instance. Do not download, transmit or otherwise distribute AppStream materials provided with the AppStream Stand-Alone instance.
Estimate cost	Link is not available

Parameters

ApplicationPresignedS3URL	https://s3.amazonaws.com/appstream-
AppStreamSDKVersion	1.5
DeveloperPassword
InstallerParameters	
KeyPairName	AppStreamKeyPair
LauncherParameters	
LauncherPath	C:\app\XStxDirectXServer\XStxDirectXServer.exe
MicrosoftVirtualAudioDrivers	Y
NvidiaGRIDdrivers	Y
TightVNC	
Create IAM resources	False

Options

Tags

Name	MyAppStreamStack
------	------------------

Advanced

Notification	
Timeout	none
Rollback on failure	Yes

[Cancel](#) [Previous](#) [Create](#)

Creating a stack can take 15 to 20 minutes to complete. While waiting for your stack to complete, you can do the other tasks.

To get the administrator password, public IP address, and Remote Desktop File of your instance

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, select **Instances**. Select your instance, and click **Connect**.

3. In the **Connect To Your Instance** dialog box, record the public IP address and user name. You'll need the public IP address to stream to your client application. Click **Get Password** (it will take a few minutes after the instance is launched before the password is available).
4. Click **Browse** and navigate to the private key file that you saved when you created the key pair. Select the file and click **Open** to copy the entire contents of the file into contents box.
5. Click **Decrypt Password**. The console displays the default administrator password for the instance in the **Connect To Your Instance** dialog box, replacing the link to **Get Password** shown previously with the actual password.
6. Record the default administrator password. You need this password to connect to the instance and install your application.
7. Click **Download Remote Desktop File**. Your browser prompts you to either open or save the .rdp file. Save the file as you might need this file to administer the instance. When you have finished, you can click **Close** to dismiss the **Connect To Your Instance** dialog box.

You can see how the stack creation is going.

To monitor the progress of stack creation

1. On the AWS CloudFormation console, select the stack **MyStandaloneStack**.
2. In the pane following the list, click the **Events** tab.

The console updates the event list every 60 seconds.

After the status of your stack is **CREATE_COMPLETE**, you can copy our application to the Amazon EC2 instance and configure the Amazon AppStream service simulator.

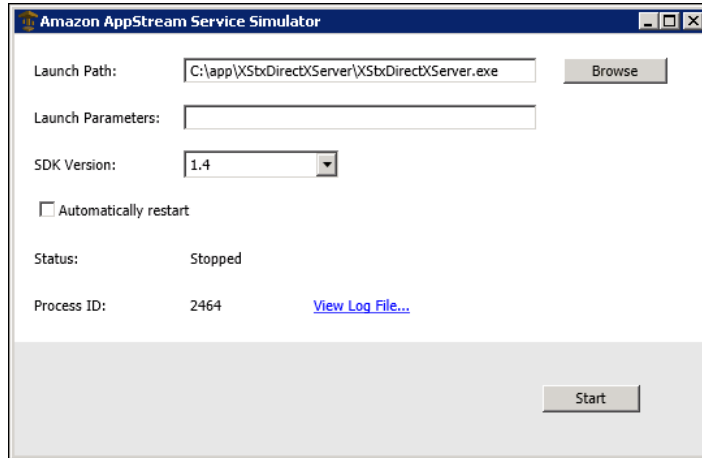
Step 4: Copy Your Application to Your Amazon EC2 Instance

Next you will copy your streaming application and dependency files to your Amazon EC2 instance and then use the Amazon AppStream Service Simulator to start your streaming application. To copy the files to your Amazon EC2, you need remote desktop file and the administrator password to connect to your instance.

You will connect to your Amazon EC2 instance by using Remote Desktop and then copy your streaming application and dependency files to the instance. Then, you configure the Amazon AppStream Service Simulator to start your streaming application.

To connect to your instance and start your streaming application

1. Double-click the Remote Desktop file. You may get a warning that the publisher of the remote connection is unknown. Click **Connect** to connect to your instance. You may get a warning that the security certificate could not be authenticated. Click **Yes** to continue.
2. Log in to the instance as prompted, using **Administrator** as the user name and the default administrator password that you recorded or copied earlier.
3. Copy your streaming application and dependency files to your Amazon EC2 instance.
4. To start the Amazon AppStream Service Simulator, start
`C:\Users\Public\AppData\Local\Temp\AmazonAppStreamSimulator.exe`.
5. In the **Amazon AppStream Service Simulator** dialog box, click **Stop**. Click **Browse** and select your the streaming application file. In **SDK Version**, select the version of the SDK that your streaming application uses. Click **Start**.



Step 5: Stream Your Application to a Device

Now that you have the Amazon AppStream service simulator running your streaming application on your Amazon EC2 instance, you are ready to stream your application to a device. Your device needs to connect to the EC2 instance through the client application from the Amazon AppStream SDK.

To stream using the Android client

Consult your device documentation if you need more information about copying and installing an application on your device.

1. Install a file manager app on your device.
2. Allow your device to install application from unknown sources other than the Google Play store.
3. Copy `<SDK_dir>\precompiled_samples\android\AppStreamExampleClient.apk` and install it on your device.
4. Start the **Amazon AppStream Example Client**.
5. Select **AppStream StandAlone Mode** and type the public IP address of your EC2 instance.
6. Click **Connect**.

Your streaming application appears in the client application. Your EC2 instance can only accept one client connection per session.

To stream using the Mac OS X client

1. Uncompress the `<SDK_dir>\precompiled_samples\osx\AppStreamOSXClientPrecompiled.zip` file.
2. Control-click `<uncompressed_dir>\build\AppStreamSample\AppStream.app` and select **Open** to start the client application.
3. Select **AppStream Standalone Mode** and type the public IP address of your EC2 instance.
4. Click **Connect**.

Your streaming application appears in the client application. Your EC2 instance can only accept one client connection per session.

To stream using the Windows client

1. Install the following dependency files:
 - Download the [DirectX Software Development Kit](#) from the Microsoft Download Center and install it.
 - Download the [Visual C++ Redistributable for Visual Studio 2012](#) from the Microsoft Download Center and install it.
2. Open a command prompt and go the directory where you downloaded the Amazon AppStream SDK.
3. In the <SDK_dir>\precompiled_samples\windows\x64 directory, run the following command:

```
quickRun64.bat <Public IP Address>
```

Where <Public IP Address> is the IP address you copied in the previous step.

Note

If this batch file fails to run, try the batch file in
<SDK_dir>\precompiled_samples\windows\x86 directory.

Your streaming application appears in the client application. Your EC2 instance can only accept one client connection per session.

Option 3: Deploy a Streaming Application on Amazon AppStream



This section shows you how to deploy a streaming application to Amazon AppStream. You will build the sample streaming application from the code in the Amazon AppStream SDK. After building the code, you will create an application installer that installs the streaming application to Amazon AppStream. You will then deploy the application installer to Amazon AppStream and then use one of the client applications to connect to the streaming application.

To complete this section, you will need the following:

- A computer running Microsoft Windows to deploy the streaming application
- A device or computer that is connected to the Internet to use the streaming application. The device or computer must run one of the following operating systems:
 - Android 2.3 (Gingerbread) or later
 - Apple iOS 7.0 or later
 - Mac OS X Mountain Lion (10.8.5) or later
 - Microsoft Windows 7 or later
- An AWS account.
- The Amazon AppStream SDK, which contains the source code that you will compile into a streaming application and the pre-compiled client applications that you will use to connect to your streaming application. You can download the SDK from [Downloads \(p. 10\)](#).
- The following downloads from Microsoft for compiling the sample streaming application. Install the downloads in this order:
 1. [DirectX Software Development Kit](#)
 2. [Microsoft Windows SDK for Windows 7 and .NET Framework 3.5 SP1](#)

3. [Visual C++ Redistributable for Visual Studio 2012](#)

- Visual Studio 2010 or later and the [AWS Toolkit for Visual Studio](#) to create a pre-signed URL to the Amazon S3 bucket required to install your sample streaming application.

Deploying the sample streaming application from the SDK has three steps:

- [Step 1: Build the streaming application](#) (p. 28)
- [Step 2: Deploy the streaming application](#) (p. 28)
- [Step 3: Stream the Streaming Application to a Device](#) (p. 48)

Step 1: Build the streaming application



In the build step, you will compile the code in the `<SDK_dir>\example_src\server\windows\SimpleDirectXServer` directory into a streaming application. The directory contains a Visual Studio solution file, the source code, and the resource files required to build the streaming application. To add streaming to your application, you follow the steps in [Build an Amazon AppStream Application](#) (p. 51).

After you create the streaming application, you then create an application installer that installs your streaming application without any user interaction. The application installer should contain the sample streaming application along with the required dependency files. Use your favorite method to create application installer.

Once you have an application installer, you are ready to deploy your streaming application.

To build the source code

1. Start Visual Studio.
2. Open `<SDK_dir>\example_src\server\windows\SimpleDirectXServer\SimpleDirectXServer.sln`.
3. In **Solution Explorer**, select **SimpleDirectXServer**.
4. In **Solution Configuration** on the toolbar, click **Release**.
5. On the **Build** menu, click **Build Solution**.

The executable file is in the `<SDK_dir>\example_src\server\windows\SimpleDirectXServer\Release` directory.

After you create your executable file, you need to create an application installer. [Build an Application Installer](#) (p. 73) describes the requirements for a silent installer. Use your favorite method to create an application installer that meets the requirements.

After you create your application installer, you are ready to deploy your streaming application to Amazon AppStream.

Step 2: Deploy the streaming application



Deploying your streaming application involves five tasks:

1. Create an Amazon S3 bucket that will store your application installer. Amazon AppStream will get your application installer from this bucket.
2. Create a pre-signed URL to the application installer in the Amazon S3 bucket. Amazon AppStream uses this URL to get your application installer in a secure way.
3. Create a key pair in Amazon EC2 to deploy the sample entitlement service.
4. Deploy the sample entitlement service so that specific users can connect to your streaming application.
5. Deploy your streaming application to Amazon AppStream.

Store Your Installer Application on Amazon S3

After creating your application installer, you will create a bucket in Amazon S3 and then upload the application to this bucket.

To create an Amazon S3 bucket

When Amazon S3 successfully creates your bucket, the console displays the properties of your empty bucket. This is the bucket where you upload the installer application.

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Create a Bucket—Select a Bucket Name and Region** dialog box, type **MyAppStreamBucket**.
3. For **Region**, select **US East (N. Virginia)**.
4. When the settings are as you want them, click **Create**.

To upload the application installer to Amazon S3

1. In the Amazon S3 console, click **MyAppStreamBucket** and then click **Upload**.
2. In the **Upload—Select Files** wizard, click **Add Files**.
3. In the **File Upload** dialog box, select the application installer.
4. Click **Open**.
5. Click **Start Upload**.

Generating a Pre-signed URL

After you have uploaded the application installer to Amazon S3, you need to create a pre-signed URL that points to the application installer in that bucket. A pre-signed URL is a special URL that allows Amazon AppStream to download and run your application installer without requiring to expose any security credentials. This URL is only valid for a specific time period.

To generate a pre-signed URL, you need to create a user in AWS Identity and Access Management. You'll also use this user to deploy the sample entitlement service in a later step.

After the creating the user and credentials, you need to install the AWS Toolkit for Visual Studio and then use the AWS Explorer from within Visual Studio to generate the pre-signed URL.

To create the user

Manage User Permissions Cancel X

Set Permissions

You can customize permissions by editing the following policy document. For more information about the access policy language, see [Overview of Policies](#) in Using IAM. To test the effects of your policies before committing them into production, you can use the [IAM Policy Simulator](#).

Policy Name

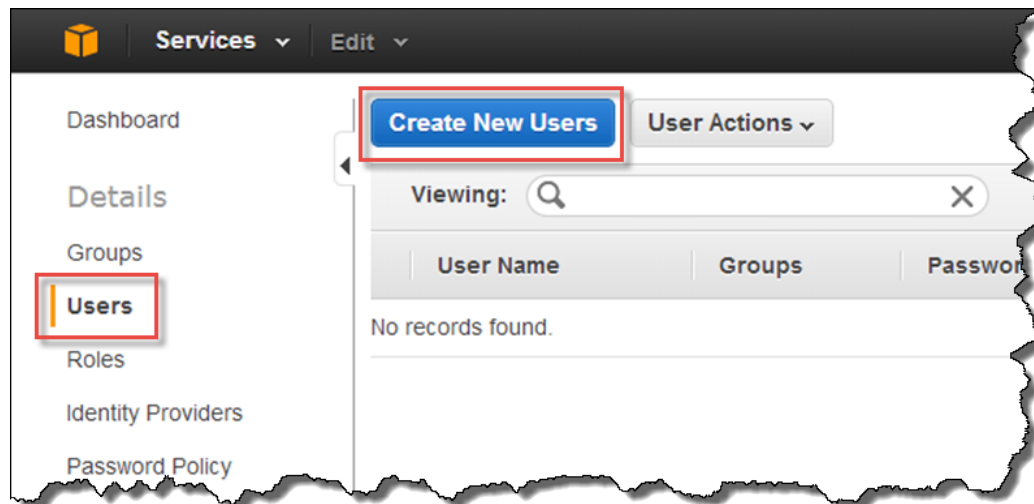
S3andEntitlementServicePolicy

Policy Document

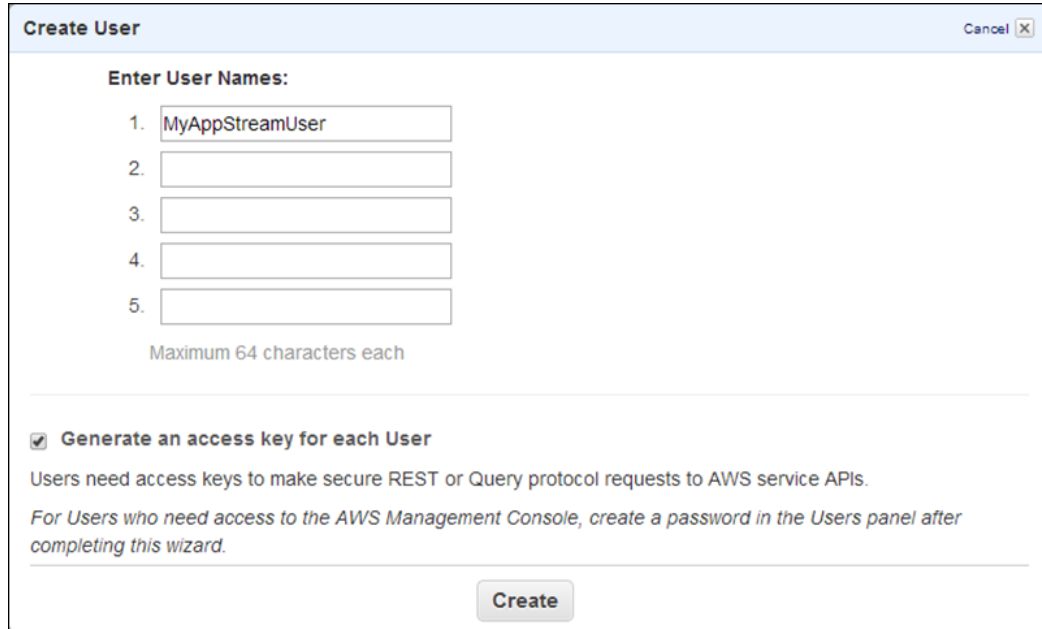
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmnt1403040000000",
      "Effect": "Allow",
      "Action": [
        "appstream:CreateSession",
        "appstream:GetApiRoot",
        "appstream:GetApplication",
        "appstream:..."
      ]
    }
  ]
}
```

[Back](#) Apply Policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the console, click **Users** and then click **Create New Users**.



3. For **Enter User Names**, type **MyAppStreamUser**. Select **Generate an access key for each User** and then click **Create**.

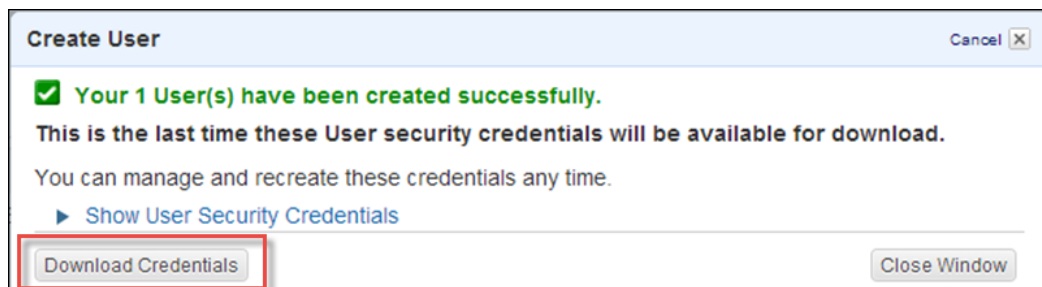


The 'Create User' dialog box has a title bar with 'Create User' and a 'Cancel' button. Below the title bar is a section titled 'Enter User Names:' containing five numbered input fields. The first field contains the text 'MyAppStreamUser'. Below the input fields is the text 'Maximum 64 characters each'. Further down is a checkbox labeled 'Generate an access key for each User' which is checked. Below the checkbox is explanatory text: 'Users need access keys to make secure REST or Query protocol requests to AWS service APIs. For Users who need access to the AWS Management Console, create a password in the Users panel after completing this wizard.' At the bottom right is a 'Create' button.

4. In the **Create User** dialog box, click Download Credentials and save the file to a safe place. The comma separated value file contains the access key and secret key that you will need to generate a pre-signed URL and in a later step, deploy the sample entitlement service.

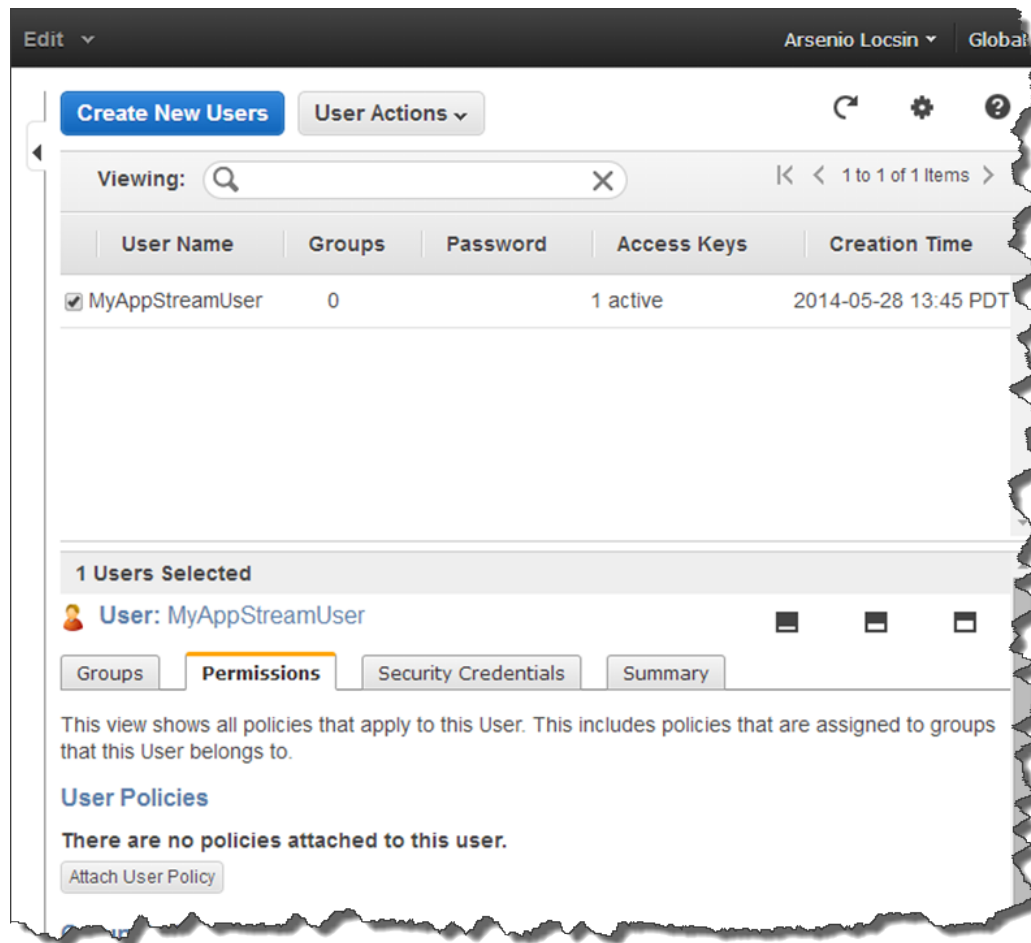
Note

Store this information in a secure place. This is the only time you will be able to get the secret key. If you lose this information, you will need to create a new access key and secret key.

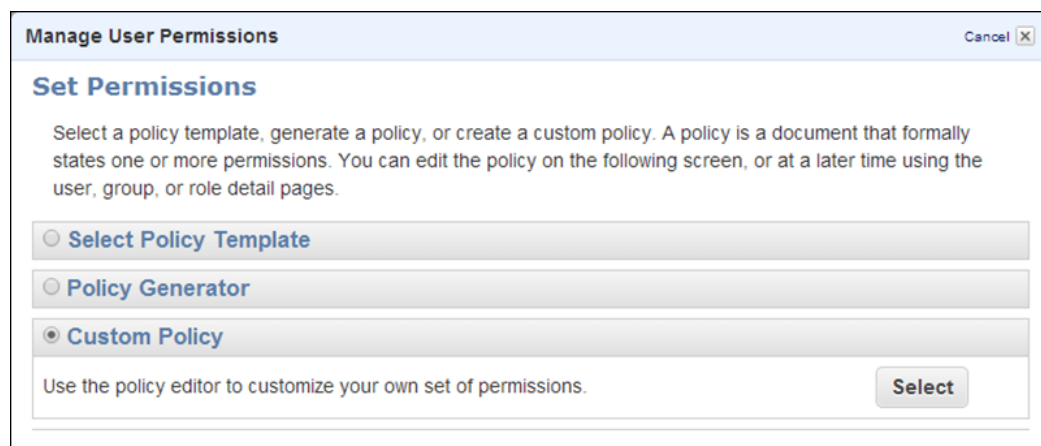


The 'Create User' dialog box shows a success message: 'Your 1 User(s) have been created successfully.' Below this is a warning: 'This is the last time these User security credentials will be available for download.' followed by 'You can manage and recreate these credentials any time.' and a link 'Show User Security Credentials'. At the bottom left is a 'Download Credentials' button, which is highlighted with a red rectangle. At the bottom right is a 'Close Window' button.

5. Click **Close Window** to close the dialog box.
6. In the console, select your new user, click the **Permissions** tab and then click **Attach User Policy**.



7. In **Manage User Permissions**, select **Custom Policy** and then click **Select**.



8. In **Manage User Permissions**, do the following:
- For **Policy Name**, type `s3andEntitlementServicePolicy`.
 - For **Policy Document**, copy and paste the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1403040000000",
      "Effect": "Allow",
      "Action": [
        "appstream:CreateSession",
        "appstream:GetApiRoot",
        "appstream:GetApplication",
        "appstream:GetApplications",
        "appstream:GetApplicationStatus",
        "appstream:GetSession",
        "appstream:GetSessions",
        "appstream:GetSessionStatus",
        "appstream:UpdateSessionState"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Stmt1403040053000",
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Stmt1403040077000",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- c. Click **Apply Policy**.

To install the AWS Toolkit for Visual Studio

1. Go to [AWS Toolkit for Visual Studio](#) and click **AWS Toolkit for Visual Studio**.
2. Run the installation wizard, which is packaged as an `.msi`. Note the following:
 - If your browser asks whether to save or run the `.msi`, select **Run**.
 - If your browser automatically saves the `.msi` file to your system, navigate to the download directory and use Windows Explorer to launch the `.msi`.

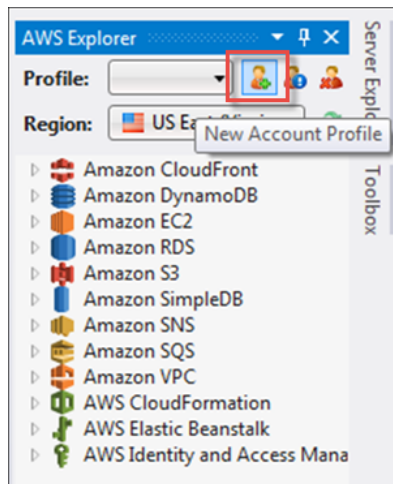
The MSI file name depends on the version, but it will look something like
AWSToolsAndSDKForNet_sdk-2.0.13.2-ps-2.0.13.2-tk-1.6.5.4.msi.

3. Follow the installation wizard's instructions to install the toolkit.

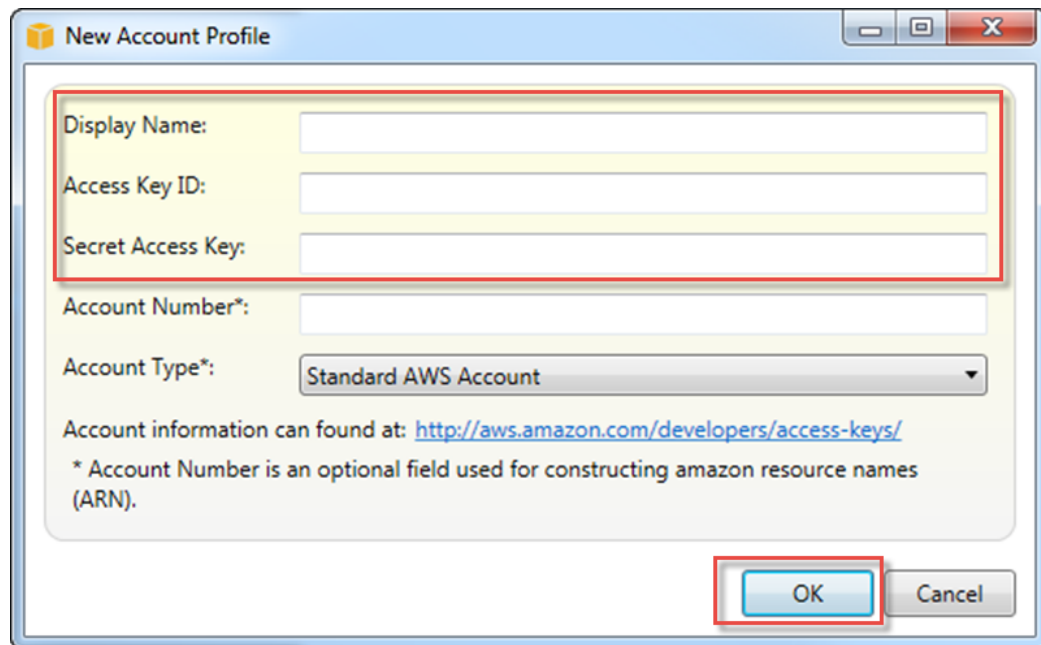
To add a profile to the Toolkit for Visual Studio

Before you can use the Toolkit for Visual Studio, you will need to create a profile using the credentials that you downloaded when you created the user in the previous step. The credentials allow you to access your AWS resources through the Toolkit for Visual Studio.

1. In Visual Studio, open **AWS Explorer** by clicking the View menu and selecting **AWS Explorer**.
2. Click the "New Account Profile" icon to the right of the **Profile** list.



3. In the **New Account Profile** dialog box, do the following:
 - a. For **Display Name**, type **MyAppStreamUser**.
 - b. For **Access Key ID**, enter the access key ID from the credentials of the user you created earlier.
 - c. For **Secret Access Key**, enter the secret key from the credentials of the user you created earlier.
 - d. Click **OK**.



4. Close the **AWS Explorer**.

To generate a pre-signed URL

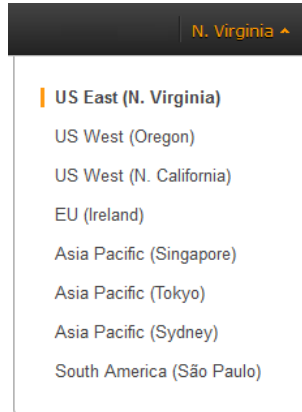
1. Open the **AWS Explorer**.
2. Expand the **Amazon S3** node and click **MyAppStreamBucket**.
3. Right-click the application installer, and then select **Create Pre-Signed URL**.
4. In the **Create Pre-Signed URL** dialog box, set an expiration date and time for the URL. Or go to the next step if you want to use the default setting which is one hour from the current time.
5. Click the **Generate** button.
6. Click **Copy** to copy the pre-signed URL to the clipboard. Save this URL to file. You'll need this URL to add your streaming application to Amazon AppStream.

Creating a Key Pair

Once you have your pre-signed URL, you are ready to create the key pair you need to deploy the sample entitlement service. If you already have an existing key pair, you can skip this step and proceed to [Deploying the Sample Entitlement Service on AWS CloudFormation \(p. 36\)](#).

To create a key pair

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. From the navigation bar, select **US East (N. Virginia)**.



3. In the navigation pane, click **Key Pairs**.
4. Click **Create Key Pair**.
5. For **Key pair name** enter `AppStreamKeyPair` and then click **Create**.
6. The private key file (`AppStreamKeyPair.pem`) is automatically downloaded by your browser. Save the private key file in a safe place.

Important

This is the only chance for you to save the private key file. You will need this file to connect to the instance you will create in the next step.

Your new key pair appears in the Amazon EC2 console.

Deploying the Sample Entitlement Service on AWS CloudFormation

An entitlement service implements code to authenticate users and—if authentication succeeds—calls into Amazon AppStream to create a new client session and provides a connection URL which clients use to connect to the session.

Amazon AppStream does not set requirements on the way you authenticate users. You can compare a username and a hash of the password field to values stored in a database, use an authentication service such as [Login with Amazon \(LWA\)](#), or make an application publically available by automatically authenticating all users. You are in complete control of user authentication and authorization.

To get you started, Amazon AppStream provides:

- **Default entitlement service**—this is an entitlement service provided by Amazon AppStream that looks up user-application mappings in an DynamoDB table and uses that information to authenticate users. It's provided as a AWS CloudFormation template that you can configure and launch to host the entitlement service on your AWS account.
- **Entitlement service sample code**—this is the sample code for the default entitlement service. It looks up user-application mappings in an DynamoDB table and uses that information to authenticate users. By examining the Java code, you can learn how the pieces of the entitlement service fit together and use the code as a basis for writing a custom entitlement service. You can build the sample code into a `.jar` file and run it on a local server or host it on AWS.

Note

While you could implement the entitlement logic directly in the client, doing so is strongly discouraged because of the requirement to call into the Amazon AppStream service to create

new sessions. It is more secure to have your AWS credentials built into a web service running on a server you control than compiled into client code running locally on end-user devices.

To deploy the default entitlement service on AWS CloudFormation

1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation/>.
2. Click **Actions** and then click **Create Stack** or **Create New Stack**.
3. On the **Select Template** page, do the following:
 - a. For **Name** under **Stack**, type **EntitlementService**.
 - b. Select **Specify an Amazon S3 template URL** and then type the pre-signed URL you created in the previous step.
 - c. Click **Next**.

The screenshot shows the 'Select Template' page in the AWS CloudFormation console. At the top, there's a header with 'Edit', 'Arsenio Locsin', 'N. Virginia', and 'Help'. Below this, the 'Select Template' title is followed by a subtitle: 'Specify a stack name and then select the template that describes the stack that you want to create.' The 'Stack' section explains that an AWS CloudFormation stack is a collection of related resources. The 'Name' field is set to 'EntitlementService'. The 'Template' section explains that a template is a JSON-formatted text file. Under 'Source', three options are available: 'Select a sample template', 'Upload a template to Amazon S3', and 'Specify an Amazon S3 template URL'. The third option is selected, and the URL 'https://s3.amazonaws.com/appstream-sdk/a' is entered. At the bottom right, there are 'Cancel' and 'Next' buttons, with 'Next' being highlighted by a red box.

4. In **Specify Parameters**, do the following:
 - a. For **AccessKey**, enter the access key ID of the user you created in the previous step.
 - b. For **CommandStr**, type `java -DUIPin=54321 -DappstreamEndpoint=appstream.us-east-1.amazonaws.com -jar /opt/appstream/entitlement-service.jar`.
 - c. For **FileURL**, type <https://s3.amazonaws.com/appstream-sdk/sample-entitlement-service.jar>.

- d. For **KeyName**, type `AppStreamKeyPair`.
- e. For **SecretKey**, type the secret access key of the user you created in the previous step.
- f. Click **Next**.

Specify Parameters

Specify values or use the default values for the parameters that are associated with your AWS CloudFormation template.

Parameters

AccessKey	<input type="text" value="AKIAIOSFODNN7EXAMPLE"/>	The access key of the user to launch this DES
CommandStr	<input type="text" value="java -DUIPin=54321 -Dappstream"/>	The JAVA launch string which must end in /opt/appstream/entitlement-service.jar, e.g. java -DUIPin=54321 -DappstreamEndpoint=appstream.us-east-1.amazonaws.com -jar /opt/appstream/entitlement-service.jar
FileURL	<input type="text" value="https://s3.amazonaws.com/appsti"/>	URL to your 'entitlement-service.jar', regardless of its original name the jar will be saved as /opt/appstream/entitlement-service.jar.
KeyName	<input type="text" value="AppStreamKeyPair"/>	Name of an existing EC2 KeyPair to enable SSH access to the instance
SecretKey	<input type="text" value="wJairXUtnFEMI/K7MDENG/bPxRfi"/>	The secret key of the user to launch this DES

[Cancel](#) [Previous](#) [Next](#)

5. On the **Options** page, do the following to identify this entitlement service in the Amazon EC2 console:
 - a. Under **Key**, type `Name`.
 - b. Under **Value**, type `EntitlementService`.
 - c. Click **Next**.

Options

Tags

You can specify tags (key-value pairs) for resources in your stack. You can add up to 10 unique key-value pairs for each stack. [Learn more.](#)

	Key (127 characters maximum)	Value (255 characters maximum)	
1	<input type="text" value="Name"/>	<input type="text" value="EntitlementService"/>	<input type="button" value="+"/>

► Advanced

You can set additional options for your stack, like notification options and a stack policy. [Learn more.](#)

On the **Review** page, review the configuration of the stack and then click **Create** to launch the stack.

Review

Template

Name	EntitlementService
Template URL	https://s3.amazonaws.com/appstream-sdk/appstreamEntitlementService.template
Description	AWS CloudFormation Sample Template for AppStream Entitlement Service: Create an Amazon EC2 instance running the Amazon Linux AMI to launch your AppString sample DES. The AMI is chosen based on the region in which the stack is run. This example creates a default security group, so to SSH is open on port 22 open and your web server is available on port 8080. **WARNING** This template an Amazon EC2 instances. You will be billed for the AWS resources used if you create a stack from this template.
Estimate cost	Cost

Parameters

AccessKey	AKIAIOSFODNN7EXAMPLE
CommandStr	java -DUIPin=54321 -DappstreamEndpoint=appstream.us-east-1.amazonaws.com -jar /opt/appstream/entitlement-service.jar
FileURL	https://s3.amazonaws.com/appstream-sdk/sample-entitlement-service.jar .
KeyName	AppStreamKeyPair
SecretKey	wJairXUtnFEM/K7MDENG/bPxRfiCYEXAMPLEKEY
Create IAM resources	False

Options

Tags

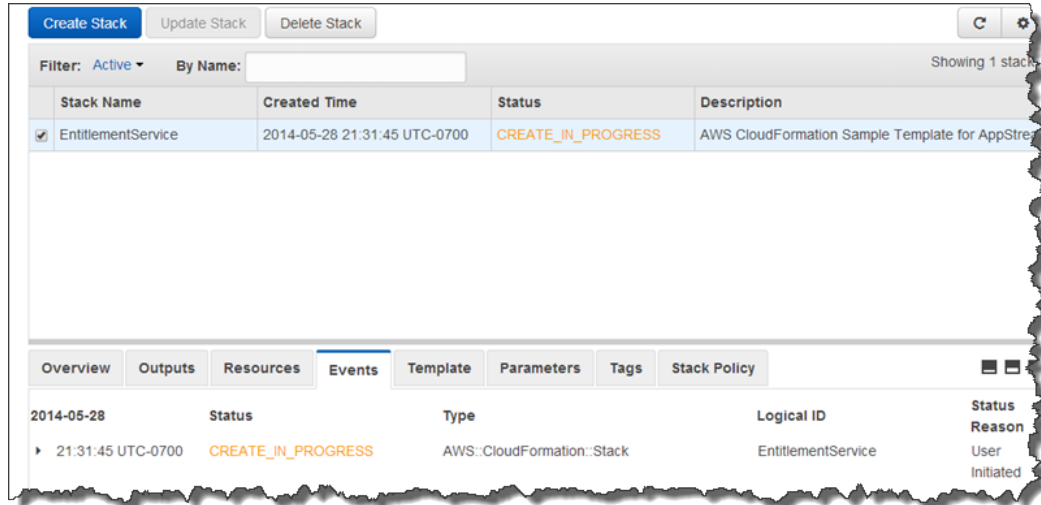
Name	EntitlementService
-------------	--------------------

Advanced

Notification	
Timeout	none
Rollback on failure	Yes

[Cancel](#) [Previous](#) [Create](#)

This process may take several minutes to complete. While the stack is launching, its status is set to **CREATE_IN_PROGRESS**.



When the status of your stack changes to **CREATE_COMPLETE**, your entitlement service is deployed and ready to use. You will need to get the URL of the entitlement service to connect your client application to your streaming application.

To locate the URL of the entitlement service deployed by AWS CloudFormation

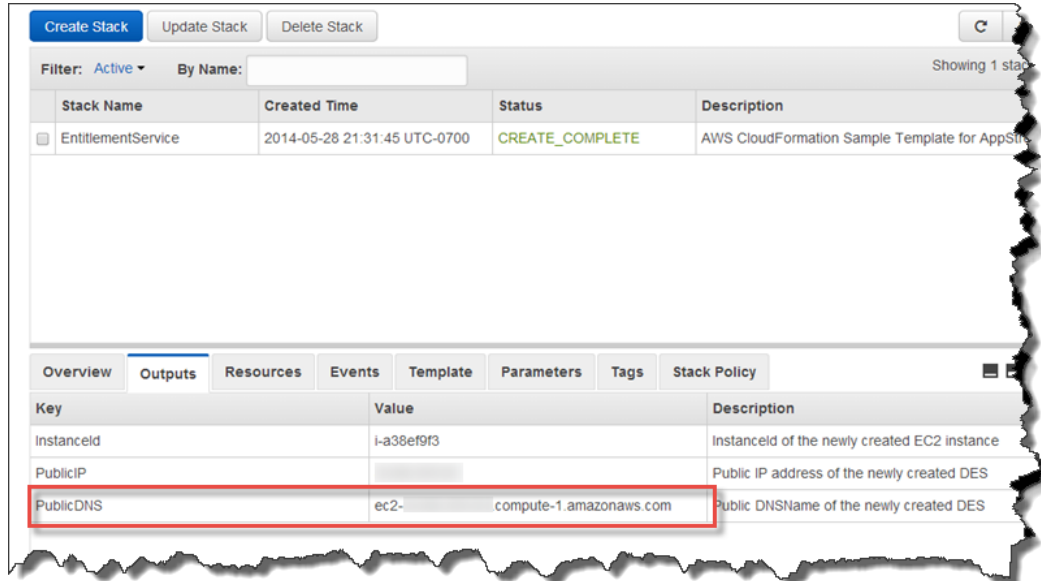
1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation/>.
2. Select your entitlement service. Click the down arrow in the **Events** tab and click **Outputs** to display details about the stack at the bottom of the browser window. The `PublicDNS` key displays the URL of the entitlement service.

The AWS CloudFormation template installs the sample entitlement service on a host that has ports 22 and 8080 open. The entitlement service URL is therefore `http://publicDNS:8080/`. Your clients should send entitlement requests to `http://publicDNS:8080/api/`.

To access the UI for the sample entitlement service, open `http://publicDNS:8080/web/`.

Important

Always include a trailing forward slash ("/") at the end of the URL to access the entitlement service.



To Configure the Entitlement Service

1. In a browser, open <http://publicDNS:8080/web/>.

Note

If you cannot connect to your entitlement service, check that your firewall settings allow you to connect using port 8080.

2. For **PIN**, type 54321 and click **Sign in**.

The screenshot shows a web form with the title 'PIN'. Below the title is a text input field with the placeholder text 'PIN'. Below the input field is a 'Sign in' button.

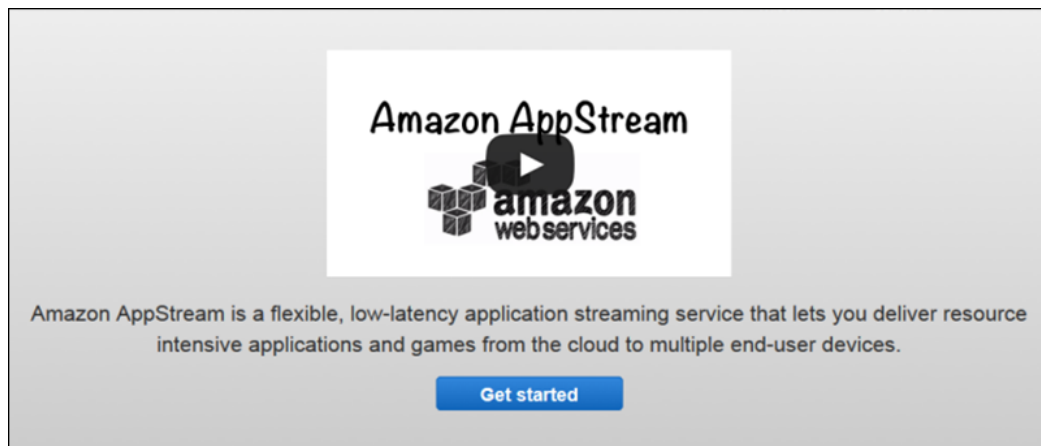
In the web page, you can specify users to have access to all entitlements or you can allow any to access the entitlement. By default, the email address of the AWS account that created the entitlement service and the email address user@domain.com have access to the entitlements.

Deploying Your Streaming Application to Amazon AppStream

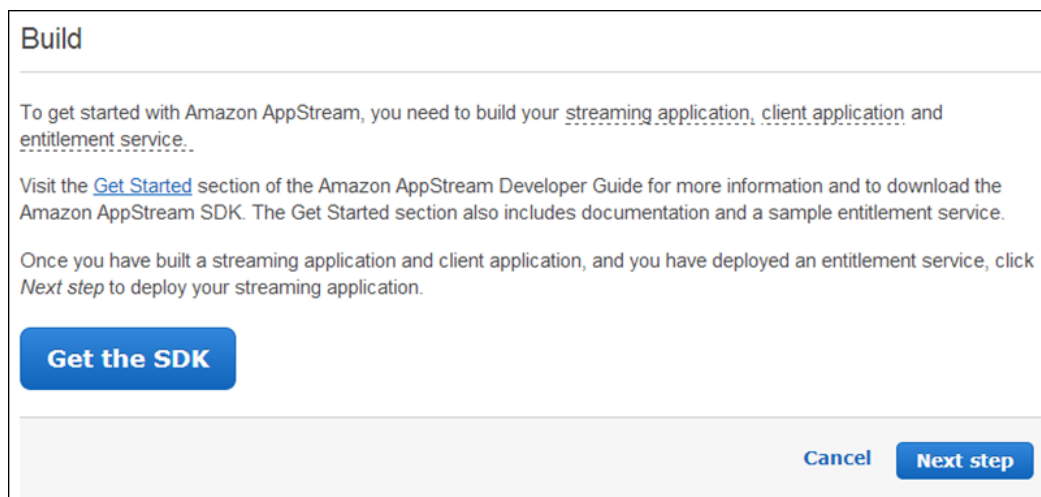
After uploading the application installer to Amazon S3 and then generating a pre-signed URL, you are ready to add your application to Amazon AppStream.

To add your application to Amazon AppStream

1. Open the Amazon AppStream console at <https://console.aws.amazon.com/appstream/>.
2. Click **Get Started**.



3. On the **Build** page, click **Next step**.



4. On the **Configure** page, do the following:
 - For **Streaming application name**, type **My Example Streaming Application**.
 - For **Pre-signed S3 URL of Installer**, type the URL you created in the previous step.
 - For **Path to launcher**, type **C:\MyApp\ExampleDirectXServer**.
 - Click **Next step**.

Configure

To deploy your streaming application, please provide us with the following information:

Streaming application name *	<input type="text" value="My Application"/>	Enter up to 64 letters, numbers, punctuation marks, and spaces to create a friendly name that identifies this streaming application within the console. This is only used in the console - your customers will not see it.
Streaming application description	<input type="text" value="Codenames, special version notes, etc."/>	Enter up to 512 letters, numbers, punctuation marks, and spaces to describe your streaming application or add notes. This is only used in the console - your customers will not see it.
Pre-signed S3 URL to installer *	<input type="text" value="https://mybucket.s3.amazonaws.co"/>	Enter a pre-signed S3 URL to an executable (*.exe) installer file you have stored in S3. The installer file should work without user interaction and contain all of the resources and dependencies for your application bundled together inside of it. Learn how to make a pre-signed S3 URL Learn more about installer requirements
Installer parameters	<input type="text" value="/silent"/>	Enter any command-line parameters required to install your streaming application.
Path to launcher *	<input type="text" value="C:\myapp\launch.exe"/>	Enter an absolute path to the Windows executable (*.exe) file that will launch your streaming application after it is installed, for example, C:\myapp\launch.exe.
Launch parameters	<input type="text" value="/locale en-US"/>	Enter any command-line parameters required to launch your streaming application.

Required *

CancelPreviousNext step

- In the **Log** page, click **Next step**.

Log

You can turn logging on and off for your application to help you diagnose potential problems after deployment.

You will also be charged for S3 usage for any log or crash data stored in your S3 account. [See S3 pricing.](#)

Application Logs

☐ Yes, save application logs
Amazon AppStream collects the following types of logs:

- A file called launch.txt.failed with the output if Amazon AppStream fails to start the streaming application.
- Minidump files (.dmp)
- Standard error stream (launch.txt.stderr)
- Standard output stream (launch.txt.stdout)
- Utilization metrics (utilization.csv) including CPU, memory and disk usage
- You can also enter absolute paths to additional files you want to collect (use * or ? as wild card characters to collect multiple files -- for instance, C:\app\logs*)

Save Location

Select an S3 bucket in which to save logs and crash data. Amazon AppStream can create and use a default bucket or use one that already exists. We configure an IAM role to access the default bucket, and offer details for adding access to an existing bucket.
[View the policy used for the default bucket](#)
[Learn how to configure an existing bucket](#)

Required *

Cancel

Previous

Next step

6. In the **Review** page, review the entries. If you need to change an entry, click **Edit**. If the entries are correct, click **Finish**.

Review

Please take a moment to review the information below and click on any row to edit its contents. When you are ready, click *Finish* to deploy your streaming application. The deployment process can take from 30 minutes to several hours depending on the size of the streaming application.

Check the progress of your deployment at any time by clicking **Applications** in the console and then click **My App**.

Describe Edit these settings	
Streaming application name	My App
Streaming application description	
Setup Edit these settings	
Pre-signed S3 URL to installer	https://s3.amazonaws.com/ap <small>i Be sure the installer file works silently and contains all of the resources and dependencies for your streaming application bundled together inside of it.</small>
Command to install the streaming application	XStxDirectXServerInstaller_1.2.exe <small>i This is the command we will run to install your streaming application.</small>
Command to launch the streaming application	C:\app\XStxDirectXServer\XStxDirectXServer.exe <small>i This is the command we will run to launch your streaming application after it has been installed.</small>
Log Edit these settings	
Application Logs	Not saving application logs


[Cancel](#) [Previous](#) [Finish](#)

- Wait while Amazon AppStream prepares your application. This may take 30 minutes or more while Amazon AppStream performs the following tasks:
 - Copies your application installer from your Amazon S3 bucket.
 - Prepares your Amazon AppStream environment.
 - Installs your streaming application on an Amazon AppStream.
 - Creates an Amazon Machine Image (AMI) of the server configuration that includes your installed application.
 - Starts your streaming application.

While Amazon AppStream is deploying your application, it displays the **Application Summary** page, which contains the metadata for your application. The **Application ID** field displays the identifier assigned to your application. Client applications specify this identifier when they call into your entitlement service to connect to your application.

Streaming Application Summary

[Back to all streaming applications](#)

 **Deploying your application**

Copying your streaming application... done.
Preparing your Amazon AppStream environment... done.
Installing your streaming application...
Creating an Amazon Machine Image (AMI)
Starting your streaming application

Note: This process can take from 30 minutes to several hours to process your streaming application, depending on its size. You can leave this page and the deployment will continue.

Streaming application name: My App

Application ID: [Select](#)

Status: Installing application ...

Utilization: 0 of 0 simultaneous sessions are in use

Streaming application description:

Pre-signed S3 URL of installer: <https://s3.amazonaws.com/ap>

Installer parameters:

Launch path: C:\app\XStxDirectXServer\XStxDirectXServer.exe

Launch parameters:

Application logs: Not saving application logs

Save location: No location selected

Edit

Clone

When your deployment finishes, Amazon AppStream displays a message indicating whether the deployment succeeded or failed.

Streaming Application Summary

[Back to all streaming applications](#)

✓ **Your streaming application has been deployed and is ready for connections!** ✕

To request a session for this streaming application, use the following Application ID in your client:

[Redacted Application ID]

Your account supports streaming up to 10 simultaneous sessions. Limit increase requests take approximately 10 weeks to fulfill, depending on size. You pay only for what you use and there is no minimum fee. When we receive your request, a member of our team will work with you to best serve your capacity needs. [Learn more about service limits](#) or [request a limit increase](#).

Streaming application name: My App

Application ID: [Redacted] [Select](#)

Status: **Running**

Utilization: 0 of 1 simultaneous sessions are in use

Streaming application description:

Pre-signed S3 URL of installer: [https://s3.amazonaws.com/ap-\[Redacted\]](https://s3.amazonaws.com/ap-[Redacted])

Installer parameters:

Launch path: C:\app\XStxDirectXServer\XStxDirectXServer.exe

Launch parameters:

Detected streaming application SDK version: SDK-1.2.0.13

Application logs: Not saving application logs

Save location: No location selected

Edit

Clone

Archive

The wizard will tell you when your client applications can connect to your streaming application. While waiting for your streaming application to launch, copy the **Application ID** to a place you can look up later. You will need this number to connect the client application to the streaming application.

Step 3: Stream the Streaming Application to a Device



In the stream step, you use the precompiled client applications to connect to your streaming application on Amazon AppStream. The SDK includes an Android, iOS, Mac OS X, and Windows sample client application that you can install on a device or computer. Consult your device documentation if you need more information about copying and installing an application on your device.

To stream using the Android sample client application

1. Install a file manager app on your device.
2. Allow your device to install application from unknown sources other than the Google Play store.

3. Copy `<SDK_dir>\precompiled_samples\android\AppStreamExampleClient.apk` and install it on your device.
4. Start the Amazon AppStream Example Client
5. Enter the Application ID of your streaming application and the URL of your entitlement service.
6. Click **Connect**.

To stream using the iOS sample client application

- Compile the source code for the iOS sample client application and then install the application on your device. The code is in the `<SDK_dir>\example_src\client\src\apple\ios` directory. Instruction for compiling the code and running the sample client application are in `<SDK_dir>\doc\html\ios_xcode.html`.

To stream using the Mac OS X sample client application

1. Uncompress the `<SDK_dir>\precompiled_samples\osx\AppStreamOSXClientPrecompiled.zip` file.
2. Control-click `<SDK_dir>\build\AppStreamSample\AppStream.app` and select **Open** to start the client application.
3. Start the Amazon AppStream Example Client
4. Enter the Application ID of your streaming application and the URL of your entitlement service.
5. Click **Connect**.

To stream using the Windows sample client application

1. If you are using a Windows computer other than the computer you used to build and deploy your streaming application, install the following dependency files:
 - a. Download the [DirectX Software Development Kit](#) from the Microsoft Download Center and install it.
 - b. Download the [Visual C++ Redistributable for Visual Studio 2012](#) from the Microsoft Download Center and install it.
2. Open a command prompt and go to the directory where you extracted the Amazon AppStream SDK.
3. In the `<SDK_dir>\precompiled_samples\windows\x64` directory, run the following command:

```
quickRun64.bat
```
4. Enter the Application ID of your streaming application and the URL of your entitlement service.
5. Click **Connect**.

Optional: Clean Up Resources

When you are done using Amazon AppStream, you can use the following procedures to delete the application, service, and bucket you allocated on AWS.

To delete the streaming application from Amazon AppStream

1. Open the Amazon AppStream console at <https://console.aws.amazon.com/appstream/>.
2. In the **AWS Management Console**, click **AppStream** under **Services** to open the **AppStream Applications** console.
3. In **Applications**, click your application to show the properties in **Application Summary**.

4. Click **Delete this application**.
5. In the confirmation message that appears, click **Delete this application**.

To delete your entitlement service from AWS CloudFormation

1. Open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation/>.
2. Select your entitlement service and click **Delete Stack**.
3. In the confirmation message that appears, click **Yes, Delete**.

To delete your files and buckets from Amazon S3

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **All Buckets** pane of the [Amazon S3 console](#), right-click the bucket that you want to delete and then click **Delete**.
3. In the confirmation message that appears, click **OK**.

Where to Go Next

After completing these options, you can start developing your application for Amazon AppStream by going to these sections:

- [Build an Amazon AppStream Application \(p. 51\)](#)—shows how to build and test your application.
- [Build a Client \(p. 80\)](#)—shows how to build your client to connect to the application.
- [Build an Entitlement Service \(p. 73\)](#)—shows how to build an entitlement service that controls who uses the application.

Build an Amazon AppStream Application

An Amazon AppStream application is composed of the following components:

- **Streaming Application**—streams to your client applications by Amazon AppStream.
- **Application Installer**—installs your streaming application on Amazon AppStream
- **Entitlement Service**—authenticates and authorizes your users.
- **Client Application**—connects your users on different devices to your streaming application.

This section shows you how to build these components.

Topics

- [Build a Streaming Application \(p. 51\)](#)
- [Build an Installer \(p. 73\)](#)
- [Build an Entitlement Service \(p. 73\)](#)
- [Build a Client \(p. 80\)](#)

Build a Streaming Application

An application to stream is the heart of your product. In order to be streamed, your application needs to make initialization calls to let Amazon AppStream know that it's ready to accept client sessions, and to have the proper interfaces implemented that Amazon AppStream can call into to connect to client sessions and stream content. You do this using the header and library files provided in the Amazon AppStream SDK. The following sections describe the modifications necessary for streaming and how to add this functionality to an application.

Throughout the discussion, we'll reference code excerpts from a sample application provided in the `<install-dir>\examples_src` directory of the Amazon AppStream SDK. You can download the SDK from the links in [Downloads \(p. 10\)](#).

Topics

- [Design Considerations \(p. 52\)](#)
- [Add Streaming to Your Application \(p. 52\)](#)

- [Test Your Streaming Application \(p. 65\)](#)

Design Considerations for Your Streaming Application

Streaming your application from the cloud offers several advantages over running it natively on consumer devices: you can run a complex application on simple devices, support new consumer devices without updating your application, seamlessly provide new versions of your application to clients, and improve the security of your code.

In exchange for these advantages, however, you have some new requirements to take into consideration when building your application:

- **Continuous network connection**—A application requires a continuous network connection. What user experience will you offer your customers when the network is unavailable? You might show them an error message, or—in the case of a hybrid application—provide them access to the portion of your application that can function offline, running entirely on the consumer device.
- **Managing latency**—Streaming your application from Amazon AppStream adds sources of latency, a small amount of latency from Amazon AppStream overhead as well as variable latency from network conditions. While in many cases, the added latency will be imperceptible to your users, your application needs to be able to tolerate some latency and to handle latency spikes gracefully.
- **Persistent storage**—When you stream your application from Amazon AppStream, it runs on Amazon AppStream hosts in the cloud. When a client session ends, the Amazon AppStream host resources are recovered and any data stored locally on the Amazon AppStream host is lost. If your application needs to persist state between user sessions, you'll need to record that data in a persistent data store (such as [Amazon S3](#), [Amazon RDS](#), or [DynamoDB](#)) before the client session ends and load the data from the persistent store when the next client session begins.
- **Redirecting video and audio output and user input**—In order to stream your application across Amazon AppStream, you'll need to redirect the output to application libraries provided in the Amazon AppStream SDK, and implement interfaces that listen to events from those libraries.
- **Hybrid applications**—Will you stream your entire application from Amazon AppStream or perform some processing on the consumer device? Hybrid applications can offer solutions for both loss of network as well as handling latency spikes. One way to use Amazon AppStream is to run basic application functionality on the consumer device and enhance the experience when a network is available, such as a game with basic character animations rendered on the device with enhanced graphics and detailed backgrounds available through streaming.

Note

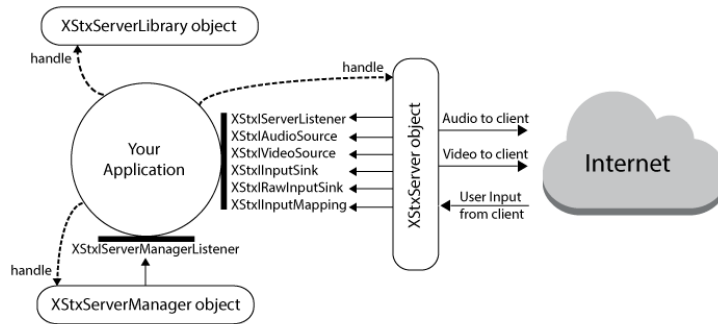
Applications with zero tolerance for latency are not recommended for streaming, such as first-person shooter games or player vs. player fighting games. For these types of applications, consider building a hybrid experience where the latency intolerant aspects of the experience are run locally on the consumer device, while other aspects of the experience requiring cloud resources are streamed.

Add Streaming to Your Application

The Amazon AppStream SDK provides server libraries that you build into your application and call in order to receive client sessions from Amazon AppStream and to stream input and output between your application and clients.

The following image illustrates how your application interacts with the libraries of the Amazon AppStream SDK. Your application has a handle to each of the objects that it can use to call the functions of that

object. It also implements callback functions and registers them with the objects in order to receive client and session events.



The following object classes are provided by the server libraries in the Amazon AppStream SDK. Documentation for these classes are available in the Amazon AppStream SDK.

Class	Description
<code>XStxServerLibrary</code>	The top level object of the server libraries. Your application uses this object to ensure that it is calling into the version of the libraries it was compiled against and to create the <code>XStxServerManager</code> object.
<code>XStxServerManager</code>	This object manages client sessions and sends events to your application when Amazon AppStream assigns your application a client session or terminates a client session. When it fires the <code>XStxIServerManagerListenerFcnServerInitialize</code> event, it returns a handle to the <code>XStxServer</code> object your application uses to stream data to the client.
<code>XStxServer</code>	Your application uses this object to stream content to clients and to receive user input from clients.

Your application implements callback functions that adhere to the following interfaces in order to receive session and client events. Documentation for these interfaces is available in the Amazon AppStream SDK. The callback functions are invoked from different threads, so your application needs to be cautious when modifying data in the callback functions and to use thread synchronization mechanisms.

Interface	Description
<code>XStxIAudioSource</code>	Called by the <code>XStxServer</code> object to get audio frames from your application. If a callback function implementing this interface is not set with the <code>XStxServerSetAudioSource</code> function of the <code>XStxServer</code> object, the <code>XStxServer</code> object captures system audio and transmits that to clients instead.
<code>XStxIInputMapping</code>	Maps user input from a format returned by the client to a format the application can use.
<code>XStxIInputSink</code>	Receives user data from the <code>XStxServer</code> object. This can be keyboard, mouse, or touch data.

Interface	Description
XStxIRawInputSink	Receives raw user data from the XStxServer object. This is sent as a stream of bytes.
XStxIServerListener	receives event messages from the XStxServer object when it establishes or loses connection to a client.
XStxIServerManagerListener	receives event messages from the XStxServerManager object when Amazon AppStream creates or terminates a client session.
XStxIVideoSource	Called by the XStxServer object to configure video streaming from the application. If the video mode is set to XSTX_VIDEO_MODE_PULL, the XStxServer object also uses this interface to get video frames from the application.

Lifecycle of a Streaming Application

Your application communicates with Amazon AppStream and clients through the libraries of the Amazon AppStream SDK.

The lifecycle of an application is as follows:

1. **Create the XStxServerLibrary object.** This ensures that the loaded libraries match the version your application was compiled with. It also gives you a handle to an XStxServerLibrary object, which is required for the next step.
2. **Create the XStxServerManager object.** This object communicates with Amazon AppStream to manage the assignment of client sessions to your application.
3. **Register an XStxIServerManagerListener event sink.** Pass the location of a callback function to receive events from the XStxServerManager object.
4. **Start the XStxServerManager object.** Your application calls XStxServerManagerStart to cause the XStxServerManager object to start accepting client sessions from Amazon AppStream.
5. **Wait for the XStxServerManager object to end the session.** During this phase, your application receives events to its XStxIServerManagerListener callback function when the session manager accepts a new client session, prompts your application to persist state before a session ends, or terminates a client session.

If the event is XStxIServerManagerListenerFcnServerInitialize, a new client session has begun, and your application is passed a handle to the XStxServer object it will use to stream content to the client. In this case, perform the following steps:

- a. **Initialize the XStxServer object.**
- b. **Register event sinks for XStxIAudioSource, XStxIInputMapping, XStxIInputSink, XStxIRawInputSink, XStxIServerListener, and XStxIVideoSource events.** These callback functions are how your application is notified about streaming events.
- c. **Call methods of the XStxServer object to send audio and video frames.** Repeat this step for the duration of the streaming session.
- d. **Terminate the XStxServer object.** When the client is disconnected or the application ends the client session, your application should call XStxServerTerminate to release resources allocated to the XStxServer object.

6. **Recycle the `XStxServerLibrary` and `XStxServerManager` objects.** After the client session ends, release the resources allocated to the `XStxServerLibrary` and `XStxServerManager` objects.

Sample Streaming Application

The file `main.cpp` is the source code for a sample implementation of a streaming application that can be streamed from Amazon AppStream. This sample is in the `<SDK_dir>\example_src\server\windows\SimpleDirectXServer` directory of the Amazon AppStream SDK.

The sample application uses DirectX for video rendering and XAudio2 for audio rendering. Walking through the sample code is useful in understanding how to construct an application.

Initialize a Streaming Application

Before your application can start streaming content to a client, it must create the objects that it will use to connect to client sessions and stream content as well as register callback functions to receive event notifications about sessions and clients. The following explanation covers steps 1–5 of [Lifecycle of a Streaming Application \(p. 54\)](#).

First, your application should create an `XStxServerLibrary` object. This is the top level object that you use to interact with the libraries you'll use to connect to sessions and stream content to clients.

The following excerpt from the sample streaming application illustrates this step. The excerpt is from the `runAsAppStreamGame` function of the file `XStxExampleServer.cpp`. The file is in the `<SDK_dir>\example_src\server\common` directory.

```
int runAsAppStreamGame(int argc, const char* argv[])
{
    ...

    /** Initialize the XStxServer library */

    XStxResult result = XStxServerLibraryCreate(
        XSTX_SERVER_API_VERSION_MAJOR,
        XSTX_SERVER_API_VERSION_MINOR,
        &serverLibraryHandle);

    if (result != XSTX_RESULT_OK)
    {
        goto exit;
    }

    ...
}
```

Next, your application creates an `XStxServerManager` object. To do so, you'll pass in the handle to the `XStxServerLibrary` object you created previously. Your application uses the `XStxServerManager` object to receive session assignments from Amazon AppStream.

The following excerpt from the sample streaming application illustrates this step. The excerpt is from the `runAsAppStreamGame` function of the file `XStxExampleServer.cpp`. The file is in the `<SDK_dir>\example_src\server\common` directory.

```
int runAsAppStreamGame(int argc, const char* argv[])
{
    ...

    /**
     * Create a session manager instance (which in this case just fakes
     * receiving session requests from external services).
     */

    XStxResult result = XStxServerLibraryCreateXStxServerManager(
        serverLibraryHandle,
        &serverManagerHandle);

    if (result != XSTX_RESULT_OK)
    {
        goto exit;
    }

    ...
}
```

After the `XStxServerManager` object is created, your application calls `XStxServerManagerSetListener` to register an event sink.

The following excerpt from the sample streaming application illustrates this step. The excerpt is from the `runAsAppStreamGame` function of the file `XStxExampleServer.cpp`. The file is in the `<SDK_dir>\example_src\server\common` directory.

The code that implements the `XStxIServerManagerListener` interface and handles the events sent by the `XStxServerManager` object is located in the file `ServerManagerListener.cpp`. The file is in the `<SDK_dir>\example_src\server\common` directory.

```
int runAsAppStreamGame(int argc, const char* argv[])
{
    ...

    /**
     * Create the application-specific server manager listener
     * that will start our hosted application and connect it to
     * an XStx server.
     */

    XStxResult result = ServerManagerListener::createServerManagerListener(
        serverLibraryHandle,
        serverManagerListener,
        (void*)&theGame);

    if (result != XSTX_RESULT_OK)
    {
        goto exit;
    }

    /* Point the session manager at our listener */

    XStxResult result = XStxServerManagerSetListener(
        serverManagerHandle,
```

```
serverManagerListener->getServerManagerListener());  
  
if (result != XSTX_RESULT_OK)  
{  
    delete serverManagerListener;  
    goto exit;  
}  
  
...  
}
```

After you've registered the event sink, your application is ready to start the server manager to notify Amazon AppStream that it's ready to receive client sessions. To do so, call the `XStxServerManagerStart` function.

The following excerpt from the sample streaming application illustrates this step. The excerpt is from the `runAsAppStreamGame` function of the file `XStxExampleServer.cpp`. The file is in the `<SDK_dir>\example_src\server\common` directory.

```
int runAsAppStreamGame(int argc, const char* argv[])  
{  
    ...  
  
    /** Start the session manager */  
  
    XStxResult result = XStxServerManagerStart(serverManagerHandle);  
  
    if (result != XSTX_RESULT_OK)  
    {  
        goto exit;  
    }  
  
    ...  
}
```

At this point, your application's main function should wait until the `XStxServerManager` terminates the session. This is done by calling `XStxServerManagerWait`.

The following excerpt from the sample streaming application illustrates this step. The excerpt is from the `runAsAppStreamGame` function of the file `XStxExampleServer.cpp`. The file is in the `<SDK_dir>\example_src\server\common` directory.

```
int runAsAppStreamGame(int argc, const char* argv[])  
{  
    ...  
  
    /** Wait for the session manager to exit */  
  
    XStxResult result = XStxServerManagerWait(serverManagerHandle);  
  
    ...  
}
```

While your application is waiting for the session to end, your `XStxIServerManagerListener` event sink is receiving session events. When the `XStxIServerManagerListenerFcnServerInitialize` event fires, it indicates that Amazon AppStream has assigned your application a client. For more information, see [Initialize a Client Session \(p. 58\)](#)

Initialize a Client Session

When the `XStxIServerManagerListenerFcnServerInitialize` event fires it returns a handle to an `XStxServer` object. This is the object that your application uses to communicate with the client: streaming audio and video output and receiving user input and client messages. Your implementation of a `XStxIServerManagerListenerFcnServerInitialize` callback function should initialize the `XStxServer` object, register event sinks for the `IServerListener` events, and set the audio source by calling the `XStxServerSetVideoSource` function. Setting the audio source by calling the `XStxServerSetAudioSource` function is optional; for more information, see [Stream Audio to a Client \(p. 62\)](#).

This excerpt from `ServerManagerListener.cpp` shows how to initialize the `XStxServer` object. The file is in the `<SDK_dir>\example_src\server\common` directory.

```
XStxResult ServerManagerListenerImp::XStxIServerManagerListenerServerInitialize(  
  
    XStxServerHandle server,  
    uint32_t timeout,  
    const char* applicationContext)  
{  
  
    if (mServerToInfoMap.find(server) != mServerToInfoMap.end())  
    {  
        return XSTX_RESULT_ALREADY_CREATED;  
    }  
  
    XStxResult result = XSTX_RESULT_OK;  
  
    /** Create an object to hold on to server specific info */  
  
    ServerInfo* info = new ServerInfo();  
    XStxIServerListener* listener = NULL;  
    info->mApp = NULL;  
    info->mServer = server;  
  
    /** Instantiate the hosted application */  
  
    result = HostedApplication::createHostedApplication(  
        server,  
        applicationContext,  
        mServerContext,  
        info->mApp);  
  
    if (result != XSTX_RESULT_OK)  
    {  
        goto exit;  
    }  
  
    /** Point the app to the server and start the app */  
  
    result = info->mApp->setServer(info->mServer);
```



```
if (result != XSTX_RESULT_OK)
{
    goto exit;
}

result = info->mApp->start();

if (result != XSTX_RESULT_OK)
{
    goto exit;
}

/** Point the server to the app and start the server */

listener = info->mApp->getServerListener();
if ( NULL != listener )
{
    XSTX_CALLBACK_NOT_NULL_OR_ERROR(listener, ServerReady);
    XSTX_CALLBACK_NOT_NULL_OR_ERROR(listener, ServerStopped);
    XSTX_CALLBACK_NOT_NULL_OR_ERROR(listener, MessageReceived);
}
result = XStxServerSetListener(
    info->mServer,
    listener);

if (result != XSTX_RESULT_OK)
{
    goto exit;
}

result = XStxServerSetInputSink(
    info->mServer,
    info->mApp->getInputSink());

if (result != XSTX_RESULT_OK)
{
    goto exit;
}

#ifdef APPLICATION_CAPTURES_AUDIO
//The example audio source does provides timestamps
result = XStxServerSetAudioSource(
    info->mServer,
    info->mApp->getAudioSource(), true);

if (result != XSTX_RESULT_OK)
{
    goto exit;
}
#endif
//need to manually change the isProvidingTimestamp flag here to false
//if it doesn't provide timestamp
result = XStxServerSetVideoSource(
    info->mServer,
    info->mApp->getVideoSource(), true);

if (result != XSTX_RESULT_OK)
{
    goto exit;
}
```

```
    }

    mServerToInfoMap[server] = info;

    return XSTX_RESULT_OK;

exit:

    if (info != NULL)
    {
        delete info->mApp;
        XStxServerRecycle(info->mServer);
        delete info;
    }

    return result;
}
```

Stream Video to a Client

In order to stream video frames to the client, your application calls the `XStxServerSetVideoSource` function of the `XStxServer` object to specify the source of video frames. For more information, see [Initialize a Client Session \(p. 58\)](#). You must also define the mode that your application will use to stream frames to the client.

Choose a Video Mode

To support a variety of applications, Amazon AppStream provides three strategies for streaming video:

- **Push with frame-rate blocking**—Calls to `XStxServerPushVideoFrame()` will block until enough time has elapsed to gate the video frame rate of the application to that specified by `XStxIVideoSourceFcnSetFrameRate()`. The amount of time blocked varies dynamically with changes to the target frame rate and the time it takes to generate frames. To use this strategy, set the video mode to `XSTX_VIDEO_MODE_PUSH_BLOCKING`.
- **Push immediately**—Calls to `XStxServerPushVideoFrame()` will return immediately. If calls are received faster than the frame rate indicated by the last call to `XStxIVideoSourceFcnSetFrameRate()` then frames will be dropped from the video stream to achieve the target frame rate. To use this strategy, set the video mode to `XSTX_VIDEO_MODE_PUSH_IMMEDIATE`.
- **Pull**—Video frames will be pulled from the application by calls to `XStxIVideoSourceFcnGetFrame()`. The thread making those calls will delay as necessary to limit the frame rate. The application is free to perform operations on this thread as long as it returns before the next frame is due. To use this strategy, set the video mode to `XSTX_VIDEO_MODE_PULL`.

The sample application uses the immediate push mode, as shown in the following excerpt from `main.cpp`. This file is in the `<SDK_dir>\example_src\server\windows\SimpleDirectXServer` directory.

```
static XStxResult getVideoMode(void* context, XStxVideoMode* mode)
{
    // We push frames to AppStream when we want to.
    // AppStream will adapt by dropping frames if we're too fast
    *mode = XSTX_VIDEO_MODE_PUSH_IMMEDIATE;
    return XSTX_RESULT_OK;
}
```

Choose a Color Subsampling Rate

Amazon AppStream streams the video at the YUV420 color subsampling rate to client applications. You can set your streaming application to stream video at the YUV444 color subsampling rate to a client application running on supported devices. The streaming application calls the `XStxServerAddChromaSamplingOption()` function to notify Amazon AppStream that the streaming application supports the YUV444 color subsampling option. The client application calls a similar function to also notify Amazon AppStream that the client application supports the YUV444 color subsampling option.

When the client application connects to the streaming application, Amazon AppStream compares the color subsampling options available on the client application and the streaming application. Amazon AppStream then selects the highest color resolution supported by both applications and then informs both applications which color sampling option will be used for the session. Amazon AppStream then calls the `XStxIServerListener2FcnServerConfigurationSettingsReceived` callback function that the streaming application supplied and passes a structure with the *XStxChromaSampling* setting.

Note

Streaming at the YUV444 subsampling rate requires higher bandwidth availability than the YUV420 rate.

The following code excerpt demonstrates how to use the YUV444 color subsampling option.

```
/**
 * Inform the server of server application's chroma sampling capability.
 * You can call this as many times as you want for each chroma sampling
 * listed in XStxChromaSampling. If this method is never called, then
 * XSTX_CHROMA_SAMPLING_YUV420 will be used by default chroma sampling.
 * Register a callback function at XStxIVideoSourceFcnSetChromaSampling
 * so that STX server can notify server application which chroma sampling
 * will be used for streaming.
 * @param[in] serverHandle The handle of the server.
 * @param[in] chromaSampling The chroma sampling scheme. Look at XStxAPI.h
 * @return This function will return one of these values.
 * Return code | Description
 * ----- | -----
 *
 * XSTX_RESULT_OK | The operation is successful.
 * XSTX_RESULT_INVALID_HANDLE | serverHandle is invalid.
 * XSTX_RESULT_INVALID_ARGUMENTS | chromaSampling is not recognized.
 */
XSTX_API_EXTERN XStxResult XSTX_API XStxServerAddChromaSamplingOption(
    XStxServerHandle serverHandle,
    XStxChromaSampling chromaSampling);
```

Send Frames to the Client

If you are converting an existing application to work with Amazon AppStream streaming, you will need to redirect the video output to streaming clients instead of displaying video on the local machine. The following excerpt is from the `render` function in `main.cpp` and shows the changes you would make to render the video to clients instead of a local machine.

```
...

EnterCriticalSection(&g_frameCriticalSection); // Don't want to be interrupted
now
```

```
// Copy back buffer data. Can also use D3DXLoadSurfaceFromSurface if we need
to resize/change pixel format
g_D3DDevice->GetRenderTargetData(g_backBuffer, g_memBuffer);
D3DLOCKED_RECT lockedRect;
g_memBuffer->LockRect(&lockedRect, NULL, D3DLOCK_READONLY);
// Convert to YUV so we can supply it to AppStream
switch (g_chromaSamplingType)
{
    case XSTX_CHROMA_SAMPLING_YUV420:
        convertToYUV420((unsigned char*)lockedRect.pBits, WINDOW_WIDTH, WINDOW_HEIGHT, 2, 1, 0, 4, lockedRect.Pitch, g_videoFrame.mPlanes);
        break;
    case XSTX_CHROMA_SAMPLING_YUV444:
        convertToYUV444((unsigned char*)lockedRect.pBits, WINDOW_WIDTH, WINDOW_HEIGHT, 2, 1, 0, 4, lockedRect.Pitch, g_videoFrame.mPlanes);
        break;
    default:
        assert(!"Unknown chroma sampling type"); // Make sure we don't get an unknown chroma sampling type
}
g_memBuffer->UnlockRect();

XStxServerPushVideoFrame(g_serverHandle, &g_videoFrame); // Push the video frame
LeaveCriticalSection(&g_frameCriticalSection);

...
```

Stream Audio to a Client

There are two ways your application can transmit audio to clients:

- **Explicitly send audio frames to the client**—Explicitly sending audio frame-by-frame is best for situations in which you need tight integration between the video and audio frames, for example, if you are streaming speech synchronized with video of a person talking. To do so, call the `XStxServerSetAudioSource` function of the `XStxServer` object and set an audio source, you can then explicitly stream audio frames to the client by implementing the `XStxIAudioSourceFcnGetFrame` function of the `XStxIAudioSource` interface.
- **Automatically capture system audio and send that to the client**—This is the easiest way to stream audio and works best in cases where you do not need tight integration between audio and video, for example if you are streaming background music during a puzzle game. To automatically stream audio, do not set an audio source by calling the `XStxServerSetAudioSource` function. When no audio source is set, the `XStxServer` object automatically captures system audio and streams that to the client.

Audio Timestamps

Timestamps are important for keeping the audio and video frames in synchronization. If your application uses automatic streaming of the system audio, the `XStxServer` object timestamps the audio frames using a built-in timestamp manager. You can also have the `XStxServer` object automatically timestamp your video frames by setting the `willProvideTimestamps` parameter to `false` when you call the `XStxServerSetVideoSource` function to set the video source of your application.

If you want to set the timestamps explicitly for your video frames and your application is using the automatic streaming of system audio, you can do so by having your application call the

`XStxServerGetTimestampUs` function. This function returns the current mono timestamp from the `XStxServer` object, in microseconds.

Receive Content from a Client

To provide a fully interactive experience for users, the client collects data from user input sources (such as keyboard, mouse, or touch inputs) and sends that data to the application so the application can respond to the user action. There are three types of content that a client may send to your application:

- **Formatted user input**—such as keyboard, mouse or touch input.
- **Raw user input**—a stream of bytes from the device. This enables your application to support new or device-specific types of input such as a data stream from an accelerometer.
- **Client messages**—messages from the client to the application that are independent of user actions. These can be status messages, additional metadata, or other content as negotiated between the client and application developers.

Your application receives this content from the client as events sent by the `XStxServer` object to your callback functions. The following sections describe how to implement the callback functions to handle these events.

Topics

- [Accept Keyboard, Mouse, and Touch User Input \(p. 63\)](#)
- [Receive Raw User Input from a Client \(p. 63\)](#)
- [Receive Client Messages \(p. 63\)](#)

Accept Keyboard, Mouse, and Touch User Input

When a client sends keyboard, mouse, or touch user input to your application, the `XStxServer` object fires an `XStxIInputSinkFcnOnInput` event. Your callback function to handle this event uses the `mType` member of the `XStxInputEvent` structure it receives to determine the input source (keyboard, mouse, or touch) and handle it appropriately.

Receive Raw User Input from a Client

When a client sends keyboard, mouse, or touch user input to your application, the `XStxServer` object fires an `XStxIRawInputSinkFcnOnRawInput` event. The data is transmitted as a raw stream of bytes. Your callback function to handle this event is responsible for interpreting the byte stream data. This requires close integration between your application and its client applications.

Receive Client Messages

The client may send messages to your application. These are not related to user input; the content and purpose of these messages is specific to the client. During application development, you should research the messages that may be sent by the client and implement code to handle those messages. Sending messages from the client is optional, and some clients may send no messages at all.

To receive and handle client messages, your application implements a callback function to handle `XStxIServerListenerMessageReceived` events. When a `XStxIServerListenerMessageReceived` event fires, it passes the message to your callback function as a byte array.

Store Persistent Data

When you host your application on Amazon AppStream, your application runs on an Amazon AppStream host in the cloud. When the session ends, the Amazon AppStream host is recycled, and any data stored on the Amazon AppStream host is lost. If your application needs to persist data between sessions, it

should record the data in a persistent data store before terminating the session. You can record the data to a physical server or store the data on AWS using a service such as [Amazon S3](#), [Amazon RDS](#), or [DynamoDB](#).

The `XStxServerManager` object fires an `XStxIServerManagerListenerFcnServerSaveState` event before terminating a session to give your application a chance to record data before the session ends. To do this, provide an implementation of `XStxIServerManagerListenerServerSaveState` that records data to a persistent data store.

The following excerpt from the sample streaming application shows an implementation of `XStxIServerManagerListenerServerSaveState`. It is from `ServerManagerListener.cpp`. This file is in the `<SDK_dir>\example_src\server\common` directory. This implementation does not persist data, but the comment shows where you would add that functionality.

```
XStxResult ServerManagerListenerImp::XStxIServerManagerListenerServerSaveState(
XStxServerHandle session,
uint32_t timeout,
XStxStopReason reason)
{
    // Add code here to record data to a persistent data store

    return XSTX_RESULT_OK;
}
```

Terminate a Client Session

When a client session ends, the `XStxServerManager` object fires an `XStxIServerManagerListenerFcnServerTerminate` event. This gives your application a chance to gracefully release resources associated with the client session.

The following excerpt from the sample streaming application illustrates this step. It is from `ServerManagerListener.cpp`. This file is in the `<SDK_dir>\example_src\server\common` directory.

```
XStxResult ServerManagerListenerImp::XStxIServerManagerListenerServerTerminate(
XStxServerHandle session,
uint32_t timeout,
XStxStopReason reason)
{
    ...

    if (info != NULL)
    {
        XStxServerRecycle(info->mServer);
        delete info->mApp;
        delete info;
    }
    return XSTX_RESULT_OK;
}
```

Terminate a Streaming Application

The `XStxServerManagerWait` function returns when a client session ends and Amazon AppStream terminates the session. Your application should release the resources allocated to the `XStxServerManager` and `XStxServerLibrary` objects.

The following excerpt from the sample streaming application illustrates this step. It is from the `runAsAppStreamGame` function of `XStxExampleServer.cpp`. This file is in the `<install_dir>\example_src\server\common` directory.

```
exit:

    XStxResult cleanUpResult = XStxServerManagerRecycle(serverManagerHandle);
    if (cleanUpResult != XSTX_RESULT_OK) {
        printf("Failed to recycle ServerManager\n");
    }

    cleanUpResult = XStxServerLibraryRecycle(serverLibraryHandle);
    if (cleanUpResult != XSTX_RESULT_OK) {
        printf("Failed to recycle ServerLibrary\n");
    }

    delete serverManagerListener;

    printf("Exit! %s\n", XStxResultGetName(result));
    fflush(stdout);

    return (result != XSTX_RESULT_OK) ? -1 : 0;
```

Test Your Streaming Application

Although you plan to stream the production version of your streaming application from the cloud using Amazon AppStream, you will likely want to run your streaming application locally or by using Amazon AppStream standalone mode during testing. This lets you test changes you make to the code quickly, without having to build an installer, upload your streaming application to Amazon S3, generate a pre-signed URL, and the other steps required to deploy an application on Amazon AppStream.

Developing a streaming application typically has the following stages:

1. **Test your application using Amazon AppStream standalone mode**—in this stage, the core functionality of your application is done, and you're adding streaming. By testing streaming on a standalone Amazon EC2 instance that is not managed by Amazon AppStream, you can connect to the Amazon AppStream host using a remote management service to modify your application code, add dependency files, or change the server configuration. For more information, see [Stream Your Application Using Amazon AppStream Standalone Mode](#) (p. 66).
2. **Deploy your production-ready application on Amazon AppStream**—in this stage, development and testing of your application is done, and you are ready to deploy it on Amazon AppStream and have the service manage client sessions. For more information, see [Deploy Your Streaming Application to Amazon AppStream](#) (p. 123).

Stream Your Application Using Amazon AppStream Standalone Mode

Testing your application using Amazon AppStream standalone mode provides a way to test that streaming works without having the overhead of a full deployment to Amazon AppStream. It also gives you a way to test your application without having to write an entitlement service or a custom client by using a AWS CloudFormation template to create your own Amazon EC2 instance with the Amazon AppStream SDK libraries installed.

You may incur charges when you use Amazon AppStream standalone mode. In standalone mode, you are using your own EC2 instance rather than Amazon AppStream. Using standalone mode does not apply to the first 20 hours of streaming from Amazon AppStream.

You can use the Windows sample client application to connect directly to the IP address of the Amazon EC2 instance. For more information, see [Connect to Your Application with the Sample Windows Client](#) (p. 70).

Note

Do not use a standalone streaming server to deploy your streaming application for client access. Doing so will prevent you from taking advantage of the session management and automatic scaling provided by Amazon AppStream.

Topics

- [Use Amazon AppStream Standalone Mode](#) (p. 66)
- [Connect to Your Standalone Host](#) (p. 70)
- [Connect to Your Application with the Sample Windows Client](#) (p. 70)
- [Clean Up Resources](#) (p. 71)

Use Amazon AppStream Standalone Mode

You can use Amazon AppStream standalone mode to test your streaming application. Amazon AppStream standalone mode configures an Amazon EC2 instance with the Amazon AppStream SDK libraries installed; is configured for SSH, RDP, or VNC access; and has an elastic IP address assigned to it. By using Amazon AppStream standalone mode, all you need do is connect to your Amazon EC2 instance and upload your application to the server and run the command to launch it.

Note

You may incur charges when you use this method to deploy a GPU EC2 instance. Use the [Simple Monthly Calculator](#) to estimate your monthly cost. For more information, see [Amazon EC2 Pricing](#).

To use the AWS CloudFormation template described below, you must have:

- An active AWS account. You will need the access keys for the account or for an IAM user in the account. If you are using the access keys of an IAM user, that user must have permissions to perform Amazon AppStream actions. For more information, see [AWS Security Credentials](#).
- An EC2 key pair. You can use this to connect to the instance that hosts the entitlement service with SSH. For more information, see [Amazon EC2 Key Pairs](#).

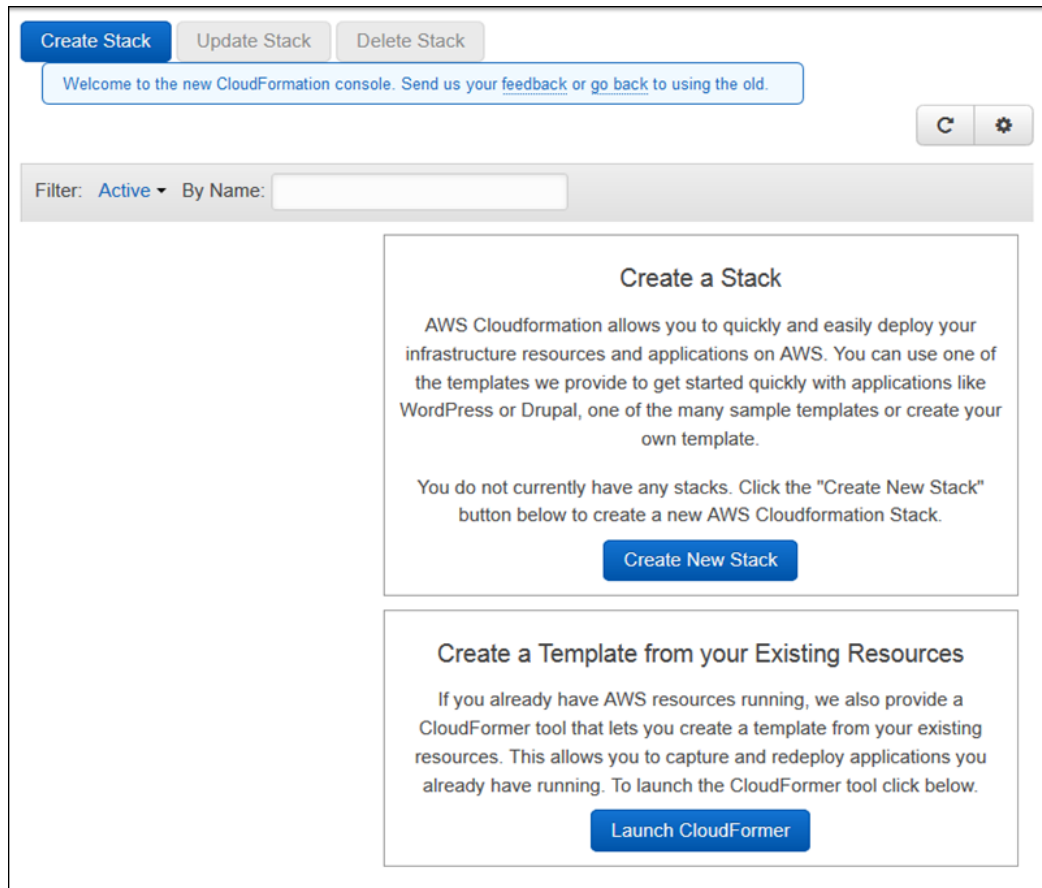
To create the Amazon AppStream standalone mode

1. Open the [AWS CloudFormation console](#).
2. From the navigation bar, select the same region where you created a new key pair or are using an existing key pair.

Important

Do not select a region that is different from your key pair was created.

3. In the [AWS CloudFormation console](#), select **Create New Stack**.



4. For **Name**, type a name to identify the stack. For example, `standaloneStack`.
5. Select **Provide an S3 URL template** and then enter `https://s3.amazonaws.com/appstream-public/AppStreamDeveloper.template` and then click **Next**.

Select Template

Specify a stack name and then select the template that describes the stack that you want to create.

Stack

An AWS CloudFormation stack is a collection of related resources that you provision and update as a single unit

Name

Template

A template is a JSON-formatted text file that describes your stack's resources and their properties. AWS CloudFormation stores the stack's template in an Amazon S3 bucket. [Learn more.](#)

Source ☒ Select a sample template

☐ Upload a template to Amazon S3

No file chosen

☐ Specify an Amazon S3 template URL

6. In **Specify Parameters**, do the following:
 - For **KeyPairName**, type the name of your existing key pair in the same region.
 - For **MicrosoftVirtualAudioDrivers**, type **yes**.
 - For **NvidiaGRIDDDrivers**, type **yes**.
 - Click **Next**.
7. In **Options**, do the following:
 - For **Key**, type **Name**.
 - For **Value**, type the name that you created earlier.
 - Click **Next**.
8. In **Review**, check that the parameters you entered are correct and then click **Create**.

Review

Template

Name	MyAppStreamStack
Template URL	https://s3.amazonaws.com/appstream-public/AppStreamDeveloper.template
Description	AppStream Stand-Alone is subject to the AWS Customer Agreement and AppStream Service Terms. AppStream Stand-Alone may only be used for development and testing; you may not host connections from end users on your AppStream Stand-Alone Instance. Do not download, transmit or otherwise distribute AppStream materials provided with the AppStream Stand-Alone instance.
Estimate cost	Link is not available

Parameters

ApplicationPresignedS3URL	https://s3.amazonaws.com/appstream-...
AppStreamSDKVersion	1.5
DeveloperPassword
InstallerParameters	
KeyPairName	AppStreamKeyPair
LauncherParameters	
LauncherPath	C:\app\XStxDirectXServer\XStxDirectXServer.exe
MicrosoftVirtualAudioDrivers	Y
NvidiaGRIDdrivers	Y
TightVNC	
Create IAM resources	False

Options

Tags

Name	MyAppStreamStack
------	------------------

Advanced

Notification	
Timeout	none
Rollback on failure	Yes

[Cancel](#) [Previous](#) [Create](#)

When the status of your stack changes to **CREATE_COMPLETE**, the Amazon AppStream standalone mode is deployed and ready to use. You can view your new server in the [Amazon EC2 console](#).

Locate the IP Address of Your standalone Host

1. In the [AWS CloudFormation console](#), click on the name of your standalone host to display details about the stack at the bottom of the browser window.
2. In the details pane, click **Outputs**. This displays the following values:

Key	Description
InstanceId	The identifier of the EC2 instance. You can use this value to locate the standalone host in the EC2 console.
PublicIp	The public IP address of the EC2 instance. You can use this value to connect the sample clients to your application without using an entitlement service.
PublicDnsName	The public DNS name of the EC2 instance.

Connect to Your Standalone Host

After AWS CloudFormation finished deploying the stack for your standalone streaming server, you can use Remote Desktop Connection (RDC) to connect to the EC2 instance, upload your application, and start your application. For more information, see [Connecting to Your Windows Instance](#) in the *Amazon Elastic Compute Cloud Microsoft Windows Guide*.

To log into your EC2 instance, you will need the administrator password and public IP address to your EC2 instance. While retrieving the administrator password and public IP address, you can download a Remote Desktop file that contains the connection information to your EC2 instance.

To get the administrator password, public IP address, and Remote Desktop File of your instance

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, select **Instances**. Select your instance, and click **Connect**.
3. In the **Connect To Your Instance** dialog box, record the public IP address and user name. You'll need the public IP address to stream to your client application. Click **Get Password** (it will take a few minutes after the instance is launched before the password is available).
4. Click **Browse** and navigate to the private key file that you saved when you created the key pair. Select the file and click **Open** to copy the entire contents of the file into contents box.
5. Click **Decrypt Password**. The console displays the default administrator password for the instance in the **Connect To Your Instance** dialog box, replacing the link to **Get Password** shown previously with the actual password.
6. Record the default administrator password. You need this password to connect to the instance and install your application.
7. Click **Download Remote Desktop File**. Your browser prompts you to either open or save the .rdp file. Save the file as you might need this file to administer the instance. When you have finished, you can click **Close** to dismiss the **Connect To Your Instance** dialog box.

Connect to Your Application with the Sample Windows Client

The Amazon AppStream SDK includes a sample pre-compiled client application, `AppStreamClient.exe`, that runs on Microsoft Windows 7 or greater. This client application does not call an entitlement service to connect to a streaming application, instead it uses the IP address of the hosted streaming application to connect. Because it does not use an entitlement service, the client application can only be used to connect to pre-release applications hosted locally or on a standalone GPU instance that you manually launched and configured in Amazon EC2. It cannot be used to connect to applications that are hosted on Amazon AppStream.

Despite this limitation, the client application is useful for testing your application during development because it gives you a known working client to test your streaming application against and you can develop and test your streaming application before creating a custom client application.

To use the pre-compiled client application to your streaming application, you'll need to obtain the IP address of your application, either the IP address of your local machine if you are running the streaming application locally, or an elastic IP address attached to an EC2 instance if you are running your streaming application on an GPU instance that you launched manually in Amazon EC2. Once you have the IP address, you pass it into as a command line parameter when you launch client application.

The client application accepts the following command line parameters:

Parameter	Required	Description
<code>-p <i>serverPort</i></code>	No	The port of the server to connect to. If not specified, the server port defaults to port 80.
<code>-i <i>sessionId</i></code>	No	The ID of the session to connect to. If not specified, the session identifier defaults to 9070-0.
<code>-w <i>videoWindowWidth</i></code>	No	The width of the video displayed by the client. If not specified, the video width defaults to 1280.
<code>-h <i>videoWindowHeight</i></code>	No	The height of the video displayed by the client. If not specified, the video height defaults to 720.

To connect to your streaming application using the pre-compiled client application

1. Open a command window in the `<SDK_dir>\precompiled_samples\windows\x64`
2. From the command line, run the following batch file.

quickRun64.bat *<public IP address>*

Replace *<public IP address>* with the public IP address of the standalone mode hosting your streaming application. You may optionally specify other command-line parameters as described in the preceding.

Note

The batch file sets the path to the required dependency files and starts the sample client application.

Clean Up Resources

When you are done testing your application, you should release the AWS resources you used to deploy the standalone streaming server to prevent further charges from accruing. To do so, you delete the AWS CloudFormation stack you created to deploy the standalone server, see [Deleting a Stack on the AWS CloudFormation Console](#) in the **AWS CloudFormation User Guide**.

Debug Your Streaming Application in Amazon AppStream Standalone Mode

You can debug your streaming application in Amazon AppStream standalone mode by installing the Remote Tools for Visual Studio on your Amazon EC2 instance. Remote Tools for Visual Studio allows you to debug your streaming application on the EC2 instance from another computer running Visual Studio.

To install Remote Tools for Visual Studio

1. From your computer, start Remote Desktop Connection (RDC) and connect to the EC2 instance. For more information, see [Connecting to Your Windows Instance](#) in the *Amazon Elastic Compute Cloud Microsoft Windows Guide*.
2. From the EC2 instance, download and install the version of Remote Tools for Visual Studio that matches the version of Visual Studio on your computer.
 - Remote Tools for Visual Studio 2013 at <http://www.microsoft.com/en-us/download/details.aspx?id=40781>.
 - Remote Tools for Visual Studio 2012 at <http://www.microsoft.com/en-us/download/details.aspx?id=38184>.
3. Restart the EC2 instance and connect to it using RDC.

Note

Restarting the EC2 instance may change its Public IP address.

To configure Remote Tools for Visual Studio

1. From the **Start** menu of the EC2 instance, click **Remote Debugger**.
2. From **Visual Studio Remote Debugging Monitor**, click **Tools > Options**.
3. In the **Options** dialog box, do the following:
 - a. For **TCP/IP port number**, type a port number of your choice. In this section, the port number is designated by **N**.
 - b. For **Authentication mode**, select **Windows Authentication**.
 - c. Click **OK**.

To configure your EC2 instance

In this procedure, you will configure the security group for your EC2 instance to accept inbound and outbound traffic from a specific TCP port. By limiting the traffic to a specific port during testing, you are reducing the chances of a malicious attack on your EC2 instance.

1. From your computer, open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, click **Instances**. Select your EC2 instance and click the link to the security group in the **Description** tab.
3. In the details pane, on the **Inbound** tab, click **Edit**.
4. In the dialog, click **Add Rule**, and then select **Custom TCP Rule** from the **Type** list. For **Port Range**, type the port number you entered in the earlier step. In the **Source** field, select **Anywhere**.
5. Click **Save**.

To start debugging

1. From your computer, start Visual Studio.
2. From the **Tools** menu, click **Attach to Process**.
3. In the **Attach to Process** dialog box, do the following:
 - a. For **Transport**, select **Default**.
 - b. For Qualifier, type the Public IP address of the EC2 instance using the form, `xxx.xxx.xxx.xxx:N`.

Where
 - `xxx.xxx.xxx.xxx` is the Public IP address of your EC2 instance.
 - `N` is the port number you specified in an earlier step.
 - c. Select **Show process from all users**.
 - d. In the **Available Process** list, select the process of your streaming application and then click **Attach**.

Build an Application Installer

To install your streaming application on Amazon AppStream, you will need a streaming application installer that does the following:. The installer must be a silent installer that is a single executable file (*.exe) which installs the streaming application and required dependency files without any user interaction.

- Runs silently without any user interaction.
- Exists as a single executable file (*.exe). Do not use Microsoft Windows Installer files (*.msi), batch files (*.bat), or self-extracting zip (*.zip) files.
- Installs the streaming application and any required dependency files.
- Installs the streaming application in a folder other than C:\AppStream.
- Sets the necessary file and folder permissions so that a Windows standard user account can run the streaming application. for more information on Windows accounts, see ["What is a standard user account?"](#) at Microsoft.com.

Build an Entitlement Service

An entitlement service authenticates and authorizes users. It is the gatekeeper between clients and your application, ensuring that only those clients entitled to access your application do so. Your entitlement service can authenticate users in a variety of ways: by comparing user login credentials to a list of subscribers in a database, by using an external login service such as [Login with Amazon](#), or by simply authenticating all clients.

An entitlement service:

1. Processes requests from clients to connect to your application.
2. Authenticates user credentials.
3. Checks whether the user is authorized to access your application.
4. Calls into Amazon AppStream to create new client sessions for authorized users.
5. Returns an entitlement URL to authorized clients that the client uses to access your application.

For more information about the lifecycle of a connection request, see [How a Client Connects to the Application](#) (p. 5).

Note

While you could implement the entitlement logic directly in the client, doing so is strongly discouraged because of the requirement to call into the Amazon AppStream service to create new sessions. It is more secure to have your AWS credentials built into a web service running on a server you control than compiled into client code running locally on end-user devices.

Topics

- [Design Considerations](#) (p. 74)
- [Build the Entitlement Web Service](#) (p. 74)
- [Publish Your Entitlement Service](#) (p. 79)
- [Sample Entitlement Request and Response](#) (p. 80)

Design Considerations for Your Entitlement Service

When you design the entitlement service for your application, there are several factors to consider:

- **Authentication mechanism**—How will you authenticate user credentials? Will you use an external service such as [Login with Amazon \(LWA\)](#), or an internal data store? If you have an existing log on process defined for your customers, you can integrate it with your entitlement service for a seamless customer experience when you add applications to your offerings.
- **Web service hosting**—Your entitlement service is a web service and must be continuously available to clients. If your entitlement service becomes unavailable, clients will not be able to create new connections. Because of this, you should host your entitlement service on a reliable platform. You can choose to host it on a physical server or in the cloud, on AWS infrastructure.
- **Service health monitoring**—Because your entitlement service is a crucial part of your product, you should monitor the health of your entitlement service and set alarms that trigger if it becomes slow to respond or unavailable. If you are hosting your entitlement service on AWS, you can use [CloudWatch](#) to monitor your entitlement service.

Build the Entitlement Web Service

Your entitlement service is a web service, handling incoming HTTP requests from clients and returning an HTTP response. The following topics walk you through the process of creating an entitlement service.

To simplify the process of writing an entitlement service, Amazon AppStream provides the Amazon AppStream SDK, which contains Java wrappers for the REST API of the Amazon AppStream service. These wrapper classes handle the overhead of signing your requests to the REST API and provide functions your entitlement service can call in order to create new client sessions. You can download the Amazon AppStream SDK from the links in [Downloads](#) (p. 10).

Documentation for the Java wrapper classes is provided in the Amazon AppStream SDK.

To build an entitlement service in a language other than Java, you can send HTTP requests directly to the [Amazon AppStream REST API](#) (p. 164).

Topics

- [Sample Entitlement Service](#) (p. 75)
- [Authenticate the Client](#) (p. 75)
- [Check Client Authorization for the Application](#) (p. 76)

- [Request a New Session from Amazon AppStream \(p. 77\)](#)
- [Return an Entitlement URL to the Client \(p. 78\)](#)

Sample Entitlement Service

Amazon AppStream provides a sample entitlement service implementation. You can download the sample entitlement service from the links in [Downloads \(p. 10\)](#).

You can use this as an example for designing your own entitlement service, and use the sample implementation during development and testing your application. .

The content of the sample organized as follows:

Directory	Description
/authorization	Classes to authenticate user credentials and check whether they are authorized to access an application.
/exceptions	Classes that return authorization exceptions to the client.
/model	Classes that interact with DynamoDB.
/rs	Classes that use JAX-RS to handle HTTP requests and responses.
/services	Classes providing the entitlement service logic.
/ui	Implementation of the web-based user interface used to add users and entitle them to access applications. .

In the following sections, we'll look at excerpts of code from the sample entitlement service to understand how to build a custom entitlement service.

Authenticate the Client

The sample entitlement service stores user credentials and entitlement mappings of users to applications in DynamoDB, a fast, fully managed NoSQL database service. For more information, see the [DynamoDB Developer Guide](#).

The following excerpt from `/rs/JaxRsEntitlementService.java` illustrates how the `JaxRsEntitlementService.requestEntitlement` method authenticates a user. This code resides in a `try-catch` block and throws an exception if a method call fails.

```
//look up the user credentials in DynamoDB
User user = entitlementService.getUserFromAuthorization(authorization);
```

The following excerpt from `/services/EntitlementService.java` shows the implementation of the `EntitlementService.getUserFromAuthorization` method.

The `getUserFromAuthorization` method looks up user credentials in an DynamoDB data store. If the user is not found, and the `createUserWhenNew` flag is not set, the method throws an exception. If the `createUserWhenNew` flag is set, the method creates a new user in the DynamoDB data store and populates it with the user credentials passed into the method. By default, `createUserWhenNew` is set to `true`.

```
public User getUserFromAuthorization(String authorization)
    throws AuthorizationException {
    if (authorization == null) {
        throw new AuthorizationException("Missing Authorization header.");
    }

    Identity identity = authorizationHandler.processAuthorization(authorization);

    User user = dynamoDBMapper.load(User.class, identity.getId());

    if (user == null) {
        if (!createUserWhenNew) {
            log.warn("No such user: " + identity.getId());

            throw new UserNotFoundException();
        }

        user = new User();

        user.setId(identity.getId());
        user.setName(identity.getName()); // May be null
        user.setEmail(identity.getEmail()); // May be null
        user.setEntitleAll(entitleAllWhenNew);
        user.setSessionCount(0);

        dynamoDBMapper.save(user);
    }

    return user;
}
```

Check Client Authorization for the Application

After your entitlement service has verified that the user credentials are valid, it should check whether the user holding those credentials is authorized to connect to the application.

The following excerpt from `/rs/JaxRsEntitlementService.java` illustrates how the `JaxRsEntitlementService.requestEntitlement` method checks whether a user is authorized to access an application. The `User` object passed into the `checkIfEntitled` method was created in the previous step, [Authenticate the Client \(p. 75\)](#).

```
//call Amazon AppStream to look up the application from its ID
Application application = entitlementService.getApplication(applicationId);

//check to see whether the user is entitled to access the application by looking
up entitlement mappings in DynamoDB
entitlementService.checkIfEntitled(user, application);
```

The following excerpts from `/services/EntitlementService.java` shows the implementation of the `EntitlementService.getApplication` and `EntitlementService.checkIfEntitled` methods.

The `getApplication` method calls Amazon AppStream to retrieve information about an application based on an application identifier. If the application is found, the method returns an `Application` object

populated with metadata about the application. If the application does not exist in Amazon AppStream, the method throws an exception.

```
public Application getApplication(String applicationId)
    throws ApplicationNotFoundException {
    try {
        return appstream.getApplications().getById(applicationId);
    } catch (Exception e) {
        log.error(e);

        throw new ApplicationNotFoundException("The application identified by
" + applicationId + " was not found.");
    }
}
```

The `checkIfEntitled` method checks the DynamoDB data store to see whether the user is entitled to access the application. Adding user-application mappings in the sample entitlement service is handled through a web-based interface. .

```
public void checkIfEntitled(User user, Application application)
    throws UserNotEntitledException {

    if (user == null || !user.isEntitled(application.getId())) {
        throw new UserNotEntitledException("You are not currently allowed to
use this application. Please ask the developer for access.");
    }
}
```

The implementation of the `isEntitled` method of the `User` class is located in `/model/User.java`.

Request a New Session from Amazon AppStream

After your entitlement service has verified that the user is authorized to access the application, it calls the `entitleSession` function of the current `Application` object to create a new client session for that application. If successful, the function returns the newly created `Session` object.

The following excerpt from `/services/EntitlementService.java` illustrates how to make this call.

When you call `entitleSession`, you have the option of passing in binary data using the `opaqueData` member of the `EntitleSessionInput` object. If supplied, this data is passed to the application when the new session is created. The format and content of this data is specific to the application. You might use `opaqueData`, for example, to initialize the client session with state information saved the last time this user connected to your application.

```
public Session entitleSession(User user, Application application) {
    Session session;

    try {
        EntitleSessionInput entitleSessionInput = new EntitleSessionInput();

        // Could optionally pass opaque data to the application that will service
the session.
        // entitleSessionInput.setOpaqueData("some application-specific data");
    }
}
```

```
        session = application.entitleSession(entitleSessionInput);
    } catch (AmazonServiceException e) {
        // Make the exception just a bit more obvious with an informational
        message.
        // It may not be obvious where the error occurred with just a stack
        trace.
        if (e.getErrorType().equals(AmazonServiceException.ErrorType.Service))
        {
            log.error("An error occurred in the AppStream service while entitling
            a session for app: " + application.getId(), e);
        }

        throw e;
    }

    user.storeSessionId(application.getId(), session.getId());
    user.setSessionCount(user.getSessionCount() + 1);

    dynamoDBMapper.save(user);

    return session;
}
```

Return an Entitlement URL to the Client

The following excerpt from `/rs/JaxRsEntitlementService.java` uses JAX-RS, a Java API for interacting with REST-ful web services. It processes incoming HTTP requests from clients to establish a connection to an application. If the client is entitled to access the application, this method returns an entitlement URL that the client can use to connect to the application. If the request is denied, it returns an error.

The logic exposed by this function is the entitlement service in a nutshell.

```
@POST
@Path("/{applicationId}")
@Produces(MediaType.TEXT_PLAIN)
public Response requestEntitlement(@HeaderParam("Authorization") String author
    ization,
                                @PathParam("applicationId") String applica
    tionId,
                                @FormParam("terminatePrevious") @Default
    Value("false") Boolean terminatePrevious) {
    try {
        User user = entitlementService.getUserFromAuthorization(authorization);

        Application application = entitlementService.getApplication(applica
    tionId);

        if (!entitlementService.shouldAlwaysEntitle()) {
            entitlementService.checkIfEntitled(user, application);
        }
    }
}
```

```
        if (checkPreviousSession) {
            entitlementService.checkForPreviousSession(user, application, terminatePrevious);
        }

        Session session = entitlementService.entitleSession(user, application);

        return response(Status.CREATED, session.getEntitlementUrl());
    } catch (AuthorizationException e) {
        String authenticateHeader = e.getAuthenticateHeader();

        if (authenticateHeader == null) {
            return response(Status.UNAUTHORIZED, e.getMessage());
        } else {
            return response(Status.UNAUTHORIZED, e.getMessage(), Collections.singletonMap("WWW-Authenticate", authenticateHeader));
        }
    } catch (UserNotEntitledException e) {
        return response(Status.FORBIDDEN, e.getMessage());
    } catch (ApplicationNotFoundException e) {
        return response(Status.NOT_FOUND, e.getMessage());
    } catch (SessionActiveException e) {
        return response(Status.CONFLICT, e.getMessage());
    }
}
```

Publish Your Entitlement Service

After you develop your entitlement service, you need to host it on a server where clients can send it HTTP requests. You host your entitlement service as you would any other web service. You can host it on a physical server or in the cloud.

The sample entitlement service includes a AWS CloudFormation template to automate the process of deploying it on the AWS cloud and instructions for how to use that template to deploy the sample entitlement service..

If you host your entitlement service on AWS, there are several services that can help:

- **Amazon Elastic Compute Cloud (Amazon EC2)**—Launches an Amazon AppStream host that runs your web server and performs server-side processing. For more information, see the [Amazon EC2 documentation](#).
- **AWS Elastic Beanstalk**—Automatically creates, deploys, and manages the IT infrastructure needed to run a custom application. For more information, see the [AWS Elastic Beanstalk documentation](#).
- **AWS CloudFormation**—Automatically deploys IT infrastructure on AWS using templates. For more information, see [the AWS CloudFormation documentation](#).
- **Amazon CloudWatch**—Collects and reports metrics on your AWS resources. For more information, see the [CloudWatch documentation](#).

In addition to the content above, AWS provides two guides that walk you through the process of hosting an web application. For more information, see [Getting Started with AWS: Web Application Hosting for Microsoft Windows](#) and [Getting Started with AWS: Web Application Hosting for Linux](#).

Sample Entitlement Request and Response

When a client attempts to connect to your application, it sends an HTTP request to your entitlement service. In the request, the client transmits an application identifier and a set of user credentials. Your entitlement service attempts to authenticate the user credentials and to authorize those credentials for the application. If the credentials are successfully authorized, your entitlement service returns a response indicating success that contains the entitlement URL that the client will use to connect to the application.

If authorization is not successful, your entitlement service returns a response that should include some indication of the reason authorization was not granted. Clients should include logic to gracefully handle failed authorizations.

Example Request from the Client

The following shows an example of the HTTP headers and JSON body of a request sent from the client to an entitlement service. It includes user credentials and an application ID.

```
POST /api/entitlements/5565ba3a-7e75-4bce-baad-436843ad209e HTTP/1.1
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y
zlib/1.2.5
Host: localhost:8080
Accept: */*
Authorization:Username myUserId
Content-Length: 0
Content-Type: application/x-www-form-urlencoded
```

Example Response from the Entitlement Service

The following shows an example of the HTTP headers and JSON body of a response sent from the entitlement service to a client. This illustrates an successful client authorization and returns an entitlement URL.

```
HTTP/1.1 201 Created
Content-Type: text/plain
Date: Fri, 01 Nov 2013 19:23:43 GMT
Content-Length: 97
Server: Jetty(9.0.6.v20130930)
https://appstream.us-east-1.amazonaws.com/entitlements/e018add2-242e-4396-8e47-
ca5cd1a6060b
```

Build a Client

The client application renders the video and audio stream from the application on a device. The client application also sends the input on the device to the application. This section provides guidelines on building client applications for the different platforms. The Amazon AppStream SDK includes both pre-compiled clients and sample code that you can compile into a client application. This section has excerpts from those samples.

Topics

- [Design Considerations \(p. 81\)](#)
- [Build a Client for Android \(p. 81\)](#)

- [Build a Client for iOS](#) (p. 89)
- [Build a Client for OS X](#) (p. 101)
- [Build a Client for Windows](#) (p. 112)
- [Codec and Open Source Licensing](#) (p. 121)

Design Considerations for Your Clients

When creating your client, take into account the following:

- **Resolution differences between the application and the device.** Your application may be designed for an resolution that is different from the resolution of your client. Consider how you will deal with that discrepancy. the resolution difference is especially important if the client will send inputs at a high precision to the application. An example of high precision input is if you are editing bitmap graphics at the pixel level.
- **Applications can have more mouse events that the client has input events.** Amazon AppStream interprets events from the client as key or mouse events. However, the number of device input events are less than available mouse events. If your application handles mouse events that do not have corresponding device events, you will need to design for those discrepancies.

Build a Client for Android

The Amazon AppStream SDK provides libraries that you build into your client to receive and decode video and audio from the application and send device inputs and raw inputs to the application.

The following object classes are provided by the client libraries in the Amazon AppStream SDK.

Class	Description
AppStreamInterface	This class manages client sessions between clients and applications.
ConnectDialogFragment	This class creates sessions between clients and applications.
DesQuery	This class is for use with the entitlement service.
ErrorDialogFragment	This class is to display messages.
GL2JNIView	This class is for the video in the client.
SampleClientActivity	This class is for the client interface.

Your client implements and populate the following structures in order to receive session and application events.

Interface	Description
ConnectDialogListener	Receives callbacks from a client.
DesQueryListener	Receives callbacks from the entitlement service

Lifecycle of a Client for Android

Your client communicates with the application on Amazon AppStream through libraries of the Amazon AppStream SDK.

The lifecycle of a client for Android is as follows:

1. **Request authorization from the entitlement service to connect to the application.** The entitlement service will return the URL to connect to your application.
2. **Connect to the application.** The object manages the session between the client and the application.
3. **Send client inputs to the application.**
4. **Close the client.**

Build the Sample Client for Android

The Amazon AppStream SDK contains source code that you can build into a sample client file. To learn how to build the source code, see the `\doc` directory.

Choose a Color Subsampling Rate

Amazon AppStream streams the video at the YUV420 color subsampling rate to client applications. If your streaming application, client application and device support YUV444, you can set your client application to display the video at YUV444. The example client application in Amazon AppStream SDK is configured to accept and render a stream in the YUV444 color subsampling rate.

When the client application connects to the streaming application, Amazon AppStream compares the color subsampling options available on the client application with the options advertised on the streaming application. Amazon AppStream then selects the highest color resolution supported by both the client application and streaming application. Amazon AppStream then calls the `XStxIClientListener2FcnSetConfiguration` callback function that the client application supplied and passes the structure with the `XStxChromaSampling` setting.

The following excerpt from `AndroidVideoDecoder.cpp` illustrates this step. This file is in the `<SDK_dir>\example_src\client\src\android\jni` directory.

```
/**
 * Provide Chroma sampling capability.
 */
bool AndroidVideoDecoder::isChromaSamplingSupported(XStxChromaSampling chromaSampling)
{
    // If we have hardware, then ask the hardware whether we have
    XSTX_H264_PROFILE_HIGH444.
    // If not, then we fall back to 420.
    if (androidHWDecodeAvailable() && (androidGetProfile() != XSTX_H264_PROFILE_HIGH444))
    {
        return chromaSampling == XSTX_CHROMA_SAMPLING_YUV420;
    }
    else
    {
        return chromaSampling == XSTX_CHROMA_SAMPLING_YUV420
            || chromaSampling == XSTX_CHROMA_SAMPLING_YUV444;
    }
}
```



```
}  
}
```

The following excerpt from `VideoPipeline.cpp` illustrates this step. This file is in the `<SDK_dir>\example_src\client\src\android\jni` directory.

```
decoderCapabilities->mProfile = XSTX_H264_PROFILE_HIGH444;
```

Get Authorization to Connect to Your Application

The first activity in your client is to get the Entitlement URL from the endpoint of the entitlement service. The entitlement service needs information in order to provide the Entitlement URL.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `makeQuery` method of `DesQuery.java`.

```
public void makeQuery(String address, final String appid, final String userid)
{
    if (!address.startsWith("http"))
    {
        address = "http://" + address;
    }

    if (!address.endsWith("/"))
    {
        address += "/";
    }

    // if there's no path, add api/entitlements/
    if (!address.matches("http[s]?://.+/.+"))
    {
        address += "api/entitlements/";
    }

    final String cleanAddress = address;

    Thread httpThread = new Thread() {

        @Override
        public void run() {
            HttpParams httpParameters = new BasicHttpParams();

            // timeout for a connection in ms
            HttpConnectionParams.setConnectionTimeout(httpParameters, 1200);

            // timeout when waiting for data
            HttpConnectionParams.setSoTimeout(httpParameters, 1500);

            HttpClient client = new DefaultHttpClient();
            HttpPost request = new HttpPost();
            String queryURL = cleanAddress+appid;
            try {
```

```
        request.setURI(new URI(queryURL));
    } catch (URISyntaxException e) {

        sendError("Either address or appid isn't in the correct format. Generated
URL: "+queryURL+" Error: "+e.getLocalizedMessage());
        return;
    }

    request.setHeader("Authorization", "Username"+userid );
    request.setHeader("Content-Type", "application/x-www-form-urlencoded"
);

    List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(1);

    nameValuePairs.add(new BasicNameValuePair("terminatePrevious", "true"));

    String entitlementResult = null;
    try {
        request.setEntity(new UrlEncodedFormEntity(nameValuePairs));
        HttpResponse response = client.execute(request);
        if (response.getStatusLine().getStatusCode()<200 || response.get
StatusLine().getStatusCode()>201)
        {
            if (response.getStatusLine().getStatusCode()==503) {
                sendError("All of our streaming servers are currently in use. Please try
again in a few minutes. [503]");
            } else if (response.getStatusLine().getStatusCode()==404) {
                sendError("The link to the streaming server has expired. [404]");
            } else {
                sendError("Query failed with response: "+response.getStatusLine());
            }
        }
        return;
    }

    BufferedReader result =
        new BufferedReader(
            new InputStreamReader(
                response.getEntity().getContent()
            )
        );

        final String url = result.readLine().trim();
        Log.v(TAG,"Resulting URL: "+url);
        result.close();

        HttpGet entitlementRequest = new HttpGet(url);
        response = client.execute(entitlementRequest);

    result = new BufferedReader(
        new InputStreamReader(
            response.getEntity().getContent()
        )
    );

    StringBuilder buffer = new StringBuilder(1024);

    char[] charBuf = new char[400];
    while (result.read(charBuf,0,400)>0)
```

```
{
    buffer.append(charBuf);
}

entitlementResult = buffer.toString();

JSONObject object = (JSONObject) new JSONTokener(entitlementResult).next
Value();
final String sessionId = object.optString("sessionId");
final String ec2Host = object.optString("ec2Host");

if (sessionId==null || ec2Host==null)
{
    final String message = object.optString("message");
    if (message!=null)
    {
        sendError("Entitlement query failed with response: "+message+"["+re
sponse.getStatusLine()+"]");
        return;
    }
    sendError("Error parsing entitlement result: "+entitlementResult);
    return;
}

mActivity.runOnUiThread(new Runnable(){
    @Override
    public void run() {
        String url =
            String.format(Locale.US,"%s:80?sessionId=%s",ec2Host,sessionId);

        Log.v(TAG,"Sending host url "+url);
        mListener.onDesQuerySuccess(url);
    }});

} catch (ClientProtocolException e) {
    sendError("Protocol Exception: "+e.getLocalizedMessage());

} catch (IOException e) {
    sendError("Problem With Connection: "+e.getLocalizedMessage());

} catch (JSONException e) {
    sendError("Error reading entitlement result: "+e.getLocalizedMessage()+"
Result: "+entitlementResult);
}
}
};

httpThread.start();
}
```

The function is successful when the entitlement service calls a callback function with the Entitlement URL to start the session with the application.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates the callback function is called to connect the client to the application. The excerpt is from the `onDesQuerySuccess` method of `SampleClientActivity.java`.

```
public void onDesQuerySuccess(String address) {
    AppStreamInterface.connect(address);
    AppStreamInterface.newFrame();
}
```

Send Your Client Inputs to Your Application

The client sends inputs to the application through several methods, depending on the input type.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates a method that sends motion inputs to the application. The excerpt is from the `dispatchGenericMotionEvent` method in `SampleClientActivity.java`.

```
public boolean dispatchGenericMotionEvent (MotionEvent event) {
    if (event.getSource()==InputDevice.SOURCE_MOUSE)
    {
        event.getPointerCoords(0, mCoordHolder);
        switch (event.getAction())
        {
            case MotionEvent.ACTION_HOVER_MOVE :
                AppStreamInterface.mouseEvent((int)mCoordHolder.x,(int)mCoordHolder.y,
0);    break;

            default:
                return super.dispatchGenericMotionEvent(event);
        }
        return true;
    }
    else if (event.getSource()==InputDevice.SOURCE_JOYSTICK
        && event.getAction() == MotionEvent.ACTION_MOVE) {
        Log.v(TAG,"Joystick event:"+event.toString());
        /// @todo: Handle motion event
        return true;
    }

    return super.dispatchGenericMotionEvent(event);
}
```

The following excerpt from the sample client in the Amazon AppStream SDK illustrates a method that sends touch inputs to the application. The excerpt is from the `dispatchTouchEvent` method in `SampleClientActivity.java`.

```
public boolean dispatchTouchEvent (MotionEvent event) {

    if (mScaleGestureDetector.onTouchEvent(event))
    {
        if (mScaleGestureDetector.isInProgress())
            return true;
    }

    int flags = 0;
    if (event.getSource()==InputDevice.SOURCE_TOUCHSCREEN)
```

```
{
    flags = AppStreamInterface.CET_TOUCH_FLAG ;
}

event.getPointerCoords(0, mCoordHolder);
switch (event.getAction())
{
    case MotionEvent.ACTION_MOVE :
        AppStreamInterface.mouseEvent((int)mCoordHolder.x,(int)mCoordHolder.y, flags); break;
    case MotionEvent.ACTION_DOWN :
        AppStreamInterface.mouseEvent((int)mCoordHolder.x,(int)mCoordHolder.y, AppStreamInterface.CET_MOUSE_1_DOWN|flags); break;
    case MotionEvent.ACTION_UP :
        AppStreamInterface.mouseEvent((int)mCoordHolder.x,(int)mCoordHolder.y, AppStreamInterface.CET_MOUSE_1_UP|flags);
        AppStreamInterface.mouseEvent((int)-1000,(int)-1000, flags); break;

    default:
        return super.dispatchGenericMotionEvent(event);
}
return super.dispatchTouchEvent(event);
}
```

The following excerpt from the sample client in the Amazon AppStream SDK illustrates methods that sends key inputs to the application. The excerpt is from `SampleClientActivity.java`.

```
private void onKey(KeyEvent msg, boolean down)
{
    int keyCode =msg.getKeyCode();
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_RIGHT :
            AppStreamInterface.keyPress(0x27,down); break;
        case KeyEvent.KEYCODE_DPAD_LEFT :
            AppStreamInterface.keyPress(0x25,down); break;
        case KeyEvent.KEYCODE_DPAD_UP :
            AppStreamInterface.keyPress(0x26,down); break;
        case KeyEvent.KEYCODE_DPAD_DOWN :
            AppStreamInterface.keyPress(0x28,down); break;
        case KeyEvent.KEYCODE_FORWARD_DEL:
            AppStreamInterface.keyPress(0x2E,down); break;
        case KeyEvent.KEYCODE_DEL:
            AppStreamInterface.keyPress(0x08,down); break;
        case KeyEvent.KEYCODE_ALT_LEFT:
            AppStreamInterface.keyPress(0x12,down);
            AppStreamInterface.keyPress(0xA4,down); break;
        case KeyEvent.KEYCODE_ALT_RIGHT:
            AppStreamInterface.keyPress(0x12,down);
            AppStreamInterface.keyPress(0xA5,down); break;
        case KeyEvent.KEYCODE_CTRL_LEFT:
            AppStreamInterface.keyPress(0x11,down);
            AppStreamInterface.keyPress(0xA2,down); break;
        case KeyEvent.KEYCODE_CTRL_RIGHT:
            AppStreamInterface.keyPress(0x11,down);
    }
}
```

```
        AppStreamInterface.keyPress(0xA3,down); break;
    case KeyEvent.KEYCODE_SHIFT_LEFT:
        AppStreamInterface.keyPress(0x10,down);
        AppStreamInterface.keyPress(0xA0,down); break;
    case KeyEvent.KEYCODE_SHIFT_RIGHT:
        AppStreamInterface.keyPress(0x10,down);
        AppStreamInterface.keyPress(0xA1,down); break;
    case KeyEvent.KEYCODE_APOSTROPHE:
        AppStreamInterface.keyPress('\'',down); break;
    case KeyEvent.KEYCODE_AT:
        AppStreamInterface.keyPress('@',down); break;
    case KeyEvent.KEYCODE_ENTER:
        AppStreamInterface.keyPress(0x0D,down); break;
    case KeyEvent.KEYCODE_BREAK:
        AppStreamInterface.keyPress(0x03,down); break;
    case KeyEvent.KEYCODE_VOLUME_UP:
        AppStreamInterface.keyPress(0xAF,down); break;
    case KeyEvent.KEYCODE_VOLUME_DOWN:
        AppStreamInterface.keyPress(0xAE,down); break;
    case KeyEvent.KEYCODE_MEDIA_NEXT:
        AppStreamInterface.keyPress(0xB0,down); break;
    case KeyEvent.KEYCODE_MEDIA_PREVIOUS:
        AppStreamInterface.keyPress(0xB1,down); break;
    case KeyEvent.KEYCODE_MEDIA_STOP:
        AppStreamInterface.keyPress(0xB2,down); break;
    case KeyEvent.KEYCODE_MEDIA_PLAY:
        AppStreamInterface.keyPress(0xFA,down); break;
    case KeyEvent.KEYCODE_MEDIA_PLAY_PAUSE:
        AppStreamInterface.keyPress(0xB3,down); break;
    case KeyEvent.KEYCODE_ESCAPE:
    case KeyEvent.KEYCODE_BACK:
        AppStreamInterface.keyPress(0x1B,down); break;

    default:
    {
        String key = new String(Character.toString((char)msg.getUnicodeChar()));

        keyCode = key.toUpperCase(Locale.US).codePointAt(0);
        AppStreamInterface.keyPress(keyCode,down);
    }
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent msg) {
    onKey(msg,true);
    return super.onKeyDown(keyCode, msg);
}

@Override
public boolean onKeyUp(int keyCode, KeyEvent msg) {
    onKey(msg,false);
    return super.onKeyUp(keyCode, msg);
}
```

Build a Client for iOS

The Amazon AppStream SDK provides client libraries that you build into your client to receive and decode video and audio from the application and send device inputs and raw inputs to the application.

The following object classes are provided by the client libraries in the Amazon AppStream SDK.

Class	Description
XStxClientLibrary	The top level object of the client libraries. Your client uses this object to ensure that it is calling into the version of the libraries it was compiled against and to create the <code>XStxClient</code> object.
XStxClient	This object manages client sessions and sends events to your application when Amazon AppStream assigns your application a client session or terminates a client session.

Your client implements and populate the following structures in order to receive session and application events.

Interface	Description
XStxIClientListener	Receives callbacks from a client.
XStxIVideoDecoder	Receives and decodes the video frame.
XStxIVideoRenderer	Renders the video frame.
XStxIRawVideoFrameAllocator	Receives and recycles the video frame.
XStxIAudioDecoder	Receives and decodes the audio frame.
XStxIAudioRenderer	Renders the audio frame.
XStxIRawAudioFrameAllocator	Receives and recycles the audio frame.

Lifecycle of a Client for iOS

Your client communicates with the application on Amazon AppStream through libraries of the Amazon AppStream SDK.

The lifecycle of a client is as follows:

1. **Create the `XStxClientLibraryHandle` object by calling the `XStxClientLibraryCreate` function to return a `clientLibraryHandle`.** The object interacts with the client side portion of the API through the `clientLibraryHandle`.
2. **Create the `XStxClient` object by calling the `XStxClientCreate` function to return a `XStxClientHandle` client handle.** The object manages the session between the client and the application.
3. **Instantiate and populate the `XStxIClientListener` structure.** This object responds to the callback functions from the application when a session is established between the client and the application. After populating the listener structure, call the `XStxClientSetListener` function with the client handle and a pointer to the `XStxIClientListener` structure.
4. **Initialize a video module to get, decode, and render the video frames.** To initialize this module, the client has to instantiate the following structures:

- **XStxIVideoDecoder** Gets and decodes the video frame.
 - **XStxIVideoRenderer** Renders the video frame.
 - **XStxIRawVideoFrameAllocator** Gets and recycles the video frame.
5. **Initialize an audio module to get, decode, and render the audio frames.** To initialize this module, the client has to instantiate the following structures:
 - **XStxIAudioDecoder** Gets and decodes the audio frame.
 - **XStxIAudioRenderer** Renders the audio frame.
 - **XStxIRawAudioFrameAllocator** Gets and recycles the audio frame.
 6. **Configure a session between the client and application.** Call the `XStxClientSetEntitlementUrl` function with the endpoint of the entitlement service to configure this session.
 7. **Call the `xStxClientStart` function to start the session.** Once the session is established, the `XStxClient` object will invoke the `XStxIClientListenerFcnClientReady` function on the `XStxIClientListener` structure. You can then call the `xStxClientLibraryRecycle` function to recycle the client library handle.
 8. **The client sends user inputs to the application**
 9. **The client closes the session.**

Build the Sample Client for iOS

The Amazon AppStream SDK contains source code that you can build into a sample client file. To learn how to build the source code, see the `\doc` directory.

Create Your Client

To create a client, create an `XStxClientLibraryHandle` object by using the `XStxClientLibraryCreate` function. The `XStxClientLibraryHandle` object is the top level object that you will use to interact with the libraries you will use to connect to sessions, render the audio and video, and send inputs to the application.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `initXStxClient` function of `AppStreamSampleClientViewController.m`.

```
-(XStxResult) initXStxClient
{
    XStxResult createResult = XSTX_RESULT_OK;

    /// create Singleton of client library handle
    createResult = XStxClientLibraryCreate(XSTX_CLIENT_API_VERSION_MAJOR,
                                           XSTX_CLIENT_API_VERSION_MINOR,
                                           &mClientLibraryHandle);

    if (createResult != XSTX_RESULT_OK)
    {
        [self printResult:createResult withMessage:@"XStxClientLibraryCreate
failed"];
        return createResult;
    }
}
```



```
    }  
  
    return XSTX_RESULT_OK;  
}
```

The function creates a client library handle (`clientLibraryHandle`) that the client will use to create a client object. The client then creates a client object by calling the `XStxClientCreate(mClientLibraryHandle` function with the client library handle. The function returns a handle to the client.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `connectXStxClient` function of `AppStreamSampleClientViewController.m`.

```
// Create a XStx Client instance  
result = XStxClientCreate(mClientLibraryHandle, &mClientHandle);  
if (result != XSTX_RESULT_OK)  
{  
    [self printResult:result withMessage:@"XStxClientCreate failed"];  
    return;  
}
```

The client object needs a structure to respond to the callback functions from the application. The client creates and populates this structure.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `H264ToYuv::init` function of `H264to Yuv.cpp`.

```
/** initialize */  
XStxResult H264ToYuv::init()  
{  
  
    AvHelper::initialize();  
  
    AVCodec *codec = avcodec_find_decoder(CODEC_ID_H264);  
    if(NULL == codec)  
    {  
        return XSTX_RESULT_AUDIO_DECODER_NULL;  
    }  
  
    mCodecContext = avcodec_alloc_context3(codec);  
    if(NULL == mCodecContext)  
    {  
        return XSTX_RESULT_VIDEO_DECODING_ERROR;  
    }  
  
    mCodecContext->codec_type      = AVMEDIA_TYPE_VIDEO;  
    mCodecContext->codec_id        = CODEC_ID_H264;  
    mCodecContext->pix_fmt         = PIX_FMT_YUV420P;  
  
    if(avcodec_open2(mCodecContext, codec, NULL) < 0)  
    {  
        return XSTX_RESULT_VIDEO_DECODING_ERROR;  
    }  
}
```

```
    av_init_packet(&mAvPacket);

    return XSTX_RESULT_OK;
}
```

After populating the structure, call a function to configure a listener for callback functions.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `clientReady` function of `AppStreamSampleClientViewController.m`.

```
// start a client listener
if ((result = XStxClietSetListener(mClientHandle, &mClientListener))
    != XSTX_RESULT_OK)
{
    [self printResult:result withMessage:@"XStxClietSetListener failed"];
    return;
}
```

The client uses structures to get, render, and decode video and audio frames.. The next step is to create and populate these structures. The sample client instantiates a class to create and populate the structures.

To handle a video frame, the client create and populate the following structures defined in `VideoModule.h`:

- **XStxIVideoDecoder**. Used to get and decode a video frame.
- **XStxIVideoRenderer**. Used to render a video frame.
- **XStxIRawVideoFrameAllocator**. Used to get and recycle a video frame.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates populating the structures for video. The excerpt is from `VideoModule.m`.

```
bool VideoModule::initialize(XStxClietHandle mClientHandle,
                             VideoDecoder* decoder, GLKView & rw)
{
    // initialize frame allocator

    if(decoder == nil)
    {
        return false;
    }

    mDecoder = decoder;

    mStxFrameAllocator.mInitFcn = &allocatorInit;
    mStxFrameAllocator.mInitCtx = this;
    mStxFrameAllocator.mGetVideoFrameBufferFcn = &allocatorGetFrame;
    mStxFrameAllocator.mGetVideoFrameBufferCtx = this;
    mStxFrameAllocator.mRecycleVideoFrameBufferFcn = &allocatorRecycleFrame;
    mStxFrameAllocator.mRecycleVideoFrameBufferCtx = this;
    mStxFrameAllocator.mSize = sizeof(mStxFrameAllocator);
    if (XStxClietSetVideoFrameAllocator(mClientHandle, &mStxFrameAllocator)
        != XSTX_RESULT_OK)
    {

```

```

        printf("Failed to SetVideoFrameAllocator.\n");
        return false;
    }

    mRenderer = &rw;

    mStxRenderer.mRenderVideoFrameFcn = &renderFrame;
    mStxRenderer.mRenderVideoFrameCtx = this;
    mStxRenderer.mSetMaxResolutionFcn = &rendererMaxResolution;
    mStxRenderer.mSetMaxResolutionCtx = this;
    mStxRenderer.mSize = sizeof(mStxRenderer);
    if (XStxClietSetVideoRenderer(mClientHandle, &mStxRenderer)
        != XSTX_RESULT_OK)
    {
        printf("Failed to set VideoRenderer.\n");
        return false;
    }

    ((GLViewManager *)mRenderer.delegate).decodeType = mDecoder->getDecodeType();

    mStxDecoder.mGetCapabilitiesCtx = mDecoder;
    mStxDecoder.mGetCapabilitiesFcn = &videoDecoderGetCapabilities;
    mStxDecoder.mDecodeVideoFrameFcn = &decodeFrame;
    mStxDecoder.mDecodeVideoFrameCtx = mDecoder;
    mStxDecoder.mStartFcn = &videoDecoderStart;
    mStxDecoder.mStartCtx = mDecoder;
    mStxDecoder.mSize = sizeof(mStxDecoder);
    if (XStxClietSetVideoDecoder(mClientHandle, &mStxDecoder)
        != XSTX_RESULT_OK)
    {
        printf("Failed to set SetVideoDecoder.\n");
        return false;
    }
    return true;
}

```

To handle an audio frame, the client create and populate the following structures defined in `AudioModule.h`:

- **XStxIAudioDecoder**. Used to get and decode an audio frame.
- **XStxIAudioRenderer**. Used to render an audio frame.
- **XStxIRawAudioFrameAllocator**. Used to get and recycle an audio frame.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates populating the structures for audio. The excerpt is from `AudioModule.m`.

```

/**
 * Initialize audio module
 * @param[in] clientHandle handle to XStx client
 */
bool AudioModule::initialize(XStxClietHandle clientHandle)

```

```
{
    mStxFrameAllocator.mInitFcn = &allocatorInit;
    mStxFrameAllocator.mInitCtx = this;
    mStxFrameAllocator.mGetAudioFrameBufferFcn = &allocatorGetFrame;
    mStxFrameAllocator.mGetAudioFrameBufferCtx = this;
    mStxFrameAllocator.mRecycleAudioFrameBufferFcn = &allocatorRecycleFrame;
    mStxFrameAllocator.mRecycleAudioFrameBufferCtx = this;
    mStxFrameAllocator.mSize = sizeof(mStxFrameAllocator);

    if (XStxClientSetAudioFrameAllocator(clientHandle, &mStxFrameAllocator)
        != XSTX_RESULT_OK)
    {
        printf("XStxClientSetAudioFrameAllocator() failed.\n");
        return false;
    }

    // initialize decoder
    mDecoder = new (std::nothrow) OpusToPcm();
    if (!mDecoder)
    {
        printf("Failed to create audio decoder.\n");
        return false;
    }

    mStxDecoder.mStartFcn = &decoderStart;
    mStxDecoder.mStartCtx = mDecoder;
    mStxDecoder.mDecodeAudioFrameFcn = &decoderDecodeFrame;
    mStxDecoder.mDecodeAudioFrameCtx = mDecoder;
    mStxDecoder.mSize = sizeof(mStxDecoder);
    if (XStxClientSetAudioDecoder(clientHandle, &mStxDecoder)
        != XSTX_RESULT_OK)
    {
        printf("XStxClientSetAudioDecoder() failed.\n");
        return false;
    }

    // initialize renderer
    mRenderer = new (std::nothrow) AudioRenderer(mFramePool,
        clientHandle);
    if (!mRenderer)
    {
        return false;
    }
    mStxRenderer.mStartFcn = &start;
    mStxRenderer.mStartCtx = mRenderer;
    mStxRenderer.mSize = sizeof(mStxRenderer);
    if (XStxClientSetAudioRenderer(clientHandle, &mStxRenderer)
        != XSTX_RESULT_OK)
    {
        printf("XStxClientSetAudioRenderer() failed. \n");
        return false;
    }

    return true;
}
```

The client is now ready to configure and start a session with the application. The client configures a session by calling a function with the Entitlement URL.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `connectXStxCliet` function in `AppStreamSampleClientViewController.m`.

```
if ((result = XStxClietSetEntitlementUrl(mClientHandle, [url UTF8String]))
    != XSTX_RESULT_OK)
{
    [self printResult:result withMessage:@"XStxClietSetEntitlementUrl
failed"];
    return;
}
```

The client starts a session by calling the `XStxClietStart` function.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `connectXStxCliet` function in `AppStreamSampleClientViewController.m`.

```
// this method is non-blocking
// defer XStxClietLibraryRecycle call for unload methods
if ((result = XStxClietStart(mClientHandle)) != XSTX_RESULT_OK)
{
    [self printResult:result withMessage:@"XStxClietStart failed"];
    return;
}

NSString *message = [NSString stringWithFormat:@"STX client initialized.
Connecting to %@:",url];
[self printResult:result withMessage:message];

[self hideBackground];
[self setupInputs];
```

Choose a Color Subsampling Rate

Amazon AppStream streams the video at the YUV420 color subsampling rate to client applications. If your streaming application, client application and device support YUV444, you can set your client application to display the video at YUV444. The example client application in Amazon AppStream SDK is configured to accept and render a stream in the YUV444 color subsampling rate.

When the client application connects to the streaming application, Amazon AppStream compares the color subsampling options available on the client application with the options advertised on the streaming application. Amazon AppStream then selects the highest color resolution supported by both the client application and streaming application. Amazon AppStream then calls the `XStxClietListener2FcnSetConfiguration` callback function that the client application supplied and passes the structure with the `XStxChromaSampling` setting.

The following excerpt from `H264ToYuv.cpp` illustrates this step. This file is in the `<SDK_dir>\example_src\client\src\apple\ios` directory.

```
bool H264ToYuv::isChromaSamplingSupported(XStxChromaSampling chromaSampling)
{
    return chromaSampling == XSTX_CHROMA_SAMPLING_YUV420
}
```

```
        || chromaSampling == XSTX_CHROMA_SAMPLING_YUV444;  
    }
```

The following excerpt from `VideoPipeline.cpp` illustrates this step. This file is in the `<SDK_dir>\example_src\client\src\apple\ios` directory.

```
decoderCapabilities->mProfile = XSTX_H264_PROFILE_HIGH444;
```

Send Your Client Inputs to the Application

The client can send key, and touch inputs to the application.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates how keyboard events are handled. The excerpt is from the `setupInputs` function of `AppStreamSampleClientViewController.m`.

```
-(void) setupInputs  
{  
    // KEYBOARD EVENTS -- pass through iOS key presses  
  
    CGFloat y = 30.0f;  
  
    // add a button for keyboard input  
    UIButton *kbbutton = [UIButton buttonWithTypeCustom];  
    [kbbutton addTarget:self action:@selector(toggleKeyboard:) forControlEvents:UIControlEventTouchUpInside];  
    CGFloat buttonWidth = 72.0f;  
    kbbutton.frame = CGRectMake(self.view.bounds.size.width-buttonWidth,  
                               y,buttonWidth,buttonWidth);  
    [kbbutton setImage:[UIImage imageNamed:@"button-keyboard"] forState:UIControlStateNormal];  
    kbbutton.autoresizingMask = UIViewAutoresizingFlexibleBottomMargin|UIViewAutoresizingFlexibleLeftMargin;  
    [self.view addSubview:kbbutton];  
  
    // add offscreen text field for keyboard input and set delegate  
    // note: text field is off screen to hide input from user  
    mInputText = [[UITextField alloc] initWithFrame:CGRectMake(-500,-500,100,20)];  
  
    [self.view addSubview:mInputText];  
    mInputText.delegate = self;  
  
    /// SENDING TOUCH AS MOUSE EVENTS  
    mTreatTouchesAsMouse = NO;  
    // add a button for mouse input  
    UIButton *mousebutton = [UIButton buttonWithTypeCustom];  
    [mousebutton addTarget:self action:@selector(toggleMouse:) forControlEvents:UIControlEventTouchUpInside];  
    mousebutton.frame = CGRectMake(self.view.bounds.size.width-(buttonWidth*2),  
                                  y,buttonWidth,buttonWidth);  
    [mousebutton setImage:[UIImage imageNamed:@"button-mouse"] forState:UIControlStateNormal];  
}
```

```
        mousebutton.autoresizingMask = UIViewAutoresizingFlexibleBottomMargin|UIViewAutoresizingFlexibleLeftMargin;
        [self.view addSubview:mousebutton];

        mShouldTrackGesture = false;
        UIButton *handbutton = [UIButton buttonWithType:UIButtonTypeCustom];
        [handbutton addTarget:self action:@selector(toggleGesture:) forControlEvents:UIControlEventTouchUpInside];
        handbutton.frame = CGRectMake(self.view.bounds.size.width-(buttonWidth*3),
                                     y,buttonWidth,buttonWidth);
        [handbutton setImage:[UIImage imageNamed:@"button-hand"] forState:UIControlStateNormal];
        handbutton.autoresizingMask = UIViewAutoresizingFlexibleBottomMargin|UIViewAutoresizingFlexibleLeftMargin;
        [self.view addSubview:handbutton];

        mPanGestureRecognizer = [[UIPanGestureRecognizer alloc] initWithTarget:self
        action:@selector(handlePanGesture)];
        mPanGestureRecognizer.enabled = false;
        [self.view addGestureRecognizer:mPanGestureRecognizer];
    }
```

The following excerpt from the sample client in the Amazon AppStream SDK illustrates how gesture events are handled. The excerpt is from the `handlePanGesture` function of `AppStreamSampleClientViewController.m`.

```
void) handlePanGesture:(UIPanGestureRecognizer*) gestureRecognizer
{
    CGPoint location = [gestureRecognizer locationInView:self.view];

    NSString *jsonString = [NSString stringWithFormat:
    Format:@"{\"state\": \"%d\", \"xy\": [%.2f, %.2f]}", gestureRecognizer.state, location.x, location.y ];

    switch (gestureRecognizer.state) {
        case UIGestureRecognizerStateEnded:
            [self showStatus:@""];
            break;
        case UIGestureRecognizerStatePossible:
        case UIGestureRecognizerStateBegan:
        case UIGestureRecognizerStateChanged:
        case UIGestureRecognizerStateFailed:
        case UIGestureRecognizerStateCancelled:
        default:
            [self sendRawInput:jsonString];
            break;
    }
}
```

Touch events are handled as mouse events.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from `AppStreamSampleClientViewController.m`.

```
// In this example we send the touch events as mouse input
-(void) toggleMouse:(UIButton*)sender
{
    if(mShouldTrackGesture)
    {
        mShouldTrackGesture = false;
        mPanGestureRecognizer.enabled = false;
        mTreatTouchesAsMouse = true;
    } else {
        if( mTreatTouchesAsMouse)
        {
            mTreatTouchesAsMouse = false;
            [self showStatus:@"touches passed as touches"];
        }
        else
        {
            mTreatTouchesAsMouse = true;
            [self showStatus:@"mouse input emulated by touch "];
        }
    }
}

#pragma mark UIResponder touch methods

-(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (!mTreatTouchesAsMouse)
    {
        [self sendTouchEvent: XSTX_TOUCH_DOWN withTouches: touches withEvent:
event];
        return;
    }

    UITouch *touch = [touches anyObject];
    [self sendMouseEvent:[touch locationInView:self.view] flags:RI_MOUSE_BUTTON_1_DOWN];
}

-(void) touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
    //When passing touches as touches we need to separate out touch_cancelled
    if (!mTreatTouchesAsMouse)
    {
        [self sendTouchEvent: XSTX_TOUCH_CANCELLED withTouches: touches
withEvent: event];
        return;
    }
    //Handle touchesCancelled exactly like touchesEnded
    [self touchesEnded:touches withEvent:event];
}

-(void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
```



```
{
    if (!mTreatTouchesAsMouse)
    {
        [self sendTouchEvent: XSTX_TOUCH_UP withTouches: touches withEvent:
event];
        return;
    }

    UITouch *touch = [touches anyObject];
    [self sendMouseEvent:[touch locationInView:self.view] flags:RI_MOUSE_BUTTON_1_UP];
    //After the mouseUp move the mouse way offscreen to prevent any unwanted
mouse hover effects
    [self sendMouseEvent:CGPointMake(-10000, -10000) flags:0];

    // clear status
    [self showStatus:@""];
}

-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (!mTreatTouchesAsMouse)
    {
        [self sendTouchEvent: XSTX_TOUCH_MOVE withTouches: touches withEvent:
event];
        return;
    }

    UITouch *touch = [touches anyObject];
    [self sendMouseEvent:[touch locationInView:self.view] flags:0];
}

#pragma mark XStx touch event handling

-(void) sendTouchEvent: (XStxTouchType) theType withTouches: (NSSet *) touches
withEvent: (UIEvent *) event
{
    if (mWidthScale==0)
    {
        return;
    }

    // here we are just going to emulate the mouse as a touch event,
    // just to demonstrate how to use it
    XStxInputEvent inputEvent;
    inputEvent.mTimestampUs = CACurrentMediaTime() * 1000000.0;
    inputEvent.mUserId = 0;
    inputEvent.mType = XSTX_INPUT_EVENT_TYPE_TOUCH;

    //How many touches we are sending
    inputEvent.mInfo.mTouch.mPointerCount = [touches count];
    //Array to hold the pointer data
    inputEvent.mInfo.mTouch.mPointers = new XStxPointer[[touches count]];

    int currPointer = 0;
    for (UITouch *currTouch in touches) {
        //Convert the touch location from the view to the server coordinates
```

```

        CGPoint touchLoc = [currTouch locationInView:self.view];
        //Make sure touchLoc is within the viewport area
        touchLoc = CGPointMake(MIN(MAX(touchLoc.x - mVideoRect.origin.x, 0),
mVideoRect.size.width), MIN(MAX(touchLoc.y - mVideoRect.origin.y, 0), mVideoR
ect.size.height));

        float scaledX = (touchLoc.x * mWidthScale);
        float scaledY = (touchLoc.y * mHeightScale);
        //Use the address of the touchObject as the pointerID
        inputEvent.mInfo.mTouch.mPointers[currPointer].mPointerId =
(uint64_t)currTouch;
        //Set the X & Y
        inputEvent.mInfo.mTouch.mPointers[currPointer].mX = scaledX;
        inputEvent.mInfo.mTouch.mPointers[currPointer].mY = scaledY;
        //No pressure support so always pass 1.0
        inputEvent.mInfo.mTouch.mPointers[currPointer].mPressure = 1.0f;

        inputEvent.mInfo.mTouch.mPointers[currPointer].mTouchType = theType;
        currPointer++;
    }
    inputEvent.mSize = sizeof(inputEvent);
    inputEvent.mDeviceId = 0;

    //Send the touches to the server
    [self sendInput:inputEvent];

    //Clean up the mPointers array
    delete [] inputEvent.mInfo.mTouch.mPointers;
}

// fill out XStxInputEvent struct emulating win32 RAWMOUSE
-(void) sendMouseEvent:(CGPoint ) xy flags:(uint32_t)flags
{
    if (mWidthScale==0)
    {
        return;
    }

    //Make sure xy is within the viewport area
    xy = CGPointMake(MIN(MAX(xy.x - mVideoRect.origin.x, 0), mVideoRect.size.width),
MIN(MAX(xy.y - mVideoRect.origin.y, 0), mVideoRect.size.height));

    int scaledX = (xy.x * mWidthScale);
    int scaledY = (xy.y * mHeightScale);

    XStxInputEvent xstxevent = { 0 };
    xstxevent.mTimestampUs = CACurrentMediaTime() * 1000000.0;
    xstxevent.mType = XSTX_INPUT_EVENT_TYPE_MOUSE;
    xstxevent.mInfo.mMouse.mLastX = scaledX;
    xstxevent.mInfo.mMouse.mLastY = scaledY;
    xstxevent.mInfo.mMouse.mButtonFlags = flags ;
    xstxevent.mInfo.mMouse.mFlags = 1; //absolute, 0 would be relative
    xstxevent.mInfo.mMouse.mButtons = 0;// not needed
    [self sendInput:xstxevent];
    NSString *outputString = [NSString stringWithFormat:@"sending mouse event with
coordinates:x:%i y:%i",scaledX,scaledY];
    [self showStatus:outputString];
}

```

Terminate Your Client

The client ends the session by calling the `XStxClientLibraryRecycle` function and then free the resources.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `recycleXStxCliet` and `dealloc` functions of `AppStreamSampleClientViewController.m`.

```
-(void) recycleXStxCliet
{
    XStxClientLibraryRecycle(mClientLibraryHandle);
}

// clean up
-(void) dealloc {
    [[NSNotificationCenter defaultCenter] removeObserver:self];
    [self recycleXStxCliet];
}
```

Build a Client for OS X

The Amazon AppStream SDK provides client libraries that you build into your client to receive and decode video and audio from the application and send device inputs and raw inputs to the application.

The following object classes are provided by the client libraries in the Amazon AppStream SDK.

Class	Description
<code>XStxClientLibrary</code>	The top level object of the client libraries. Your client uses this object to ensure that it is calling into the version of the libraries it was compiled against and to create the <code>XStxCliet</code> object.
<code>XStxCliet</code>	This object manages client sessions and sends events to your application when Amazon AppStream assigns your application a client session or terminates a client session.

Your client implements and populates the following structures in order to receive session and application events.

Interface	Description
<code>XStxIClientListener</code>	Receives callbacks from a client.
<code>XStxIVideoDecoder</code>	Receives and decodes the video frame.
<code>XStxIVideoRenderer</code>	Renders the video frame.
<code>XStxIRawVideoFrameAllocator</code>	Receives and recycles the video frame.
<code>XStxIAudioDecoder</code>	Receives and decodes the audio frame.
<code>XStxIAudioRenderer</code>	Renders the audio frame.

Interface	Description
XStxIRawAudioFrameAllocator	Receives and recycles the audio frame.

Lifecycle of a Client for OS X

Your client communicates with the application on Amazon AppStream through libraries of the Amazon AppStream SDK.

The lifecycle of a client is as follows:

1. **Create the `XStxClientLibraryHandle` object by calling the `XStxClientLibraryCreate` function to return a `clientLibraryHandle`.** The object interacts with the client side portion of the API through the `clientLibraryHandle`.
2. **Create the `XStxClient` object by calling the `XStxClientCreate` function to return a `XStxClientHandle` client handle.** The object manages the session between the client and the application.
3. **Instantiate and populate the `XStxIClientListener` structure.** This object responds to the callback functions from the application when a session is established between the client and the application. After populating the listener structure, call the `XStxClientSetListener` function with the client handle and a pointer to the `XStxIClientListener` structure.
4. **Initialize a video module to get, decode, and render the video frames.** To initialize this module, the client has to instantiate the following structures:
 - `XStxIVideoDecoder` Gets and decodes the video frame.
 - `XStxIVideoRenderer` Renders the video frame.
 - `XStxIRawVideoFrameAllocator` Gets and recycles the video frame.

The Amazon AppStream SDK contains `VideoPipeline.cpp` which shows how to create the `VideoDecoder` and `VideoRenderer` objects. The source code is in `<install_dir>\example_src\client\src\apple\osx`. The implementation of `VideoDecoder` in the sample client attempts to use `VDA Decoder` for hardware accelerated H.264 video decoding. If the hardware decoder is not available, then the client will use the software decoder `FFmpeg`.

5. **Initialize an audio module to get, decode, and render the audio frames.** To initialize this module, the client has to instantiate the following structures:
 - `XStxIAudioDecoder` Gets and decodes the audio frame.
 - `XStxIAudioRenderer` Renders the audio frame.
 - `XStxIRawAudioFrameAllocator` Gets and recycles the audio frame.

The Amazon AppStream SDK contains `AudioPipeline.cpp` which shows how to create the `AudioDecoder` and `AudioRenderer` objects. The source code is in `<install_dir>\example_src\client\src\apple\shared`.

6. **Configure a session between the client and application.** Call the `XStxClientSetEntitlementUrl` with the endpoint of the entitlement service to configure this session.
7. **Call the `XStxClientStart` function to start the session.** Once the session is established, the `XStxClient` object will invoke the `XStxIClientListenerFcnClientReady` function on the `XStxIClientListener` structure. You can then call `XStxClientLibraryRecycle` to recycle the client library handle.
8. **The client sends user inputs to the application**
9. **The client closes the session.**

Build the Sample Client for OS X

The Amazon AppStream SDK contains source code that you can build into a sample client file. To learn how to build the source code, see the `\doc` directory.

Create Your Client

To create a client, your client must create an `XStxClientLibraryHandle` object. This is the top level object that you will use to interact with the libraries you will use to connect to sessions, render the audio and video, and send inputs to the application.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `XStxModule::init()` function of `XStxModule.cpp`.

Note

We strongly recommend using the provided source code and pre-compiled clients rather than creating your own implementation based on these excerpts.

```
XStxResult XStxModule::init()
{
    XStxResult createResult = XSTX_RESULT_OK;
    if ((createResult = XStxClientLibraryCreate(XSTX_CLIENT_API_VERSION_MAJOR,
                                                XSTX_CLIENT_API_VERSION_MINOR,
                                                &mClientLibraryHandle)) != XSTX_RESULT_OK)
    {
        const char *name; const char *desc;
        XStxResultGetInfo(createResult, &name, &desc);
        LOGE("XStxClientLibraryCreate failed with: %s", name);
        return createResult;
    }

    mVideoRenderer = newVideoRenderer();

    return XSTX_RESULT_OK;
}
```

The function creates a client library handle (`mClientLibraryHandle`) that the client will use to create a client object. The sample client creates the client object by instantiating an object from a user-defined class.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `XStxModule::connect` function of `XStxModule.cpp`.

```
if ((result = XStxClientCreate(mClientLibraryHandle, &mClientHandle))
    != XSTX_RESULT_OK)
{
    LOGE("Failed to create client.");
    const char *name; const char *desc;
    XStxResultGetInfo(result, &name, &desc);
    LOGE("XStxClientCreate failed with: %s", name);
    return result;
}
```

The object needs a structure to respond to the callback functions from the application. The client creates and populates a `XStxIClientListener` structure to respond to the callback functions.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `XStxModule::connect` function of `XStxModule.cpp`.

```
...
memset(&mStxListener, 0, sizeof(mStxListener));
...

mStxListener.mClientReadyFcn = &::clientReady;
mStxListener.mClientReadyCtx = this;
mStxListener.mClientStoppedFcn = &::clientStopped;
mStxListener.mClientStoppedCtx = this;
mStxListener.mMessageReceivedFcn = &::messageReceived;
mStxListener.mMessageReceivedCtx = this;
mStxListener.mStreamQualityMetricsReceivedFcn = &::clientQoS;
mStxListener.mStreamQualityMetricsReceivedCtx = this;
mStxListener.mSize = sizeof(mStxListener);
```

After populating the structure, call the `XStxClietSetListener` function to configure a listener that responds to the callback functions.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `XStxModule::connect` function of `XStxModule.cpp`.

```
if ((result = XStxClietSetListener(mClientHandle, &mStxListener))
    != XSTX_RESULT_OK)
{
    LOGE("Failed to set listener");
    const char *name; const char *desc;
    XStxResultGetInfo(result, &name, &desc);
    LOGE("XStxClietSetListener failed with: %s", name);
    platformErrorMessage(true, desc);
    return result;
}
```

The client uses structures to get, render, and decode video and audio frames.. The next step is to create and populate these structures. The sample client instantiates a class to create and populate the structures.

To handle a video frame, the client create and populate the following structures defined in `VideoModule.h`:

- **XStxIVideoDecoder**. Used to get and decode a video frame.
- **XStxIVideoRenderer**. Used to render a video frame.
- **XStxIRawVideoFrameAllocator**. Used to get and recycle a video frame.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates populating the structures for video. The excerpt is from the `XStxModule::connect` function of `XStxModule.cpp`.

```
// initialize video module
if (!mVideoModule.initialize(mClientHandle, *mVideoRenderer))
{
```

```
    LOGE("Failed to create video decoder/renderer");  
    return XSTX_RESULT_NOT_INITIALIZED_PROPERLY;  
}
```

To handle an audio frame, the client create and populate the following structures defined in `AudioModule.h`:

- **XStxIAudioDecoder**. Used to get and decode an audio frame.
- **XStxIAudioRenderer**. Used to render an audio frame.
- **XStxIRawAudioFrameAllocator**. Used to get and recycle an audio frame.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates populating the structures for audio. The excerpt is from `XStxModule::connect` function of `XStxModule.cpp`.

```
// initialize audio module  
if (!mAudioModule.initialize(mClientHandle))  
{  
    LOGE("Failed to create audio decoder/renderer");  
    return XSTX_RESULT_NOT_INITIALIZED_PROPERLY;  
}
```

The client is now ready to configure and start a session with the application. The client configures a session by calling the `XStxClietSetEntitlementUrl` function with the endpoint of the entitlement service.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `XStxModule::connect` function of `XStxModule.cpp`.

```
if ((result = XStxClietSetEntitlementUrl(mClientHandle, address.c_str()))  
    != XSTX_RESULT_OK)  
{  
    const char *name; const char *desc;  
    XStxResultGetInfo(result, &name, &desc);  
    LOGE("XStxClietSetEntitlementUrl failed with: %s", name);  
    return result;  
}
```

The client starts a session by calling a function.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `XStxModule::connect` function of `XStxModule.cpp`.

```
// non-blocking!  
if ((result = XStxClietStart(mClientHandle)) != XSTX_RESULT_OK)  
{  
    LOGE("Failed to start client.");  
    const char *name; const char *desc;  
    XStxResultGetInfo(result, &name, &desc);  
    LOGE("XStxClietStart failed with: %s", name);  
    return result;  
}
```

Choose a Color Subsampling Rate

Amazon AppStream streams the video at the YUV420 color subsampling rate to client applications. If your streaming application, client application and device support YUV444, you can set your client application to display the video at YUV444. The example client application in Amazon AppStream SDK is configured to accept and render a stream in the YUV444 color subsampling rate.

When the client application connects to the streaming application, Amazon AppStream compares the color subsampling options available on the client application with the options advertised on the streaming application. Amazon AppStream then selects the highest color resolution supported by both the client application and streaming application. Amazon AppStream then calls the `XStxIClientListener2FcnSetConfiguration` callback function that the client application supplied and passes the structure with the `XStxChromaSampling` setting.

The following excerpt from `H264ToYUV.cpp` illustrates this step. This file is in the `<SDK_dir>\example_src\client\src\apple\osx` directory.

```
bool H264ToYuv::isChromaSamplingSupported(XStxChromaSampling chromaSampling)
{
    return chromaSampling == XSTX_CHROMA_SAMPLING_YUV420
        || chromaSampling == XSTX_CHROMA_SAMPLING_YUV444;
}
```

The following excerpt from `VideoPipeline.cpp` illustrates this step. This file is in the `<SDK_dir>\example_src\client\src\apple\osx` directory.

```
decoderCapabilities->mProfile = XSTX_H264_PROFILE_HIGH444;
```

Send Your Client Inputs to the Application

The client can send keyboard, mouse, touch, or raw inputs to the application. The client sends an input by filling out the `XStxInputEvent` structure that describes the input and then calling a function that passes the structure to the application.

Keyboard

Before the client can fill out the `XStxInputEvent` structure, the keyboard input must first be converted to a Windows keyboard input from an OS X keyboard input. In the `processKeyEvent` function is the call to `getConvertedKeyUsingCharacter` which converts the keyboard input to a Windows keyboard input.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates the function that converts the OS X keyboard input to a Windows keyboard input. The excerpt is from the `getConvertedKeyUsingCharacter` function in `AppStreamSampleClientWindowController.m`.

```
(int) getConvertedKeyUsingCharacter: (NSEvent *) theEvent withModifierMask:
(uint16_t &) modifierKeyMask withModifierKeyIgnoreMask: (uint16_t &) modifier
KeysIgnoreMask
{
    //Get the characters from the key being pressed
    NSString *theKeyString = [theEvent charactersIgnoringModifiers];
```



```
    if ([theKeyString length] <= 0) {
        //Reject dead keys
        return -1;
    }

    //Get the characters as a char
    uint16_t theChar = [theKeyString characterAtIndex:0];

    //First check the generic key mapping, this should convert any non-system
    // specific keys
    bool didMapKey = getVirtualKeyUsingChar(theChar, modifierKeyMask, modifier
    KeysIgnoreMask);

    if (!didMapKey) {
        //The generic key mapping didn't map the key so check the platform
        specific ones
        didMapKey = appleGetVirtualKeyUsingChar(theChar, modifierKeyMask, modi
        fierKeysIgnoreMask);
    }

    if (!didMapKey) {
        //Neither the generic nor the system specific mapping could remap the
        key
        NSLog(@"\n===== Unmapped KEY! =====");
        unichar upperKeyChar = [[theKeyString uppercaseString] characterAtIn
        dex:0];
        NSString *modKeyString = [theEvent characters];

        NSLog(@"Key: %hu String: %@ Mod: %@", upperKeyChar, theKeyString, mod
        KeyString);

        return -1;
    }

    return theChar;
}
```

The following excerpt from the sample client in the Amazon AppStream SDK illustrates the function that processes the keyboard action from the client. The excerpt is from the `processKeyEvent` function in `AppStreamSampleClientWindowController.m`.

```
(void) processKeyEvent: (NSEvent *) theEvent
{
    //Depending on your application you might want to block repeat keys
    //    if ([theEvent isARepet]) {
    //        //Ignore repeats
    //        return;
    //    }

    //Setup a couple bitmasks for the command keys
    uint16_t modifierKeyMask = 0;
    uint16_t modifierKeysIgnoreMask = 0;
    int newKey = [self getConvertedKeyUsingCharacter:theEvent withModifier
    Mask:modifierKeyMask withModifierKeyIgnoreMask:modifierKeysIgnoreMask];

    if (newKey < 0)
```

```
{
    //Not a valid key mapping just return
    return;
}

BOOL needAlt = (modifierKeyMask & STX_ALT_KEY_MASK) == STX_ALT_KEY_MASK;
BOOL needControl = (modifierKeyMask & STX_CONTROL_KEY_MASK) == STX_CON
TROL_KEY_MASK;
BOOL needShift = (modifierKeyMask & STX_SHIFT_KEY_MASK) ==
STX_SHIFT_KEY_MASK;
BOOL needWin = (modifierKeyMask & STX_WINDOWS_KEY_MASK) == STX_WIN
DOWS_KEY_MASK;

//Check the status of the command keys
if ((modifierKeysIgnoreMask & STX_SHIFT_KEY_MASK) != STX_SHIFT_KEY_MASK) {

    if (needShift && !shiftIsDown) {
        //Need shift pressed but it isn't
        [[AppStreamClient sharedClient] sendKeyDown:VK_SHIFT];
    } else if (!needShift && shiftIsDown)
    {
        //Need shift released but it is pressed
        [[AppStreamClient sharedClient] sendKeyUp:VK_SHIFT];
    }
}

if ((modifierKeysIgnoreMask & STX_ALT_KEY_MASK) != STX_ALT_KEY_MASK) {
    if (needAlt && !altIsDown) {
        [[AppStreamClient sharedClient] sendKeyDown:VK_MENU];
    } else if (!needAlt && altIsDown)
    {
        [[AppStreamClient sharedClient] sendKeyUp:VK_MENU];
    }
}

if ((modifierKeysIgnoreMask & STX_CONTROL_KEY_MASK) != STX_CONTROL_KEY_MASK)
{
    if (needControl && !controlIsDown) {
        [[AppStreamClient sharedClient] sendKeyDown:VK_CONTROL];
    } else if (!needControl && controlIsDown)
    {
        [[AppStreamClient sharedClient] sendKeyUp:VK_CONTROL];
    }
}

if ((modifierKeysIgnoreMask & STX_WINDOWS_KEY_MASK) != STX_WINDOWS_KEY_MASK)
{
    if (needWin && !windowsIsDown) {
        [[AppStreamClient sharedClient] sendKeyDown:VK_LWIN];
    } else if (!needWin && windowsIsDown)
    {
        [[AppStreamClient sharedClient] sendKeyUp:VK_LWIN];
    }
}

//Now send the actual keydown event
if (theEvent.type == NSKeyDown) {
    [[AppStreamClient sharedClient] sendKeyDown:newKey];
}
```

```
} else
{
    //Not keyDown so must be keyUp
    [[AppStreamClient sharedClient] sendKeyUp:newKey];
}

//Reset the command keys state
if ((modifierKeysIgnoreMask & STX_SHIFT_KEY_MASK) != STX_SHIFT_KEY_MASK) {

    if (needShift && !shiftIsDown) {
        //Need shift pressed but it isn't
        [[AppStreamClient sharedClient] sendKeyUp:VK_SHIFT];
    } else if (!needShift && shiftIsDown)
    {
        //Need shift released but it is pressed
        [[AppStreamClient sharedClient] sendKeyDown:VK_SHIFT];
    }
}

if ((modifierKeysIgnoreMask & STX_ALT_KEY_MASK) != STX_ALT_KEY_MASK) {
    if (needAlt && !altIsDown) {
        [[AppStreamClient sharedClient] sendKeyUp:VK_MENU];
    } else if (!needAlt && altIsDown)
    {
        [[AppStreamClient sharedClient] sendKeyDown:VK_MENU];
    }
}

if ((modifierKeysIgnoreMask & STX_CONTROL_KEY_MASK) != STX_CONTROL_KEY_MASK)
{
    if (needControl && !controlIsDown) {
        [[AppStreamClient sharedClient] sendKeyUp:VK_CONTROL];
    } else if (!needControl && controlIsDown)
    {
        [[AppStreamClient sharedClient] sendKeyDown:VK_CONTROL];
    }
}

if ((modifierKeysIgnoreMask & STX_WINDOWS_KEY_MASK) != STX_WINDOWS_KEY_MASK)
{
    if (needWin && !windowsIsDown) {
        [[AppStreamClient sharedClient] sendKeyUp:VK_LWIN];
    } else if (!needWin && windowsIsDown)
    {
        [[AppStreamClient sharedClient] sendKeyDown:VK_LWIN];
    }
}
}
```

After converting the keyboard input, the client populates the `XStxInputEvent` structure. The structure is then sent to AppStream.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates a keyboard action that populates the structures and then sending that structure to the application. The excerpt is from the `XStxModule::keyPress` function in `XStxModule.cpp`.

```
void XStxModule::keyPress(int key, bool down)
{
    XStxInputEvent xstxevent = { 0 };
    xstxevent.mTimestampUs = mud::TimeVal::mono().toMilliseconds();
    xstxevent.mType = XSTX_INPUT_EVENT_TYPE_KEYBOARD;
    xstxevent.mInfo.mKeyboard.mVirtualKey = key;
    xstxevent.mInfo.mKeyboard.mIsKeyDown = down;
    xstxevent.mSize = sizeof(XStsxInputEvent);

    sendInput(xstxevent);
}
```

Mouse

The following excerpt from the sample client in the Amazon AppStream SDK illustrates how the mouse actions are sent to the structure. The excerpt is from `AppStreamSampleClientWindowControll.m`.

```
(void) mouseDown:(NSEvent *)theEvent
{
    NSPoint theLoc = [theEvent locationInWindow];

    theLoc.y = _glView.bounds.size.height - theLoc.y;

    [[AppStreamClient sharedClient] sendMouseEvent:theLoc flags:RI_MOUSE_BUTTON_1_DOWN];
}

(void) mouseUp:(NSEvent *)theEvent
{
    NSPoint theLoc = [theEvent locationInWindow];

    theLoc.y = _glView.bounds.size.height - theLoc.y;

    [[AppStreamClient sharedClient] sendMouseEvent:theLoc flags:RI_MOUSE_BUTTON_1_UP];
}

(void) mouseDragged:(NSEvent *)theEvent
{
    NSPoint theLoc = [theEvent locationInWindow];

    theLoc.y = _glView.bounds.size.height - theLoc.y;

    [[AppStreamClient sharedClient] sendMouseEvent:theLoc flags:0];
}

(void) rightMouseDown:(NSEvent *)theEvent
{
    NSPoint theLoc = [theEvent locationInWindow];

    theLoc.y = _glView.bounds.size.height - theLoc.y;

    [[AppStreamClient sharedClient] sendMouseEvent:theLoc flags:RI_MOUSE_BUTTON_2_DOWN];
}
```

```
(void) rightMouseUp:(NSEvent *)theEvent
{
    NSPoint theLoc = [theEvent locationInWindow];

    theLoc.y = _glView.bounds.size.height - theLoc.y;

    [[AppStreamClient sharedClient] sendMouseEvent:theLoc flags:RI_MOUSE_BUTTON_2_UP];
}

(void) rightMouseDragged:(NSEvent *)theEvent
{
    NSPoint theLoc = [theEvent locationInWindow];

    theLoc.y = _glView.bounds.size.height - theLoc.y;

    [[AppStreamClient sharedClient] sendMouseEvent:theLoc flags:0];
}

(void) mouseMoved:(NSEvent *)theEvent
{
    NSPoint theLoc = [theEvent locationInWindow];

    theLoc.y = _glView.bounds.size.height - theLoc.y;

    [[AppStreamClient sharedClient] sendMouseEvent:theLoc flags:0];
}
```

The client then populates the `XStxInputEvent` structure. The structure is then sent to AppStream.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates a mouse action that populates the structures and then sends that structure to the application. The excerpt is from the `XStxModule::mouseEvent` function in `XStxModule.cpp`.

```
void XStxModule::mouseEvent(int x, int y, uint32_t flags)
{
    // no video renderer? Return!
    if (!mVideoRenderer)
    {
        return;
    }

    int xOff, yOff;
    float scale;
    mVideoRenderer->getScaleAndOffset(scale, xOff, yOff);

    // no scale yet; ignore!
    if (scale == 0) return;

    XStxInputEvent xstxevent = { 0 };
    xstxevent.mTimestampUs = mud::TimeVal::mono().toMilliseconds();
    xstxevent.mType = XSTX_INPUT_EVENT_TYPE_MOUSE;
    xstxevent.mInfo.mMouse.mButtonFlags = flags;
    xstxevent.mInfo.mMouse.mFlags = 1; //absolute
}
```

```
static int lastX = 0, lastY = 0;

if (flags & CET_MOUSE_WHEEL)
{
    LOGV("Mouse wheel data: %d", x);
    // mouse wheel data goes in mButtonData.
    xstxevent.mInfo.mMouse.mButtonData = x;
    xstxevent.mInfo.mMouse.mLastX = lastX;
    xstxevent.mInfo.mMouse.mLastY = lastY;
}
else
{
    lastX = xstxevent.mInfo.mMouse.mLastX = (int)((x + xOff) * scale);
    lastY = xstxevent.mInfo.mMouse.mLastY = (int)((y + yOff) * scale);
    // LOGV("Mouse data: x:%d,y:%d", lastX, lastY);
}

sendInput(xstxevent);
}
```

Terminate Your Client

The client can end the session with the application in the following ways:

- End the session and then confirm the session ended.
- End the session without regard as to when the session ends.

To end the session and then confirm the session ended, first call `XStxCliientStop`. This is a non-blocking function call that immediately returns a result. If the call was successful, then call `XStxCliientWait` to wait until the session actually ends. When the `XStxCliientWait` call is successful, call `XStxCliientRecycle` to recycle the client handle.

To end the session without regard as to when the session ends, call `XStxCliientRecycle`. The session then ends without further interaction from the client.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `XStxModule::recycle` function in `XStxModule.cpp`.

```
XStxResult XStxModule::recycle()
{
    XStxResult result = XStxCliientRecycle(mClientHandle);
    if (result != XSTX_RESULT_OK)
        return result;
    else
        return XStxCliientLibraryRecycle(mClientLibraryHandle);
}
```

Build a Client for Windows

The Amazon AppStream SDK provides client libraries that you build into your client to receive and decode video and audio from the application and send device inputs and raw inputs to the application.

The following object classes are provided by the client libraries in the Amazon AppStream SDK.

Class	Description
XStxClientLibrary	The top level object of the client libraries. Your client uses this object to ensure that it is calling into the version of the libraries it was compiled against and to create the XStxClient object.
XStxClient	This object manages client sessions and sends events to your application when Amazon AppStream assigns your application a client session or terminates a client session.

Your client implements and populate the following structures in order to receive session and application events.

Interface	Description
XStxIClientListener	Receives callbacks from a client.
XStxIVideoDecoder	Receives and decodes the video frame.
XStxIVideoRenderer	Renders the video frame.
XStxIRawVideoFrameAllocator	Receives and recycles the video frame.
XStxIAudioDecoder	Receives and decodes the audio frame.
XStxIAudioRenderer	Renders the audio frame.
XStxIRawAudioFrameAllocator	Receives and recycles the audio frame.

The Amazon AppStream SDK includes source code so that you can build into a sample client. The documentation is in the `\doc` directory.

Lifecycle of a Client for Windows

Your client communicates with the application on Amazon AppStream through libraries of the Amazon AppStream SDK.

The lifecycle of a client is as follows:

1. **Create the `xStxClientLibraryHandle` object by calling the `xStxClientLibraryCreate` function to return a `clientLibraryHandle`.** The object interacts with the client side portion of the API through the `clientLibraryHandle`.
2. **Create the `xStxClient` object by calling the `xStxClientCreate` function to return a `xStxClientHandle` client handle.** The object manages the session between the client and the application.
3. **Instantiate and populate the `xStxIClientListener` structure.** This object responds to the callback functions from the application when a session is established between the client and the application. After populating the listener structure, call the `xStxClientSetListener` function with the client handle and a pointer to the `xStxIClientListener` structure.
4. **Initialize a video module to get, decode, and render the video frames.** To initialize this module, the client has to instantiate the following structures:
 - `XStxIVideoDecoder` Gets and decodes the video frame.
 - `XStxIVideoRenderer` Renders the video frame.
 - `XStxIRawVideoFrameAllocator` Gets and recycles the video frame.

5. **Initialize an audio module to get, decode, and render the audio frames.** To initialize this module, the client has to instantiate the following structures:
 - `XStxIAudioDecoder` Gets and decodes the audio frame.
 - `XStxIAudioRenderer` Renders the audio frame.
 - `XStxIRawAudioFrameAllocator` Gets and recycles the audio frame.
6. **Configure a session between the client and application.** Call the `XStxCliSetEntitlementUrl` function with the endpoint of the entitlement service to configure this session.
7. **Call the `xStxCliStart` function to start the session.** Once the session is established, the `XStxCli` object will invoke the `XStxICliListenerFncClientReady` function on the `XStxICliListener` structure. You can then call `xStxCliLibraryRecycle` to recycle the client library handle.
8. **The client sends user inputs to the application**
9. **The client closes the session.**

Build the Sample Client for Windows

The Amazon AppStream SDK contains source code that you can build into a sample client file. To learn how to build the source code, see the `\doc` directory.

Create Your Client

To create a client, your client must create an `XStxCliLibraryHandle` object. This is the top level object that you will use to interact with the libraries you will use to connect to sessions, render the audio and video, and send inputs to the application.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `main` function of `XStxExampleClient.cpp`.

```
/** instantiate client library handle */
XStxCliLibraryHandle clientLibraryHandle;
XStxResult createResult = XSTX_RESULT_OK;

if ((createResult = XStxCliLibraryCreate(
XSTX_CLIENT_API_VERSION_MAJOR,
XSTX_CLIENT_API_VERSION_MINOR,
&clientLibraryHandle)) != XSTX_RESULT_OK)
{
    const char * name; const char * desc;
    XStxResultGetInfo(createResult, &name, &desc);
    printf("XStxCliLibraryCreate failed with: %s\n", name);
    return 1;
}
```

The function creates a client library handle (`clientLibraryHandle`) that the client will use to create a client object. The sample client creates the client object by instantiating an object from a user-defined class.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `WindowListener` class definition of `XStxExampleClient.cpp`.


```
// setup xstx client and start making the connection
if ((result = XStxClientCreate(mClientLibraryHandle, &mClientHandle))
    != XSTX_RESULT_OK)
{
    mRenderWindow->setErrorText("Failed to create client");
    const char * name; const char * desc;
    XStxResultGetInfo(result, &name, &desc);
    printf("XStxClientCreate failed with: %s\n", name);
    return;
}
```

The object needs a structure to respond to the callback functions from the application. The client creates and populates a `XStxIClientListener` structure to respond to the callback functions.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `readyForConnection` function of `XStxExampleClient.cpp`.

```
...
XStxIClientListener mStxListener;
...
mStxListener.mClientReadyFcn = &clientReady;
mStxListener.mClientReadyCtx = this;
mStxListener.mClientStoppedFcn = &clientStopped;
mStxListener.mClientStoppedCtx = this;
mStxListener.mMessageReceivedFcn = &messageReceived;
mStxListener.mMessageReceivedCtx = this;
```

After populating the structure, call the `XStxClientSetListener` function to configure a listener that responds to the callback functions.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `readyForConnection` function of `XStxExampleClient.cpp`.

```
if ((result = XStxClientSetListener(mClientHandle, &mStxListener))
    != XSTX_RESULT_OK)
{
    mRenderWindow->setErrorText("Failed to set listener");
    const char * name; const char * desc;
    XStxResultGetInfo(result, &name, &desc);
    printf("XStxClientSetListener failed with: %s\n", name);
    return;
}
```

The client uses structures to get, render, and decode video and audio frames.. The next step is to create and populate these structures. The sample client instantiates a class to create and populate the structures.

To handle a video frame, the client create and populate the following structures defined in `VideoModule.h`:

- **XStxIVideoDecoder**. Used to get and decode a video frame.
- **XStxIVideoRenderer**. Used to render a video frame.
- **XStxIRawVideoFrameAllocator**. Used to get and recycle a video frame.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates populating the structures for video. The excerpt is from `VideoModule.cpp`.

```
/**
 * Initialize video module
 * @param[in] clientHandle handle to XStx client
 * @param[in] rw renderer
 */
bool VideoModule::initialize(XStxClientHandle mClientHandle,
    RenderWindow & rw)
{
    // initialize frame allocator
    mStxFrameAllocator.mInitFcn = &allocatorInit;
    mStxFrameAllocator.mInitCtx = this;
    mStxFrameAllocator.mGetVideoFrameBufferFcn = &allocatorGetFrame;
    mStxFrameAllocator.mGetVideoFrameBufferCtx = this;
    mStxFrameAllocator.mRecycleVideoFrameBufferFcn = &allocatorRecycleFrame;
    mStxFrameAllocator.mRecycleVideoFrameBufferCtx = this;
    mStxFrameAllocator.mSize = sizeof(mStxFrameAllocator);
    if (XStxClientSetVideoFrameAllocator(mClientHandle, &mStxFrameAllocator)
        != XSTX_RESULT_OK)
    {
        printf("Failed to SetVideoFrameAllocator.\n");
        return false;
    }

    // initialize renderer
    mRenderer = &rw;

    mStxRenderer.mRenderVideoFrameFcn = &renderFrame;
    mStxRenderer.mRenderVideoFrameCtx = mRenderer;
    mStxRenderer.mSetMaxResolutionFcn = &rendererMaxResolution;
    mStxRenderer.mSetMaxResolutionCtx = mRenderer;
    mStxRenderer.mSize = sizeof(mStxRenderer);
    if (XStxClientSetVideoRenderer(mClientHandle, &mStxRenderer)
        != XSTX_RESULT_OK)
    {
        printf("Failed to set SetVideoRenderer.\n");
        return false;
    }

    // initialize decoder
    mDecoder = new H264ToYuv();
    if (!mDecoder)
    {
        return false;
    }
    mStxDecoder.mGetCapabilitiesCtx = mDecoder;
    mStxDecoder.mGetCapabilitiesFcn = &videoDecoderGetCapabilities;
    mStxDecoder.mDecodeVideoFrameFcn = &decodeFrame;
    mStxDecoder.mDecodeVideoFrameCtx = mDecoder;
    mStxDecoder.mStartFcn = &videoDecoderStart;
    mStxDecoder.mStartCtx = mDecoder;
    mStxDecoder.mSize = sizeof(mStxDecoder);
    if (XStxClientSetVideoDecoder(mClientHandle, &mStxDecoder)
        != XSTX_RESULT_OK)
    {

```

```
    printf("Failed to set SetVideoDecoder.\n");  
    return false;  
}
```

To handle an audio frame, the client create and populate the following structures defined in `AudioModule.h`:

- **XStxIAudioDecoder**. Used to get and decode an audio frame.
- **XStxIAudioRenderer**. Used to render an audio frame.
- **XStxIRawAudioFrameAllocator**. Used to get and recycle an audio frame.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates populating the structures for audio. The excerpt is from `AudioModule.cpp`.

```
/**  
 * Initialize audio module  
 * @param[in] clientHandle handle to XStx client  
 */  
bool AudioModule::initialize(XStxCliHandle clientHandle)  
{  
    // initialize frame allocator  
    mStxFrameAllocator.mInitFcn = &allocatorInit;  
    mStxFrameAllocator.mInitCtx = this;  
    mStxFrameAllocator.mGetAudioFrameBufferFcn = &allocatorGetFrame;  
    mStxFrameAllocator.mGetAudioFrameBufferCtx = this;  
    mStxFrameAllocator.mRecycleAudioFrameBufferFcn = &allocatorRecycleFrame;  
    mStxFrameAllocator.mRecycleAudioFrameBufferCtx = this;  
    mStxFrameAllocator.mSize = sizeof(mStxFrameAllocator);  
    if (XStxCliSetAudioFrameAllocator(clientHandle, &mStxFrameAllocator)  
        != XSTX_RESULT_OK)  
    {  
        printf("Failed to SetAudioFrameAllocator\n");  
        return false;  
    }  
  
    // initialize decoder  
    mDecoder = new (std::nothrow) OpusToPcm();  
    if (!mDecoder)  
    {  
        printf("Failed to create Decoder\n");  
        return false;  
    }  
    mStxDecoder.mDecodeAudioFrameFcn = &decoderDecodeFrame;  
    mStxDecoder.mDecodeAudioFrameCtx = mDecoder;  
    mStxDecoder.mStartFcn = &decoderStart;  
    mStxDecoder.mStartCtx = mDecoder;  
    mStxDecoder.mSize = sizeof(mStxDecoder);  
    if (XStxCliSetAudioDecoder(clientHandle, &mStxDecoder)  
        != XSTX_RESULT_OK)  
    {  
        printf("Failed to SetAudioDecoder\n");  
        return false;  
    }  
}
```

```
// initialize renderer
mRenderer = new (std::nothrow) AudioRenderer(mFramePool,
    clientHandle);
if (!mRenderer)
{
    return false;
}

// Set XStx callbacks and contexts on for the XStxIAudioRenderer struct
mStxRenderer.mStartFcn = &start;
mStxRenderer.mStartCtx = mRenderer;
mStxRenderer.mSize = sizeof(mStxRenderer);

// set the XStxIAudioRenderer to be used with the given clientHandle
if (XStxClietSetAudioRenderer(clientHandle, &mStxRenderer)
    != XSTX_RESULT_OK)
{
    printf("Failed to SetAudioRenderer\n");
    return false;
}
return true;
}
```

The client is now ready to configure and start a session with the application. The client configures a session by calling the `XStxClietSetEntitlementUrl` function with the endpoint of the entitlement service.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `readyForConnection` function in `XStxExampleClient.cpp`.

```
if ((result = XStxClietSetEntitlementUrl(
    mClientHandle, mEntitlementUrl.c_str())) != XSTX_RESULT_OK)
{
    mRenderWindow->setErrorText("Failed to set entitlement URL");
    const char * name; const char * desc;
    XStxResultGetInfo(result, &name, &desc);
    printf("XStxClietSetEntitlementUrl failed with: %s\n", name);
    return;
}
```

The client starts a session by calling a function.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `readyForConnection` function in `XStxExampleClient.cpp`.

```
// non-blocking!
if ((result = XStxClietStart(mClientHandle)) != XSTX_RESULT_OK)
{
    mRenderWindow->setErrorText("Failed to start client");
    const char * name; const char * desc;
    XStxResultGetInfo(result, &name, &desc);
    printf("XStxClietStart failed with: %s\n", name);
    return;
}
```

```
}  
  
// success !  
mRenderWindow->setErrorText("Starting STX");
```

Choose a Color Subsampling Rate

Amazon AppStream streams the video at the YUV420 color subsampling rate to client applications. If your streaming application, client application and device support YUV444, you can set your client application to display the video at YUV444. The example client application in Amazon AppStream SDK is configured to accept and render a stream in the YUV444 color subsampling rate.

When the client application connects to the streaming application, Amazon AppStream compares the color subsampling options available on the client application with the options advertised on the streaming application. Amazon AppStream then selects the highest color resolution supported by both the client application and streaming application. Amazon AppStream then calls the `XStxIClientListener2FcnSetConfiguration` callback function that the client application supplied and passes the structure with the `XStxChromaSampling` setting.

Send Your Client Inputs to the Application

The client can send keyboard, mouse, touch, or raw inputs to the application. The client sends an input by filling out the `XStxInputEvent` structure that describes the input and then calling a function that passes the structure to the application.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates a keyboard action that populates the structures and then sending that structure to the application. The excerpt is from the `keyChange` function in `DirectXRenderWindow.cpp`.

```
/** Handles WM_KEYDOWN & WM_KEYUP messages. */  
void keyChange(WPARAM wParam, LPARAM lParam, bool isKeyDown)  
{  
    if (!mListener)  
    {  
        return;  
    }  
    XStxInputEvent inputEvent;  
    inputEvent.mTimestampUs = mud::TimeVal::mono().toMicroSeconds();  
    inputEvent.mDeviceId = 0;  
    inputEvent.mUserId = 0;  
    inputEvent.mType = INPUT_EVENT_TYPE_KEYBOARD;  
  
    inputEvent.mInfo.mKeyboard.mIsKeyDown = isKeyDown;  
    inputEvent.mInfo.mKeyboard.mVirtualKey = wParam;  
    inputEvent.mInfo.mKeyboard.mScanCode = lParam;  
  
    inputEvent.mSize = sizeof(inputEvent);  
  
    mListener->sendInput(inputEvent);  
}
```

The following excerpt from the sample client in the Amazon AppStream SDK illustrates a mouse action that populates the structures and then sending that structure to the application. The excerpt is from the `mouseChange` function in `DirectXRenderWindow.cpp`.

```
void mouseChange(WPARAM wParam, LPARAM lParam, bool isDown, bool isLeft)
{
    if (!mListener)
    {
        return;
    }
    XStxInputEvent inputEvent;
    inputEvent.mTimestampUs = mud::TimeVal::mono().toMicroSeconds();
    inputEvent.mDeviceId = 0;
    inputEvent.mUserId = 0;
    inputEvent.mType = INPUT_EVENT_TYPE_MOUSE;

    inputEvent.mInfo.mMouse.mFlags = MOUSE_MOVE_ABSOLUTE;

    int32_t leftchange = 0;
    int32_t rightChange = 0;
    if (isLeft) {
        leftchange = isDown ? RI_MOUSE_LEFT_BUTTON_DOWN : RI_MOUSE_LEFT_BUTTON_UP;
    } else {
        rightChange = isDown ? RI_MOUSE_RIGHT_BUTTON_DOWN : RI_MOUSE_RIGHT_BUTTON_UP;
    }

    inputEvent.mInfo.mMouse.mButtonFlags = leftchange | rightChange;
    inputEvent.mInfo.mMouse.mButtons = 0; // not needed
    inputEvent.mInfo.mMouse.mButtonData = 0; // not sending wheel data
    rescaleMouseInput(lParam, inputEvent.mInfo.mMouse.mLastX,
        inputEvent.mInfo.mMouse.mLastY);

    inputEvent.mSize = sizeof(inputEvent);
    mListener->sendInput(inputEvent);
}
```

If the client is a different size from the application, the client rescales the mouse position to adjust for the different sizes.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `rescaleMouseInput` function in `DirectXRenderWindow.cpp`.

```
/** Rescales absolute mouse position. */
void rescaleMouseInput(LPARAM lParam, int32_t & outX, int32_t & outY)
{
    // if we send absolute input, we need to rescale it
    RECT desktopRect;
    GetWindowRect(mWindow, &desktopRect);
    float rescaleHeight = mLastSetHeight /
        (float) (desktopRect.bottom - desktopRect.top - mWindowBorderAdjustHeight);
    float rescaleWidth = mLastSetWidth /
        (float) (desktopRect.right - desktopRect.left - mWindowBorderAdjustWidth);

    outX = GET_X_LPARAM(lParam) * rescaleWidth + 0.5f; // add 0.5 for rounding
```

```
    outY = GET_Y_LPARAM(lParam) * rescaleHeight + 0.5f;
}
```

Touch input is sent as a mouse action.

Terminate Your Client

The client can end the session with the application in the following ways:

- End the session and then confirm the session ended.
- End the session without regard as to when the session ends.

To end the session and then confirm the session ended, first call the `XStxClientStop` function. This is a non-blocking function call that immediately returns a result. If the call was successful, then call the `XStxClientWait` function to wait until the session actually ends. When the `XStxClientWait` call is successful, call the `XStxClientRecycle` function to recycle the client handle.

To end the session without regard as to when the session ends, call the `XStxClientRecycle` function. The session then ends without further interaction from the client.

The following excerpt from the sample client in the Amazon AppStream SDK illustrates this step. The excerpt is from the `windowClosed` function in `XStxExampleClient.cpp`.

```
/** Clean up */
void windowClosed()
{
    // the window was closed, do what I need
    if (!mClientHandle)
    {
        return;
    }
    // ensure the audio stopped pulling
    mAudioModule.stop();

    XStxClientRecycle(mClientHandle);
    mClientHandle = NULL;
}
```

Codec and Open Source Licensing

What audio and video formats does Amazon AppStream use?

AppStream utilizes the H.264/AVC video format for encoding streamed video, and the open-source Opus audio format for encoding streamed audio.

How does my client decode video from Amazon AppStream?

You may need to include an H.264/AVC decoder with your client application. In our experience, the built-in decoder included with Windows 7 provides sufficient functionality, but the decoders provided with the iOS and Android platforms do not. The sample client we provide for developer education and testing uses FFmpeg, an LGPL2.1-licensed open-source decoder. We have also found that CoreAVC, a proprietary

decoder available from CoreCodec, Inc., is a good option as well. You are responsible for complying with the license terms which apply to the decoder you use in your client application.

Does use of Amazon AppStream require proprietary licenses?

You may have to obtain a license from MPEG-LA for use of a video decoder with your client application and transmission of video of your hosted application, depending on the nature of your application and the video content transmitted to your end users. You should reach out to the provider of your decoder and/or MPEG-LA for guidance.

Content that you or any End User run on, cause to interface with, or upload to Amazon AppStream is Your Content. You are responsible for determining whether your content or technology used in your Amazon AppStream hosted applications, entitlement service, or clients require any additional licensing.

Are there any open source considerations?

Amazon AppStream components, including client binary components, utilize certain open source packages, which are noted in the `notice.txt` files provided with the Amazon AppStream SDK. When you distribute your Amazon AppStream client, you should include the attributions for the Amazon AppStream client binary component, which are listed in the applicable `notice.txt` file for the particular client in the Amazon AppStream SDK. In addition, Amazon AppStream sample code provided in the SDK also includes several open source packages, which carry their own licenses; if you use this sample code you need to comply with the applicable license terms.

Deploy Your Streaming Application to Amazon AppStream

Adding your application to Amazon AppStream gives the service the information it needs to deploy your application on Amazon AppStream hosts.

Prerequisites

Before you deploy your application on Amazon AppStream, you must do the following:

1. Add streaming functionality to your application. For more information, see [Build an Amazon AppStream Application \(p. 51\)](#).
2. Create an application installer for your application. For more information see [Build an Application Installer \(p. 73\)](#)
3. Upload the application installer to Amazon S3. For more information, see [Upload the Application Installer to Amazon Simple Storage Service \(p. 123\)](#).
4. Generate a pre-signed URL for the application installer stored in Amazon S3. For more information, see [Create a Pre-signed URL \(p. 124\)](#).
5. Build and deploy an entitlement service. For more information, see [Build an Entitlement Service \(p. 73\)](#).

For a walkthrough of how to host an application on Amazon AppStream using sample components, see [Get Started \(p. 12\)](#).

Upload the Application Installer to Amazon Simple Storage Service

Amazon AppStream starts the application installer to install the application from an Amazon Simple Storage Service (Amazon S3) bucket.

To create a bucket and upload the application installer, follow these steps:

To Create a Bucket

1. In the [Amazon S3 console](#), click **Create Bucket**.
2. In the **Region** box, expand the **Region Selector** on the navigation bar, and select **US East (N. Virginia)**.
3. In the **Create a Bucket—Select a Bucket Name and Region** dialog box, type a name in the **Bucket Name** box.

The bucket name you choose must be unique across all existing bucket names in Amazon S3. Your bucket name must be between 3 and 63 characters long, composed of lowercase letters and numbers.

Important

After you create a bucket, you cannot change the bucket name.

4. When the settings are as you want them, click **Create**.

When Amazon S3 successfully creates your bucket, the console displays the bucket name in the **All Buckets** pane. This is the bucket where you'll upload the installer application.

To Upload the Application Installer to Amazon S3

1. In the Amazon S3 console, select the bucket that you previously created.
2. Click **Actions** and then click **Upload**.
3. In the **Upload—Select Files** dialog box, click **Add Files**.
4. In the **File Upload** dialog box, select the application installer that you downloaded and saved to your local drive.
5. Click **Open**.
6. Click **Start Upload**.

Create a Pre-signed URL

To upload your the application installer to Amazon S3, you need to create a pre-signed URL that points to the file. The pre-signed URL must be generated using the HTTPS protocol. When you add your application installer to Amazon AppStream, you'll type the pre-signed URL into the online form that describes your application. Be sure to set the expiration time of the URL far enough in the future that you have time to deploy your application using the application installer.

To generate a pre-signed URL, you can use the AWS Toolkit for Visual Studio, which installs the AWS Explorer into Visual Studio. You will use the AWS Explorer to generate the pre-signed URL. To use the AWS Explorer, you will need the access ID key and the secret key of the AWS account that has access to the Amazon S3 bucket that contains your application installer.

To install the AWS Toolkit for Visual Studio

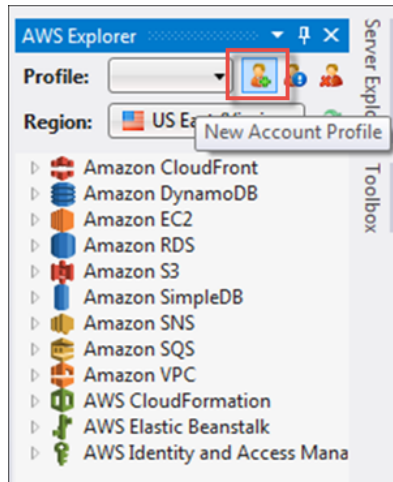
1. Go to [AWS Toolkit for Visual Studio](#) and click **AWS Toolkit for Visual Studio**.
2. Run the installation wizard, which is packaged as an `.msi`.
 - If your browser asks whether to save or run the `.msi`, select **Run**.
 - If your browser automatically saves the `.msi` file to your system, navigate to the download directory and use Windows Explorer to launch the `.msi`.

The MSI file name depends on the version, but it will look something like `AWSToolsAndSDKForNet_sdk-2.0.13.2-ps-2.0.13.2-tk-1.6.5.4.msi`.

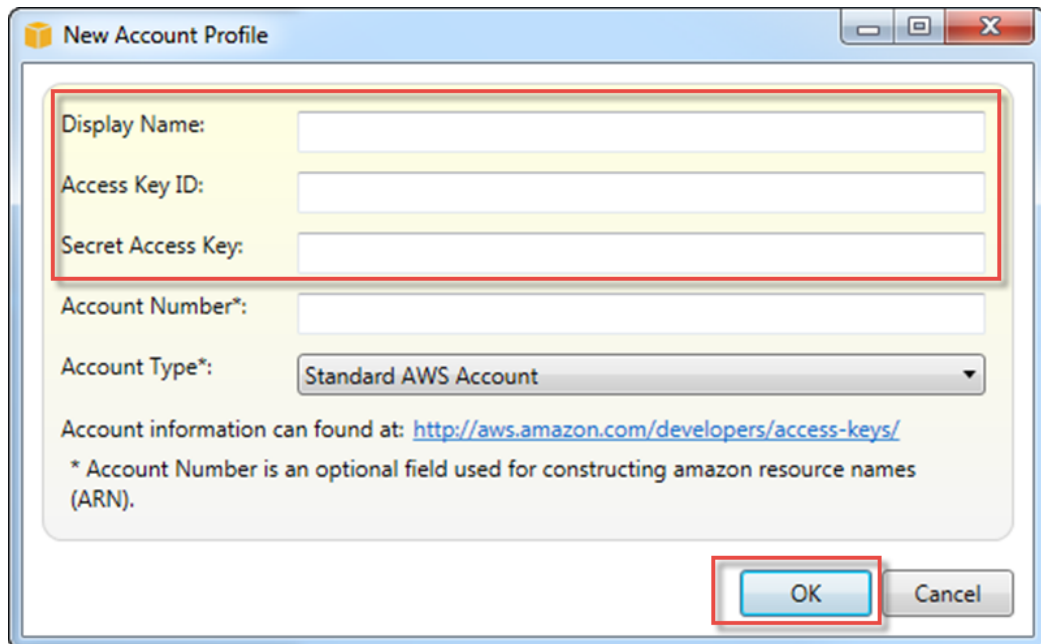
3. Follow the installation wizard's instructions to install the toolkit.

To add a profile to the Toolkit for Visual Studio

1. In Visual Studio, open **AWS Explorer** by clicking the View menu and selecting **AWS Explorer**.
2. Click the "New Account Profile" icon to the right of the **Profile** list.



3. In the **New Account Profile** dialog box, do the following:
 - a. For **Display Name**, type a name to identify the profile.
 - b. For **Access Key ID**, enter the access key ID of the user with access to the Amazon S3 bucket with your application installer.
 - c. For **Secret Access Key**, enter the secret key of the user with access to the Amazon S3 bucket with your application installer.
 - d. Click **OK**.



4. Close the **AWS Explorer**.

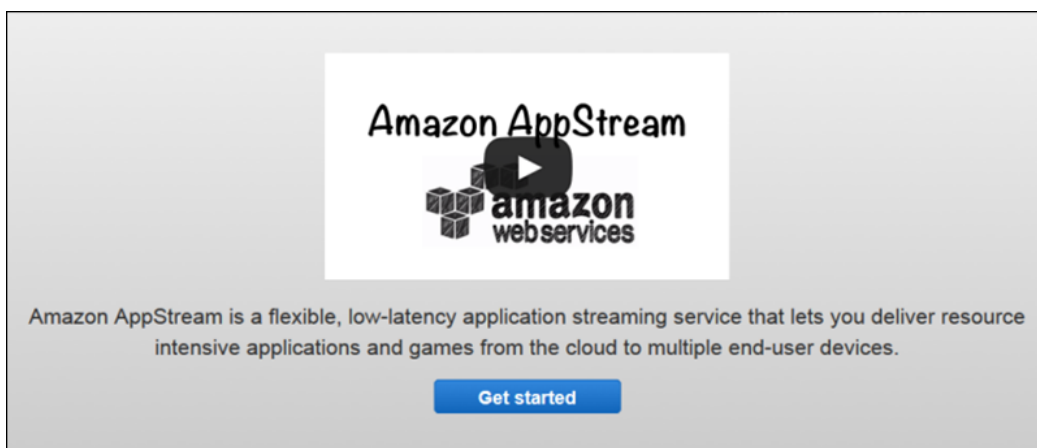
To generate a pre-signed URL

1. Open the **AWS Explorer**.
2. In **AWS Explorer**, expand the **Amazon S3** node until you see your application installer, right-click the application installer, and then select **Create Pre-Signed URL**.
3. In the **Create Pre-Signed URL** dialog box, set an expiration date and time for the URL or go to the next step if you want to use the default setting which is one hour from the current time.
4. Click the **Generate** button.
5. Click **Copy** to copy the pre-signed URL to the clipboard. Save this URL to file.

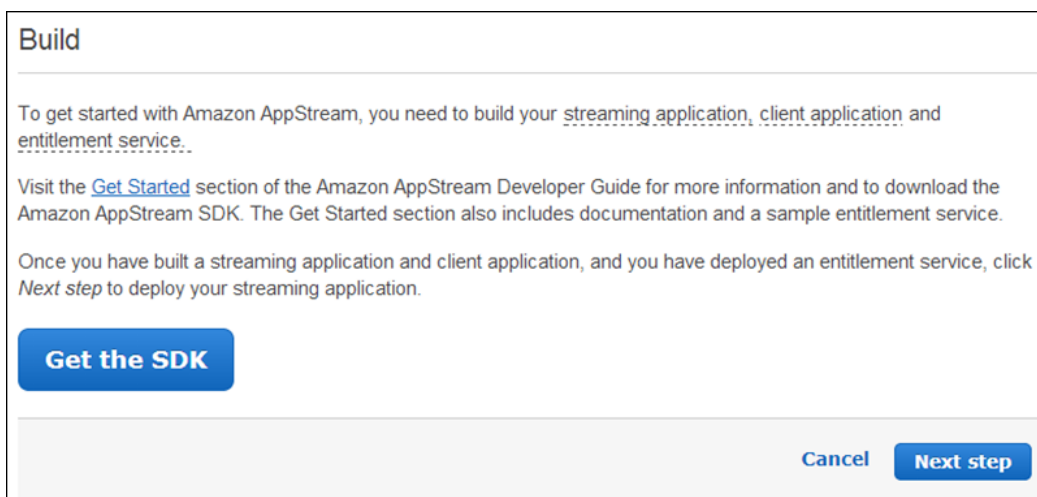
Deploy Your Streaming Application

To deploy your streaming application, do the following:

1. In the [Amazon AppStream console](#), click **Get Started**.



2. On the **Build** page, click **Next step**.



3. On the **Configure** page, do the following and then click **Next step**:

- In **Streaming application name**, enter a name for the application. This name is used only in the Amazon AppStream console, it is not displayed to users. The name can be up to 64 characters long, and can include letters, numbers, spaces, and punctuation.
- (Optional) In **Streaming application description**, enter a description for the application. This description is used only in the Amazon AppStream console, it is not displayed to users. The name can be up to 512 characters long, and can include letters, numbers, spaces, and punctuation.
- In **Pre-signed S3 URL to Installer**, enter a pre-signed URL that points to the location of a file stored in Amazon S3 that installs your application. The installer must be a single file that installs your application as well as all of its resources and dependencies. Ensure that the pre-signed URL has not expired. Instead of a pre-signed URL, you can enter the URL to a public S3 bucket. Use a public bucket only for testing sample code; putting content in a public S3 bucket makes it accessible to all users and is *strongly discouraged* for production applications.
- (Optional) In **Installer Parameters**, enter any command-line parameters that Amazon AppStream includes when it runs the installer.
- In **Path to launcher**, enter the local path and filename that Amazon AppStream calls to launch your application on a host.
- (Optional) In **Launcher parameters**, enter command-line parameters that Amazon AppStream includes when it launches your application on an Amazon AppStream host.

Configure

To deploy your streaming application, please provide us with the following information:

Streaming application name *	<input type="text" value="My Application"/>	Enter up to 64 letters, numbers, punctuation marks, and spaces to create a friendly name that identifies this streaming application within the console. This is only used in the console - your customers will not see it.
Streaming application description	<input type="text" value="Codenames, special version notes, etc."/>	Enter up to 512 letters, numbers, punctuation marks, and spaces to describe your streaming application or add notes. This is only used in the console - your customers will not see it.
Pre-signed S3 URL to installer *	<input type="text" value="https://mybucket.s3.amazonaws.co"/>	Enter a pre-signed S3 URL to an executable (*.exe) installer file you have stored in S3. The installer file should work without user interaction and contain all of the resources and dependencies for your application bundled together inside of it. Learn how to make a pre-signed S3 URL Learn more about installer requirements
Installer parameters	<input type="text" value="/silent"/>	Enter any command-line parameters required to install your streaming application.
Path to launcher *	<input type="text" value="C:\myapp\launch.exe"/>	Enter an absolute path to the Windows executable (*.exe) file that will launch your streaming application after it is installed, for example, C:\myapp\launch.exe.
Launch parameters	<input type="text" value="/locale en-US"/>	Enter any command-line parameters required to launch your streaming application.

Required *

CancelPreviousNext step

4. You can enable logging on the **Log** page in **Application logs** or click **Next step** to continue deploying your streaming application.

If you want to enable logging on your streaming application, select **Yes, save application logs**. After you select the check box, a text box appears below so that you can optionally specify another path to a directory that contains other log files or a path to a log file that will be added to the other log files. You can specify more than one path or file by filling out the other text box that appears after you fill in the text box.

After specifying the paths and log files, click **Next step**.

Log

You can turn logging on and off for your application to help you diagnose potential problems after deployment.

You will also be charged for S3 usage for any log or crash data stored in your S3 account. [See S3 pricing.](#)

Application Logs

☐ Yes, save application logs
Amazon AppStream collects the following types of logs:

- A file called `launch.txt.failed` with the output if Amazon AppStream fails to start the streaming application.
- Minidump files (`.dmp`)
- Standard error stream (`launch.txt.stderr`)
- Standard output stream (`launch.txt.stdout`)
- Utilization metrics (`utilization.csv`) including CPU, memory and disk usage
- You can also enter absolute paths to additional files you want to collect (use `*` or `?` as wild card characters to collect multiple files -- for instance, `C:\app\logs*`)

Save Location

Select an S3 bucket in which to save logs and crash data. Amazon AppStream can create and use a default bucket or use one that already exists. We configure an IAM role to access the default bucket, and offer details for adding access to an existing bucket.
[View the policy used for the default bucket](#)
[Learn how to configure an existing bucket](#)

Required *

Cancel

Previous

Next step

The custom log files and paths must meet the following requirements:

- Log files must be located in the `C:\` directory.
- Log paths must be fully qualified paths. Do not use relative paths. You can use wildcards.
- Log paths must be less than the path length limits. See [Naming Files, Paths, and Namespaces](#).
- Log file names cannot contain Windows reserved characters. See [Naming Conventions](#).
- Your application must have read access to the paths.

Note

The total size of logs in the `.zip` file is subject to the Amazon S3 key size limits.

The following log files are included in the `.zip` file:

- A file called `launch.txt.failed` with the output if Amazon AppStream fails to start the streaming application.
- [Minidump files](#) (`.dmp`)
- Standard error stream (`launch.txt.stderr`)
- Standard output stream (`launch.txt.stdout`)
- Utilization metrics (`utilization.csv`) which is a file that provides operational metrics. The first row contains column headings and the following rows are comma-separated values:

- **UTC Time**—Time in Universal Coordinated Time (UTC) when the row was created.
- **Processor (n) %**—The percentage of use for each CPU core starting with Core 0.
- **Memory in Use (Bytes)**—Memory that was used in bytes.
- **Memory Available (Bytes)**—Memory that was available in bytes.
- **Disk Read (Bytes)**—Disk read speed in bytes per second.
- **Disk Write (Bytes)**—Disk write speed in bytes per second.
- **GPU Utilization %**—The percentage of use for the GPU.
- **GPU Memory Utilization %**—The percentage of GPU memory that was used.
- **GPU Memory Total (MB)**—The total GPU memory available in the megabytes (MB).
- **GPU Memory Used (MB)**—The amount of GPU memory used in MB.
- **GPU Memory Free (MB)**—The amount of GPU memory available in MB.
- **GPU Temperature (C)**—The temperature of the GPU in degrees centigrade.
- **GPU Power Draw (W)**—The number of watts the GPU is using.
- **GPU Power Limit (W)**—The maximum number of watts required by the GPU.

The following is an example of the metrics in this utilization log.

```
UTC Time,Processor(0) %,Processor(1) %,Processor(2) %,Processor(3) %,Memory
In Use (Bytes),Memory Available (Bytes),Disk Write (Bytes/sec),Disk Read
(Bytes/sec),GPU Utilization %,GPU Memory Utilization %,GPU Memory Total
(MB),GPU Memory Used (MB),GPU Memory Free (MB),GPU Temperature (C),GPU
Power Draw (W),GPU Power Limit (W)
2014/05/15
11:17:39,0.92,3.14,6.58,1.88,2724400196.27,14014595549.87,614818.20,103422.13,0,
1, 4095, 4063, 32, 33, 18.44, 125.00
2014/05/15
11:18:40,0.95,1.15,0.32,1.24,2745420390.40,14010372983.47,8689.31,99415.41,0,
1, 4095, 4063, 32, 33, 18.26, 125.00
2014/05/15
11:19:41,0.36,0.67,0.20,1.30,2734278997.33,14029225574.40,6994.11,18322.61,0,
1, 4095, 4063, 32, 33, 18.44, 125.00
2014/05/15
11:20:42,0.08,0.69,0.10,0.97,2724477064.53,14033759027.20,27142.17,14072.01,0,
1, 4095, 4063, 32, 33, 18.54, 125.00
2014/05/15
11:21:43,0.18,0.27,0.10,1.42,2719389627.73,14038990438.40,40446.51,203.51,0,
1, 4095, 4063, 32, 33, 18.53, 125.00
2014/05/15
11:22:44,0.13,0.44,0.12,0.87,2720666555.73,14039756800.00,43574.68,1696.60,0,
1, 4095, 4063, 32, 32, 18.67, 125.00
2014/05/15
11:23:45,0.18,0.44,0.24,0.92,2718720887.47,14040533674.67,44824.72,271.31,0,
1, 4095, 4063, 32, 32, 18.43, 125.00
2014/05/15
11:24:46,0.30,0.50,0.14,1.73,2707566796.80,14053706888.53,72334.31,209776.90,0,
1, 4095, 4063, 32, 32, 18.21, 125.00
2014/05/15
11:25:46,0.34,0.44,0.14,1.52,2654716040.53,14106968064.00,43219.86,1479.45,0,
1, 4095, 4063, 32, 33, 18.41, 125.00
2014/05/15
11:26:47,0.05,0.88,0.16,1.08,2655204010.67,14107344554.67,43875.24,0.00,0,
1, 4095, 4063, 32, 32, 18.31, 125.00
2014/05/15
```



```
11:27:48,0.38,0.79,0.16,1.25,2653159219.20,14107987626.67,40567.09,0.00,0,
1, 4095, 4063, 32, 33, 18.23, 125.00
2014/05/15
11:28:49,0.17,0.59,0.09,1.32,2650824772.27,14108805666.13,43111.49,0.00,0,
1, 4095, 4063, 32, 33, 18.20, 125.00
2014/05/15
11:29:50,0.16,0.45,0.07,0.76,2651427157.33,14109135803.73,43341.33,0.00,0,
1, 4095, 4063, 32, 33, 18.33, 125.00
2014/05/15
11:30:51,0.05,0.83,0.11,0.92,2650787703.47,14109916501.33,44690.80,0.00,0,
1, 4095, 4063, 32, 33, 18.30, 125.00
2014/05/15
11:31:52,0.13,0.64,0.29,0.79,2651821602.13,14110050781.87,44109.33,0.00,0,
0, 4095, 4063, 32, 33, 18.18, 125.00
2014/05/15
11:32:52,0.14,0.99,0.44,1.34,2655279445.33,14108282675.20,56575.66,2570.57,0,
0, 4095, 4063, 32, 32, 17.78, 125.00
```

The log files are saved to a .zip file which is then uploaded to the bucket. Logs are saved with the filename: `<bucketName>/AppStream/<region>/<Application ID>/yyyy-mm-dd/<log file name>.zip`

5. For **Save Location**, you can choose to have Amazon AppStream create an Amazon S3 bucket. To use a bucket created by Amazon AppStream, click **Create a default bucket**. To use one of your existing Amazon S3 buckets, click the down arrow next to **Create a default bucket** and select one of your buckets.

Note

Your bucket must allow Amazon AppStream to use the `PutObject` method. You can allow this method by using a bucket policy similar to the following:

```
{
  "Version": "2008-10-17",
  "Id": "Policy1396472828471",
  "Statement": [
    {
      "Sid": "Stmt1396472820469",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::990116983315:root" },
      "Action": [ "s3:PutObject" ],
      "Resource": "arn:aws:s3:::your bucket name/*"
    }
  ]
}
```

For more information about modifying the Amazon S3 bucket policy, see [Bucket Policy Examples](#) in the Amazon S3 Developer Guide.

The log files are saved to a .zip file which is then uploaded to the bucket. Logs are saved with the following filename:

`<bucketName>/AppStream/<region>/<Application ID>/yyyy-mm-dd/<log file name>.zip`

6. On the **Review** page, check that the values are correct. If the values are correct, then click **Finish** to add your streaming application to Amazon AppStream. If you need to change a value, click **Previous** until you return to page where you can change the value.

Review

Please take a moment to review the information below and click on any row to edit its contents. When you are ready, click *Finish* to deploy your streaming application. The deployment process can take from 30 minutes to several hours depending on the size of the streaming application.

Check the progress of your deployment at any time by clicking **Applications** in the console and then click **My App 1**.

Describe	
Streaming application name	My App
Streaming application description	
Setup	
Pre-signed S3 URL to installer	<p>https://s3.amazonaws.com/ap</p> <p>i Be sure the installer file works silently and contains all of the resources and dependencies for your streaming application bundled together inside of it.</p>
Command to install the streaming application	<p>XStxDirectXServerInstaller_1.2.exe</p> <p>i This is the command we will run to install your streaming application.</p>
Command to launch the streaming application	<p>C:\app\XStxDirectXServer\XStxDirectXServer.exe</p> <p>i This is the command we will run to launch your streaming application after it has been installed.</p>
Log	
Application Logs	Yes, save application logs
Application log paths	Amazon AppStream will copy application log output from the following paths:
Save location	

[Cancel](#) [Previous](#) [Finish](#)

- Wait while Amazon AppStream prepares your application. This may take 30 minutes or more while Amazon AppStream performs the following tasks:
 - Copies your application installer from your Amazon S3 bucket.
 - Prepares your Amazon AppStream environment.
 - Installs your streaming application on an Amazon AppStream.
 - Creates an Amazon Machine Image (AMI) of the server configuration that includes your installed application.
 - Starts your streaming application.

While Amazon AppStream is deploying your application, it displays the **Application Summary** page, which contains the metadata for your application. The **Application ID** field displays the identifier assigned to your application. Client applications specify this identifier when they call into your entitlement service to connect to your application.

Streaming Application Summary

[Back to all streaming applications](#)

✓

Your streaming application has been deployed and is ready for connections!

✕

To request a session for this streaming application, use the following Application ID in your client:

Your account supports streaming up to 10 simultaneous sessions. Limit increase requests take approximately 10 weeks to fulfill, depending on size. You pay only for what you use and there is no minimum fee. When we receive your request, a member of our team will work with you to best serve your capacity needs. [Learn more about service limits](#) or [request a limit increase](#).

Streaming application name: My App

Application ID: [Select](#)

Status: Running

Utilization: 0 of 2 simultaneous sessions are in use

Streaming application description:

Pre-signed S3 URL of installer: <https://s3.amazonaws.com/ap-...>

Installer parameters:

Launch path: C:\app\XStxDirectXServer\XStxDirectXServer.exe

Launch parameters:

Application logs: Yes, save application logs

Save location:

Edit

Clone

Archive

Manage Your Application

You can use the [Amazon AppStream console](#) to manage your streaming applications in Amazon AppStream.

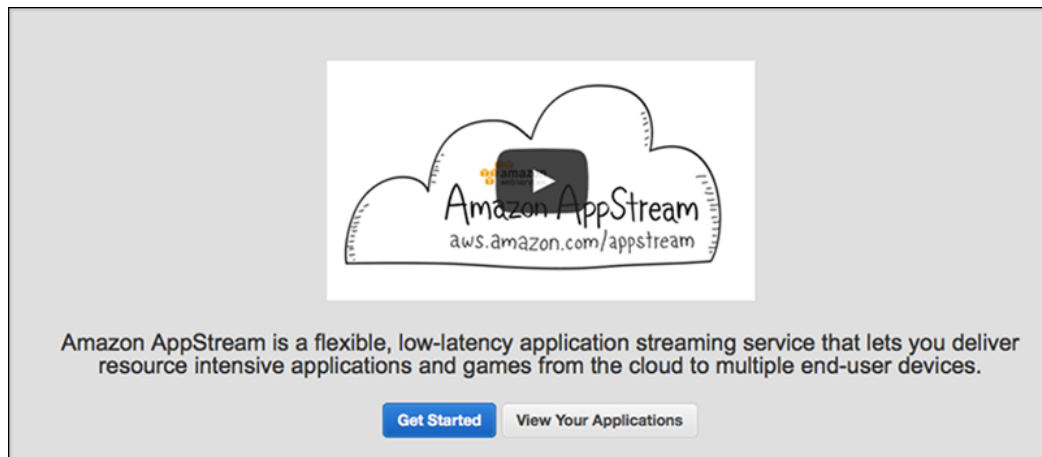
Topics

- [View All Applications](#) (p. 135)
- [View Application Summary](#) (p. 136)
- [Edit an Application](#) (p. 137)
- [Clone an Application](#) (p. 140)
- [Archive an Application](#) (p. 145)
- [Enable Logging on an Application](#) (p. 147)
- [Increase Your Service Limits](#) (p. 152)

View All Applications

To view a list of your applications in Amazon AppStream

- In the [Amazon AppStream console](#), click **View Your Streaming Applications**. This button appears only if you have added applications to Amazon AppStream.



Amazon AppStream displays a list of your applications.

Your account supports streaming up to 10 simultaneous sessions at the same time. [Learn more about service limits](#) or [request a limit increase](#).

[Add a new streaming application](#)

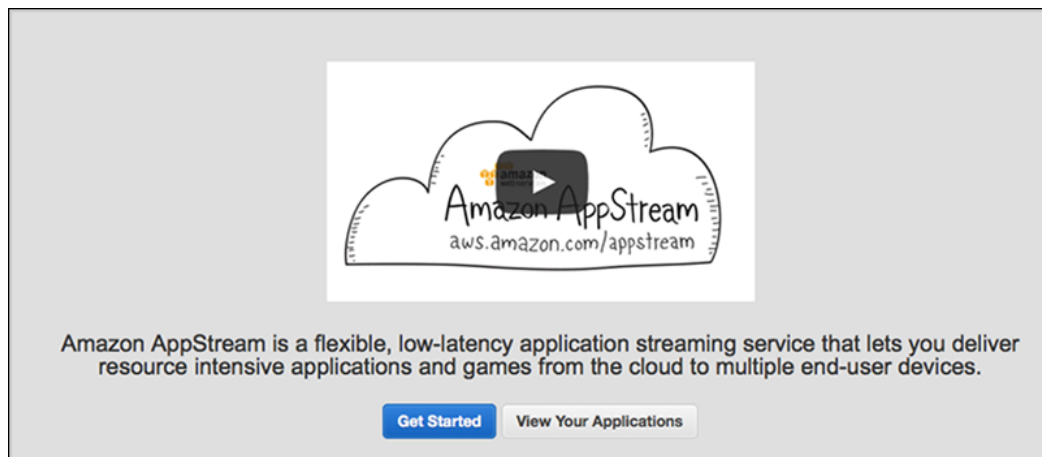
Name	Active sessions	Available sessions	Status
My App	0	2	Running
My App 2	0	2	Running

View Application Summary

The application summary page displays the settings associated with your application, including the application identifier.

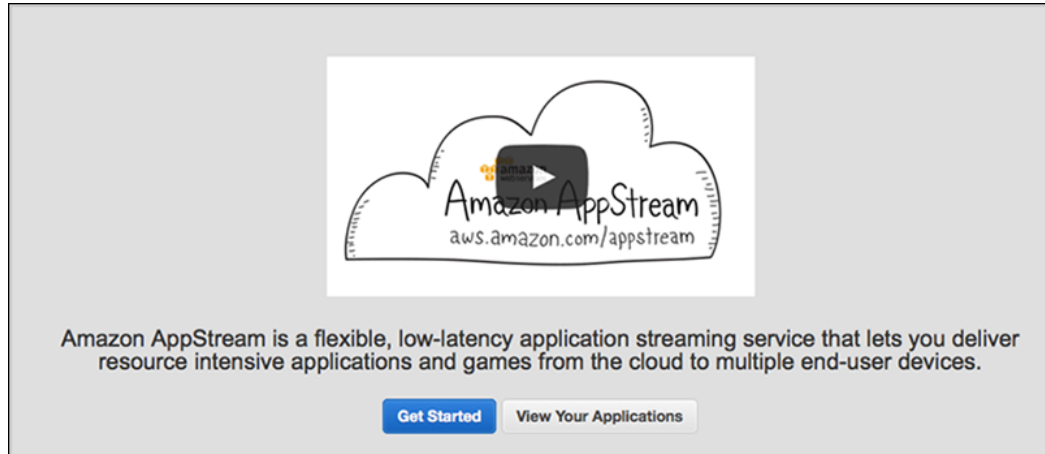
To view detailed information about an application

1. In the [Amazon AppStream console](#), click **View Your Streaming Applications**. This button appears only if you have added applications to Amazon AppStream.



2. In the list of applications, click the name of the streaming application to view.

0.11 0.10 0.09 0.08 0.07 0.06 0.05 0.04 0.03 0.02 0.01 0.00



2. In the list of applications, click the name of the streaming application to view.

Your account supports streaming up to 10 simultaneous sessions at the same time. [Learn more about service limits](#) or [request a limit increase](#).

[Add a new streaming application](#)

Name	Active sessions	Available sessions	Status
My App	0	2	Running
My App 2	0	2	Running

3. On the **Streaming Application Summary** page, click **Edit**.

Streaming Application Summary

Back to all streaming applications

✓

Your streaming application has been deployed and is ready for connections!

✕

To request a session for this streaming application, use the following Application ID in your client:

Your account supports streaming up to 10 simultaneous sessions. Limit increase requests take approximately 10 weeks to fulfill, depending on size. You pay only for what you use and there is no minimum fee. When we receive your request, a member of our team will work with you to best serve your capacity needs. [Learn more about service limits](#) or [request a limit increase](#).

Streaming application name: My App

Application ID: [Select](#)

Status: Running

Utilization: 0 of 2 simultaneous sessions are in use

Streaming application description:

Pre-signed S3 URL of installer: <https://s3.amazonaws.com/ap->

Installer parameters:

Launch path: C:\app\XStxDirectXServer\XStxDirectXServer.exe

Launch parameters:

Application logs: Yes, save application logs

Save location:

Edit

Clone

Archive

4. On the **Edit Streaming Application Details** page, you can enter new values for one or more of the following fields:
- **Streaming application name**, enter a name for the application. This name is visible only in the Amazon AppStream console, it is not displayed to users. The name can be up to 64 characters long, and can include letters, numbers, spaces, and punctuation.
 - (Optional) **Streaming application description**, a description for the application. This description is used only in the Amazon AppStream console, it is not displayed to users. The name can be up to 512 characters long, and can include letters, numbers, spaces, and punctuation.
 - **Path to launcher**, the local path and filename that Amazon AppStream calls to launch your streaming application on a host.
 - (Optional) In **Launcher Parameters**, enter any command-line parameters that Amazon AppStream should include when it launches your application on an Amazon AppStream host.

Click **Save**.

Edit Streaming Application Details

[Back to streaming application summary](#)

Streaming application name *	<input type="text" value="My App"/>	Enter up to 64 letters, numbers, punctuation marks, and spaces to create a friendly name that identifies this streaming application within the console. This is only used in the console - your customers will not see it.
Streaming application description 512 of 512 characters left	<input type="text" value="Codenames, special version notes, etc."/>	Enter up to 512 letters, numbers, punctuation marks, and spaces to describe your streaming application or add notes. This is only used in the console - your customers will not see it.
Path to launcher *	<input type="text" value="C:\app\XStxDirectXServer\XStxDirectX"/>	Enter an absolute path to the Windows executable (*.exe) file that will launch your streaming application after it is installed, for example, C:\myapp\launch.exe.
Launch parameters	<input type="text" value="/locale en-US"/>	Enter any command-line parameters required to launch your streaming application.
Application Logs	<div><input checked="" type="checkbox"/> Yes, save application logs <input type="text" value="C:\MyApp\logs"/></div>	<p>Amazon AppStream collects the following types of logs:</p> <ul style="list-style-type: none">• A file called launch.txt failed with the output if Amazon AppStream fails to start the streaming application.• Minidump files (.dmp)• Standard error stream (launch.txt.stderr)• Standard output stream (launch.txt.stdout)• Utilization metrics (utilization.csv) including CPU, memory and disk usage• You can also enter absolute paths to additional files you want to collect (use * or ? as wild card characters to collect multiple files -- for instance, C:\app\logs*)
Save Location *	<div><div>appstream-us-east-1-111122223333 ▾</div><div>Amazon AppStream is able to write to this S3 bucket.</div></div>	Select an S3 bucket in which to save logs and crash data. Amazon AppStream can create and use a default bucket or use one that already exists. We configure an IAM role to access the default bucket, and offer details for adding access to an existing bucket.

Required *

Cancel

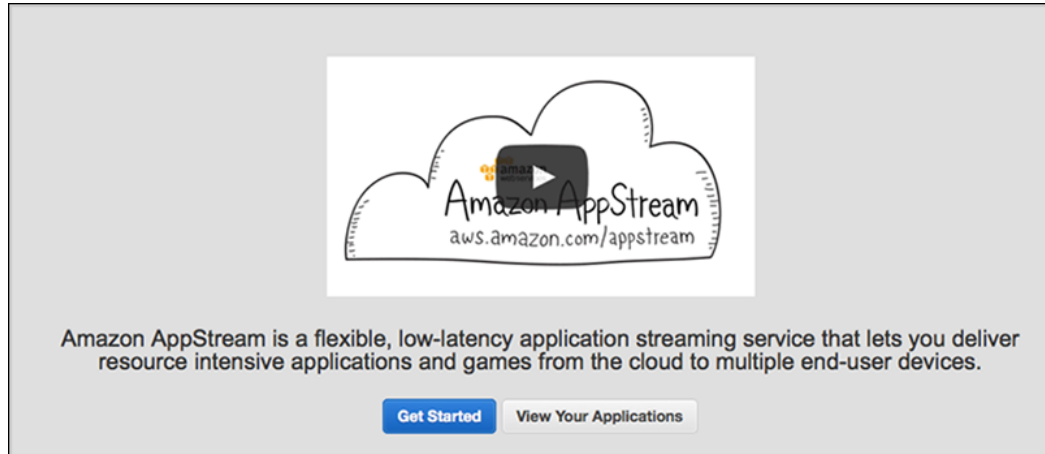
Save

Clone an Application

Cloning an application is a quick way to deploy a copy of your application. You can then change parameters on the cloned copy to make it behave differently from the original. You might, for example, use one set of launcher parameters to launch a version of your application that records logs and uploads the files to Amazon S3 for debugging purposes later and use the same application with a different set of launcher parameters in production.

To clone an application

1. In the [Amazon AppStream console](#), click **View Your Streaming Applications**. This button appears only if you have added applications to Amazon AppStream.



2. In the list of applications, click the name of the application to view.

Your account supports streaming up to 10 simultaneous sessions at the same time. [Learn limits](#) or [request a limit increase](#).

Add a new streaming application

Name	Active sessions	Available sessions	Status
My App	0	1	Running
My App 2	0	0	Running

3. On the **Application Summary** page, click **Clone**.

Streaming Application Summary

Back to all streaming applications

✓

Your streaming application has been deployed and is ready for connections!

✕

To request a session for this streaming application, use the following Application ID in your client:

Your account supports streaming up to 10 simultaneous sessions. Limit increase requests take approximately 10 weeks to fulfill, depending on size. You pay only for what you use and there is no minimum fee. When we receive your request, a member of our team will work with you to best serve your capacity needs. [Learn more about service limits](#) or [request a limit increase](#).

Streaming application name: My App

Application ID: [Select](#)

Status: Running

Utilization: 0 of 2 simultaneous sessions are in use

Streaming application description:

Pre-signed S3 URL of installer: <https://s3.amazonaws.com/ap->

Installer parameters:

Launch path: C:\app\XStxDirectXServer\XStxDirectXServer.exe

Launch parameters:

Application logs: Yes, save application logs

Save location:

Edit

Clone

Archive

- On the **Build** page, click **Next step**.

Build

To get started with Amazon AppStream, you need to build your streaming application, client application and entitlement service.

Visit the [Get Started](#) section of the Amazon AppStream Developer Guide for more information and to download the Amazon AppStream SDK. The Get Started section also includes documentation and a sample entitlement service.

Once you have built a streaming application and client application, and you have deployed an entitlement service, click *Next step* to deploy your streaming application.

Get the SDK

Cancel

Next step

- On the **Configure** page, the application settings are pre-populated with the values from the copied application. You can modify the settings. After you modify the settings, click **Next step**.

142

- In **Streaming application name**, enter a name for the streaming application. This name is visible only in the Amazon AppStream console, it is not displayed to users. The name can be up to 64 characters long, and can include letters, numbers, spaces, and punctuation.
- (Optional) In **Streaming application description**, enter a description for the streaming application. This description is used only in the Amazon AppStream console, it is not displayed to users. The name can be up to 512 characters long, and can include letters, numbers, spaces, and punctuation.
- In **Pre-signed S3 URL to installer**, enter a pre-signed URL that points to the location of a file stored in Amazon S3 that installs your application. The installer must be a single file that installs your application as well as all of its resources and dependencies. Ensure that the pre-signed URL has not expired.
- (Optional) In **Installer parameters**, enter any command-line parameters that Amazon AppStream includes when it runs the installer on an Amazon AppStream host.
- In **Path to launcher**, enter the local path and filename that Amazon AppStream calls to launch your streaming application on a host.
- (Optional) In **Launcher parameters**, enter command-line parameters that Amazon AppStream includes when it launches your streaming application on an Amazon AppStream host.

Configure

To deploy your streaming application, please provide us with the following information

Streaming application name *
58 of 64 characters left

My App

Streaming application description
481 of 512 characters left

A sample streaming application.

Pre-signed S3 URL to installer *
[Learn how to make a pre-signed S3 URL](#)
[Learn more about installer requirements](#)

https://s3.amazonaws.com/appstream-public/

Installer parameters

/silent

Path to launcher *
213 of 259 characters left

C:\app\XStxDirectXServer\XStxDirectXServer

Launch parameters

/locale en-US

Required *

Cancel

Prev


6. Wait while Amazon AppStream prepares your application. This may take 30 minutes or more while Amazon AppStream performs the following tasks:

- Copies your application installer from your Amazon S3 bucket.
- Prepares your Amazon AppStream environment.
- Installs your streaming application on an Amazon AppStream.
- Creates an Amazon Machine Image (AMI) of the server configuration that includes your installed application.
- Starts your streaming application.

While Amazon AppStream is deploying your application, it displays the **Application Summary** page, which contains the metadata for your application. The **Application ID** field displays the identifier assigned to your application. Client applications specify this identifier when they call into your entitlement service to connect to your application.

Streaming Application Summary

[Back to all streaming applications](#)

 **Deploying your application**

Copying your streaming application... done.
Preparing your Amazon AppStream environment... done.
Installing your streaming application..
Creating an Amazon Machine Image (AMI)
Starting your streaming application

Note: This process can take from 30 minutes to several hours to process your streaming application, depending on its size. You can leave this page and the deployment will continue.

Streaming application name: My App

Application ID: [Select](#)

Status: Installing application ...

Utilization: 0 of 0 simultaneous sessions are in use

Streaming application description:

Pre-signed S3 URL of installer: <https://s3.amazonaws.com/ap-...>

Installer parameters:

Launch path: C:\app\XStxDirectXServer\XStxDirectXServer.exe

Launch parameters:

Application logs: Yes, save application logs

Save location:

Edit

Clone

When your deployment finishes, Amazon AppStream displays a message indicating whether the deployment succeeded or failed.

Streaming Application Summary

Back to all streaming applications

✓

Your streaming application has been deployed and is ready for connections!

✕

To request a session for this streaming application, use the following Application ID in your client:

Your account supports streaming up to 10 simultaneous sessions. Limit increase requests take approximately 10 weeks to fulfill, depending on size. You pay only for what you use and there is no minimum fee. When we receive your request, a member of our team will work with you to best serve your capacity needs. [Learn more about service limits](#) or [request a limit increase](#).

Streaming application name: My App

Application ID: [Select](#)

Status: Running

Utilization: 0 of 2 simultaneous sessions are in use

Streaming application description:

Pre-signed S3 URL of installer: <https://s3.amazonaws.com/ap->

Installer parameters:

Launch path: C:\app\XStxDirectXServer\XStxDirectXServer.exe

Launch parameters:

Application logs: Yes, save application logs

Save location:

Edit

Clone

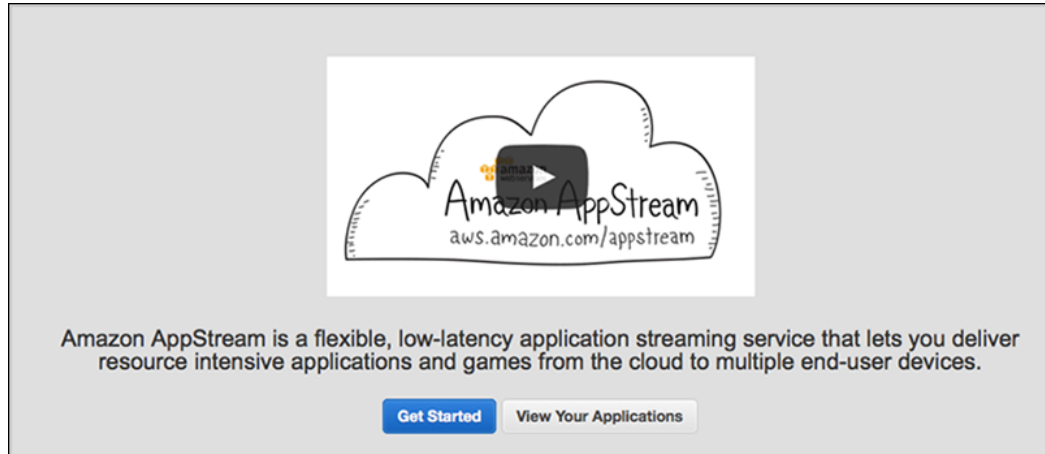
Archive

Archive an Application

You can deprecate a streaming application and prevent it from accepting new client connections. When you do this, the streaming application is put into the archived state. The application can be restored to production at a later time. While the streaming application is being archived, the streaming application continues to stream content to current client sessions, but does not accept new client sessions. When all the existing client sessions have finished, the streaming application enters the archived state.

To archive an application

1. In the [Amazon AppStream console](#), click **View Your Streaming Applications**. This button appears only if you have added applications to Amazon AppStream.



2. In the list of applications, click the name of the streaming application to view.

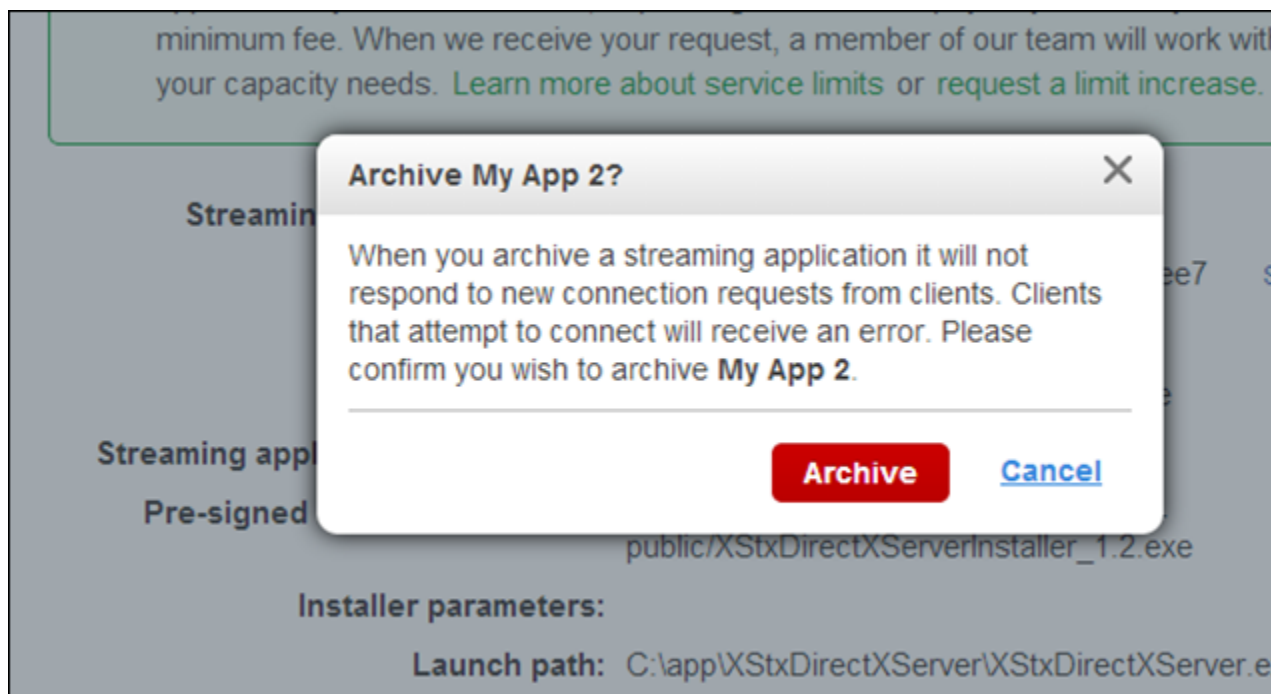
Your account supports streaming up to 10 simultaneous sessions at the same time. [Learn more about service limits](#) or [request a limit increase](#).

[Add a new streaming application](#)

Name	Active sessions	Available sessions	Status
My App	0	2	Running
My App 2	0	2	Running

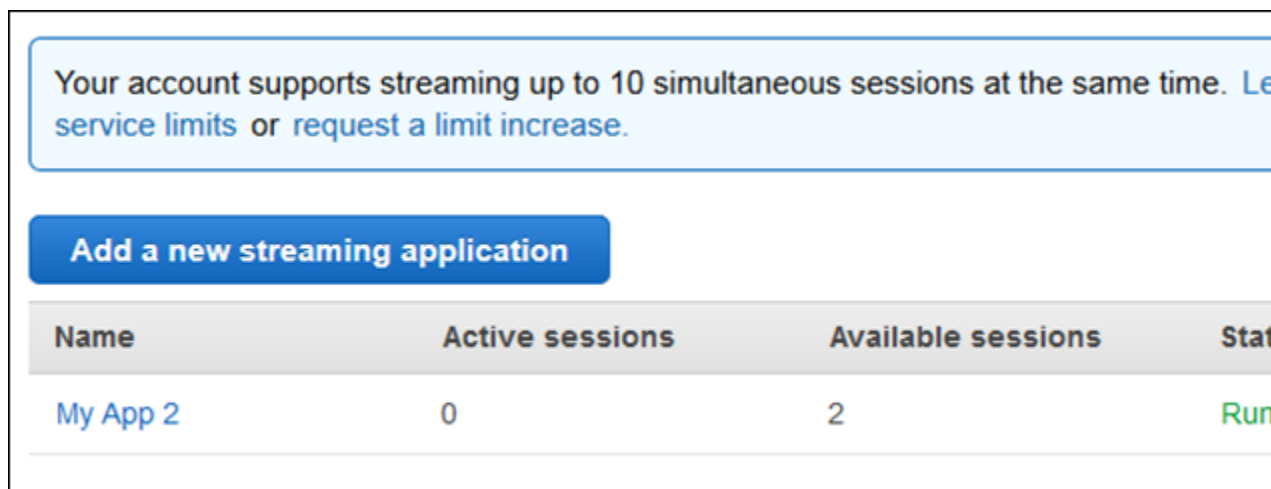
3. On the **Application Summary** page, click **Archive**.

4. In the **Archive** message box, click **Archive**.



Amazon AppStream moves the application into the archived state. This may take a few minutes.

When the application is archived, it no longer appears in the list of applications.



Enable Logging on a Streaming Application

When your application exhibits unusual behaviors, such as unexpected shutdowns, intermittent streaming, or poor performance, or you want to know what your streaming application is doing, you can enable logging to see what is happening.

When logging is enabled, Amazon AppStream creates a .zip file of all the logs that you specify and uploads that .zip file is then uploaded to a specified Amazon S3 bucket in your account. Once the logs are stored in Amazon S3, you can download them and see what was happening within your streaming application during the session.

Although Amazon AppStream only upload logs after a session terminates, log creation during a session can adversely affect the streaming experience. Balance the need for logging events with the effect on the streaming experience by enabling logging.

Note

Amazon AppStream does not encrypt the log files by default. If you want encrypted logs, encrypt them when they are generated (client side encryption) or on the Amazon S3 bucket (server side encryption).

AppStream Log Names

Amazon AppStream uses this pattern to name the .zip file:

`<bucketName>/AppStream/<region>/<Application ID>/yyyy-mm-dd/<log file name>.zip`

Default Amazon AppStream Logs

By default, Amazon AppStream saves the following logs:

- A file called `launch.txt.failed` with the output if Amazon AppStream fails to start the streaming application.
- [Minidump files](#) (.dmp)
- Standard error stream (`launch.txt.stderr`)
- Standard output stream (`launch.txt.stdout`)
- Utilization metrics (`utilization.csv`) which is a file that provides operational metrics. The first row contains column headings and the following rows are comma-separated values:
 - **UTC Time**—Time in Universal Coordinated Time (UTC) when the row was created.
 - **Processor (n) %**—The percentage of use for each CPU core starting with Core 0.
 - **Memory in Use (Bytes)**—Memory that was used in bytes.
 - **Memory Available (Bytes)**—Memory that was available in bytes.
 - **Disk Read (Bytes)**—Disk read speed in bytes per second.
 - **Disk Write (Bytes)**—Disk write speed in bytes per second.
 - **GPU Utilization %**—The percentage of use for the GPU.
 - **GPU Memory Utilization %**—The percentage of GPU memory that was used.
 - **GPU Memory Total (MB)**—The total GPU memory available in the megabytes (MB).
 - **GPU Memory Used (MB)**—The amount of GPU memory used in MB.
 - **GPU Memory Free (MB)**—The amount of GPU memory available in MB.
 - **GPU Temperature (C)**—The temperature of the GPU in degrees centigrade.
 - **GPU Power Draw (W)**—The number of watts the GPU is using.
 - **GPU Power Limit (W)**—The maximum number of watts required by the GPU.

The following is an example of the metrics in this utilization log.

```
UTC Time,Processor(0) %,Processor(1) %,Processor(2) %,Processor(3) %,Memory  
In Use (Bytes),Memory Available (Bytes),Disk Write (Bytes/sec),Disk Read  
(Bytes/sec),GPU Utilization %,GPU Memory Utilization %,GPU Memory Total
```

```
(MB),GPU Memory Used (MB),GPU Memory Free (MB),GPU Temperature (C),GPU Power
Draw (W),GPU Power Limit (W)
2014/05/15
11:17:39,0.92,3.14,6.58,1.88,2724400196.27,14014595549.87,614818.20,103422.13,0,
1, 4095, 4063, 32, 33, 18.44, 125.00
2014/05/15
11:18:40,0.95,1.15,0.32,1.24,2745420390.40,14010372983.47,8689.31,99415.41,0,
1, 4095, 4063, 32, 33, 18.26, 125.00
2014/05/15
11:19:41,0.36,0.67,0.20,1.30,2734278997.33,14029225574.40,6994.11,18322.61,0,
1, 4095, 4063, 32, 33, 18.44, 125.00
2014/05/15
11:20:42,0.08,0.69,0.10,0.97,2724477064.53,14033759027.20,27142.17,14072.01,0,
1, 4095, 4063, 32, 33, 18.54, 125.00
2014/05/15
11:21:43,0.18,0.27,0.10,1.42,2719389627.73,14038990438.40,40446.51,203.51,0,
1, 4095, 4063, 32, 33, 18.53, 125.00
2014/05/15
11:22:44,0.13,0.44,0.12,0.87,2720666555.73,14039756800.00,43574.68,1696.60,0,
1, 4095, 4063, 32, 32, 18.67, 125.00
2014/05/15
11:23:45,0.18,0.44,0.24,0.92,2718720887.47,14040533674.67,44824.72,271.31,0,
1, 4095, 4063, 32, 32, 18.43, 125.00
2014/05/15
11:24:46,0.30,0.50,0.14,1.73,2707566796.80,14053706888.53,72334.31,209776.90,0,
1, 4095, 4063, 32, 32, 18.21, 125.00
2014/05/15
11:25:46,0.34,0.44,0.14,1.52,2654716040.53,14106968064.00,43219.86,1479.45,0,
1, 4095, 4063, 32, 33, 18.41, 125.00
2014/05/15
11:26:47,0.05,0.88,0.16,1.08,2655204010.67,14107344554.67,43875.24,0.00,0, 1,
4095, 4063, 32, 32, 18.31, 125.00
2014/05/15
11:27:48,0.38,0.79,0.16,1.25,2653159219.20,14107987626.67,40567.09,0.00,0, 1,
4095, 4063, 32, 33, 18.23, 125.00
2014/05/15
11:28:49,0.17,0.59,0.09,1.32,2650824772.27,14108805666.13,43111.49,0.00,0, 1,
4095, 4063, 32, 33, 18.20, 125.00
2014/05/15
11:29:50,0.16,0.45,0.07,0.76,2651427157.33,14109135803.73,43341.33,0.00,0, 1,
4095, 4063, 32, 33, 18.33, 125.00
2014/05/15
11:30:51,0.05,0.83,0.11,0.92,2650787703.47,14109916501.33,44690.80,0.00,0, 1,
4095, 4063, 32, 33, 18.30, 125.00
2014/05/15
11:31:52,0.13,0.64,0.29,0.79,2651821602.13,14110050781.87,44109.33,0.00,0, 0,
4095, 4063, 32, 33, 18.18, 125.00
2014/05/15
11:32:52,0.14,0.99,0.44,1.34,2655279445.33,14108282675.20,56575.66,2570.57,0,
0, 4095, 4063, 32, 32, 17.78, 125.00
```

Custom Amazon AppStream Logs

In addition to the above files, Amazon AppStream also collects logs located in directories that you specify. To specify custom files or directories simply include that information during the create application process or by updating the metadata of your application.

If you want Amazon AppStream to create the bucket, the new bucket name will use the following naming convention:

AppStream-*<region>-<Account ID>*

If you want to use your own bucket, you must allow Amazon AppStream to use the `PutObject` method. To allow this method, use a bucket policy similar to the following:

```
{
  "Version": "2008-10-17",
  "Id": "Policy1396472828471",
  "Statement": [
    {
      "Sid": "Stmt1396472820469",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::990116983315:root" },
      "Action": [ "s3:PutObject" ],
      "Resource": "arn:aws:s3:::your bucket name/*"
    }
  ]
}
```

For more information about modifying the Amazon S3 bucket policy, see [Bucket Policy Examples](#) in the Amazon S3 Developer Guide.

You can also specify log files from other directories by filling the text box with the paths to directories that contain the log files or the path and filename of the log. Separate each path with a semi-colon (;). The paths and files must meet the following requirements:

- Log files must be located in the `C:\` directory.
- Log paths must be fully qualified paths. Do not use relative paths. You can use wildcards.
- Log paths must be less than the path length limits. See [Naming Files, Paths, and Namespaces](#).
- Log file names cannot contain Windows reserved characters. See [Naming Conventions](#).
- Your application must have read access to the paths.

Note

The total size of logs in the `.zip` file is subject to the Amazon S3 key size limits.

Enabling Amazon AppStream Logging through The Console

1. Sign in to the AWS Management Console and open the Amazon AppStream console at <https://console.aws.amazon.com/appstream/>.
2. Click **View your streaming applications**.
3. Click on the streaming application that you want to enable logging.
4. On the **Streaming Application Summary** page, click **Edit**.
5. On the **Edit Streaming Application Details** page in **Application Logs**, select **Yes, save application logs**.
6. (Optional) Use the text box that appears to specify other paths or log files to include in the `.zip` file. Each time you fill in a text box, another text box appears so that you can specify other paths or logs.

Edit Streaming Application Details

[Back to streaming application summary](#)

Streaming application name *	<input type="text" value="My App"/>	Enter up to 64 letters, numbers, punctuation marks, and spaces to create a friendly name that identifies this streaming application within the console. This is only used in the console - your customers will not see it.
Streaming application description 512 of 512 characters left	<input type="text" value="Codenames, special version notes, etc."/>	Enter up to 512 letters, numbers, punctuation marks, and spaces to describe your streaming application or add notes. This is only used in the console - your customers will not see it.
Path to launcher *	<input type="text" value="C:\app\XStxDirectXServer\XStxDirectX"/>	Enter an absolute path to the Windows executable (*.exe) file that will launch your streaming application after it is installed, for example, C:\myapp\launch.exe.
Launch parameters	<input type="text" value="/locale en-US"/>	Enter any command-line parameters required to launch your streaming application.
Application Logs	<input checked="" type="checkbox"/> Yes, save application logs <input type="text" value="C:\MyApp\logs"/>	Amazon AppStream collects the following types of logs: <ul style="list-style-type: none">• A file called launch.txt failed with the output if Amazon AppStream fails to start the streaming application.• Minidump files (.dmp)• Standard error stream (launch.txt.stderr)• Standard output stream (launch.txt.stdout)• Utilization metrics (utilization.csv) including CPU, memory and disk usage• You can also enter absolute paths to additional files you want to collect (use * or ? as wild card characters to collect multiple files -- for instance, C:\app\logs*)
Save Location *	<div><div>appstream-us-east-1-111122223333 ▾</div><div>Amazon AppStream is able to write to this S3 bucket.</div></div>	Select an S3 bucket in which to save logs and crash data. Amazon AppStream can create and use a default bucket or use one that already exists. We configure an IAM role to access the default bucket, and offer details for adding access to an existing bucket.

Required *

Cancel

Save

7. To have Amazon AppStream send your logs to the default Amazon S3 bucket, click **Create a default bucket**. To use one of your existing Amazon S3 buckets, click the down arrow next to **Create a default bucket** and select one of your buckets.
8. Click **Save**.

Enabling Amazon AppStream Logging Programmatically

You can enable logging programmatically when you add your streaming application to Amazon AppStream or update the metadata for an existing streaming application. In either case, use one of the methods to provide the relevant data in the `applicationManifest` input field.

To enable logging when creating a new streaming application:

1. Access the root by calling `/`.
2. Follow the [Applications](#) (p. 173) link to get a list of applications.
3. Follow the [application:create](#) (p. 181) link to create the application.
4. Fill in the necessary information in the `applicationManifest` input field.

To enable logging on an existing streaming application:

1. Access the root by calling `/`.
2. Follow the [Applications](#) (p. 173) link to get a list of applications.
3. Follow the [application:by-id](#) (p. 181) link to get the proper application representation.
4. Follow the [application:update](#) (p. 183) link to update the streaming application.
5. Fill in the necessary information in the `applicationManifest` input field.

Increase Your Amazon AppStream Service Limits

By default, your streaming application has a service limit of up to 10 simultaneous sessions. This means that your streaming application can stream to 10 users at any given time. If you need more simultaneous connections, you can request a service limit increase by starting a new case at the [Support Center](#).

Before contacting the Support Center, you should gather the following information:

- The number of simultaneous sessions you need
- Expected date on which you need your new limit.
- How many months the limit needs to be in effect.
- A description of your use case.
- Email addresses of other people who need to know about your request.

After you submit a new case, the Amazon AppStream product team will contact you within three business days to discuss your service limit increase. Depending on the number of simultaneous sessions you need, increasing your service limits can take from 2 weeks to 10 weeks to complete. The product team representative will discuss your needs with you and provide more information about the amount of time required to fulfill your request.

As soon as you know how many simultaneous sessions you'll need, request a session limit increase so that you will have the capacity when you need it.

To request a service limit increase

1. Go to the [Support Center](#) and click **Open a new case**.
2. Select **Service Limit increase**. The form changes after you make this selection.

The screenshot shows the AWS Support Center interface for opening a new case. The breadcrumb trail is 'Home > Open a new case'. The 'Regarding' section has three radio buttons: 'Account and Billing Support', 'Service Limit Increase' (which is selected), and 'Technical Support'. The 'Limit Type' dropdown is set to 'AppStream'. Other fields include 'Expected launch date', 'Peak number of simultaneous users', 'Sector/industry' (set to '-- select --'), and 'Expected Lifetime (months)'. A large text area for 'Use Case Description' is present. The 'Your Contact Information' section shows 'Name' as 'John Doe', 'CC' as 'janedoe@example.com, richardroe@example.com', 'Company Name' as 'Example Corp.', 'Account' as '1111-2222-3333', and 'Account Email' as 'johndoe@example.com'. A note below the CC field says 'Separate multiple addresses with commas or semi-colons.' At the bottom, there is a 'Please select a contact method:' section with a 'Web' button. A footer note states 'Send us an e-mail and we will get back to you within 1 business day.'

3. For **Regarding**, click **Service Limit Increase**.
4. For **Limit Type**, click **AppStream**.
5. Fill in all of the required options in the form and then click **Web**.

Security Considerations

Topics

- [Controlling Access Using IAM \(p. 154\)](#)
- [Security Best Practices \(p. 157\)](#)

Using IAM to Control Access to Amazon AppStream Resources

Amazon AppStream integrates with AWS Identity and Access Management (IAM), which allows you to control access to Amazon AppStream.

For general information about IAM, go to:

- [Identity and Access Management \(IAM\)](#)
- [Using IAM](#)

You can give IAM users of your AWS account access to all Amazon AppStream operations or to a subset of them. The following is the list of Amazon AppStream operations that can be made available to IAM users.

```
appstream:GetApiRoot
appstream:GetApplications
appstream:GetApplication
appstream:GetApplicationStatus
appstream:GetApplicationErrors
appstream:GetApplicationError
appstream>CreateApplication
appstream:UpdateApplication
appstream>DeleteApplication
appstream:UpdateApplicationState
appstream:GetSessions
appstream:GetSession
appstream:GetSessionStatus
```



```
appstream:CreateSession
appstream:UpdateSessionState
```

Example IAM User Policies for Amazon AppStream

By default, IAM users have no access to Amazon AppStream or to the resources that it uses. If you want IAM users to be able to work with Amazon AppStream, for example, in the AWS Management Console, you must grant them permissions.

This section shows simple policies for controlling access to Amazon AppStream. To use these policies, you create an IAM user and attach one of these policies to the user or to the IAM group that the user belongs to.

Give users access to DynamoDB and a specific set of AppStream API calls

The following policy lets users access DynamoDB.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1394569913000",
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "Stmt1394569933000",
      "Effect": "Allow",
      "Action": [
        "appstream:CreateSession",
        "appstream:GetApiRoot",
        "appstream:GetApplication",
        "appstream:GetApplications",
        "appstream:GetApplicationStatus",
        "appstream:GetSession",
        "appstream:GetSessions",
        "appstream:GetSessionStatus",
        "appstream:UpdateSessionState"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Give IAM users broad access to Amazon AppStream

The following policy lets users perform any Amazon AppStream action.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "appstream:*"
      ],
      "Resource": [ "*" ]
    }
  ]
}
```

Give users permission to modify applications and sessions

The following policy grants users the permission to create, update, and delete applications and sessions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "appstream:Create*",
        "appstream:Update*",
        "appstream>Delete*"
      ],
      "Resource": [ "*" ]
    }
  ]
}
```

Give users permission to modify applications

The following policy grants users the permission to deploy applications on Amazon AppStream and to update or delete those applications.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "appstream:CreateApplication",
        "appstream:UpdateApplication*",
        "appstream>DeleteApplication"
      ],
      "Resource": [ "*" ]
    }
  ]
}
```

```
]
}
```

Give users read-only access to Amazon AppStream

The following policy grants users read-only access to Amazon AppStream.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "appstream:Get*"
      ],
      "Resource": ["*"]
    }
  ]
}
```

Security Best Practices

AWS has several features to help you keep your assets secure.

Versioning

Versioning offers an additional level of protection by providing a means of recovery when customers accidentally overwrite or delete objects. This allows you to easily recover from unintended user actions and application failures. You can also use versioning for data retention and archiving. For more information, see [Amazon Simple Storage Service FAQs](#) and the [Amazon Simple Storage Service Developer Guide](#).

Multi-Factor Authentication

AWS multi-factor authentication (MFA) is an additional layer of security that offers enhanced control over your AWS account settings and the management of the AWS resources to which the account has subscribed. When you enable this opt-in feature, you need to provide a six-digit single-use code in addition to your user name and password before access is granted. You get this single use code from an authentication device or a special application on a mobile phone that you keep in your physical possession.

This feature is called multi-factor authentication because two factors are checked before access is granted to your account: you need to provide both your AWS email ID and password (the first factor: something you know) and the particular code from your authentication device (the second factor: something you have). You can enable multi-factor authentication for your AWS account as well as for the users you have created under your AWS account using IAM.

It's easy to obtain an authentication device from a participating third-party provider. You can also download and install appropriate software on your mobile phone, then set it up for use via the AWS website. For more information, see [AWS Multi-Factor Authentication](#).

Key Rotation

You should keep your AWS passwords and access keys safe for the same reasons it is important to change your password frequently. AWS recommends that you rotate your access keys and certificates on a regular basis. To let you do this without potential impact to the availability of your applications, AWS supports multiple concurrent access keys and certificates. With this feature, you can regularly rotate keys and certificates into and out of operation without any downtime to your application. This can help to mitigate risk from lost or compromised access keys or certificates. You can use the IAM APIs to rotate the access keys of your AWS account as well as for users created under your AWS account. For more information, see [AWS Security Credentials](#).

Use A Strong Password For Remote Management

Use a strong password with remote management services such as SSH and VNC to restrict access to your instances. If you do not configure a strong password with these remote management services, malicious users could access your instances.

Restrict Access to Your Streaming Application

Restrict your security groups to allow connections only from ports required to support the necessary services. The following are suggested ports to allow incoming connections:

- **SSH**—port 22
- **STX**—port 80
- **STX TCP**—port 5900
- **STX UDP**—ports 9070-9080

For additional protection, restrict access to incoming traffic to a group of IP addresses.

Troubleshooting Amazon AppStream

If you encounter a problem with your product, the following topics offer information about possible causes of your issue as well as proposed solutions.

If you are still having a problem, check the [Amazon AppStream forum](#) or contact [Amazon AppStream support](#).

Topics

- [Deployment Problems](#) (p. 159)
- [Streaming Problems](#) (p. 160)
- [Error Codes](#) (p. 160)

Deployment Problems

If you encounter an issue with deploying your application, the following checks may help you locate the source of the problem.

Is Your Installer Corrupted?

Download your application installer from Amazon S3 and verify that it works as intended.

Has Your Pre-Signed URL Expired?

The URL you pass in to Amazon AppStream for your application installer must point to a file stored in Amazon S3. The URL must be a pre-signed URL that has not expired. If you are not sure when your pre-signed URL expires, generate a new pre-signed URL and pass that in.

Does Your Pre-Signed URL Use HTTP Protocol?

The pre-signed URL you pass in to Amazon AppStream; for your application installer must use HTTPS protocol instead of HTTP. You can check the protocol by looking at the beginning of your pre-signed URL.

It should begin with the characters **https://**. If it does not, generate a new pre-signed URL using the HTTPS protocol. .

Streaming Problems

If you encounter a problem with your streaming application, such as unexpected behaviors or poor streaming quality, enable logging to get a log file with the events that have occurred while your streaming application was running on Amazon AppStream. The events may provide detailed information about your problem. Simply write to any log directory you have access to and configure Amazon AppStream to upload those logs to the specified Amazon S3 bucket.

To enable logging, see [Enable Logging on a Streaming Application \(p. 147\)](#).

Error Codes

The following are the error messages you may encounter using Amazon AppStream. You can see these error messages through [application:errors \(p. 185\)](#) and [Application Error \(p. 177\)](#).

Topics

- [APPLICATION_DELETION_FAILED \(p. 160\)](#)
- [APPLICATION_INSTALLATION_FAILED \(p. 160\)](#)
- [APPLICATION_LAUNCH_FAILED \(p. 161\)](#)
- [APPLICATION_INSTALLATION_NOT_SILENT \(p. 161\)](#)
- [APPLICATION_INSTALLATION_TIMED_OUT \(p. 161\)](#)
- [APPLICATION_LAUNCH_TIMED_OUT \(p. 161\)](#)
- [APPLICATION_RUNTIME_FAILURE \(p. 162\)](#)
- [INTERNAL_FAILURE \(p. 162\)](#)
- [SDK_VERSION_DETECTION_FAILED \(p. 162\)](#)
- [S3_URL_INVALID \(p. 162\)](#)

APPLICATION_DELETION_FAILED

This error occurs when Amazon AppStream attempts to delete your application.

Cause

Amazon AppStream generated an unknown error while deleting an application.

Solution

Contact [Amazon AppStream support](#)

APPLICATION_INSTALLATION_FAILED

This error occurs when Amazon AppStream attempts to install your application.

Cause

Amazon AppStream generated an unknown error while installing your application.

Solution

Ensure that your application installer installs your application in a reasonable amount of time. If the application installer works, then contact [Amazon AppStream support](#).

APPLICATION_LAUNCH_FAILED

This error occurs when Amazon AppStream starts your application.

Cause

Amazon AppStream generated an unknown error while starting your application.

Solution

Contact [Amazon AppStream support](#)

APPLICATION_INSTALLATION_NOT_SILENT

This error occurs when Amazon AppStream detects that the application installer requires user interaction.

Cause

Amazon AppStream detected that the application installer requires user interaction.

Solution

Revise the application installer to run without any user interaction. For more information, see [Build an Application Installer \(p. 73\)](#).

APPLICATION_INSTALLATION_TIMED_OUT

This error occurs when Amazon AppStream runs the application installer.

Cause

Amazon AppStream timed out waiting for the application installer to complete installing the application.

Solution

Revise the application installer to complete the installation in a reasonable amount of time. For more information, see [Build an Application Installer \(p. 73\)](#). Deploy the updated application installer.

APPLICATION_LAUNCH_TIMED_OUT

This error occurs when Amazon AppStream attempted to start your application.

Cause

Amazon AppStream timed out while starting your application.

Solution

Verify that your application installer installs the files and sets the correct permission. The application installer must meet the requirements in [Build an Application Installer \(p. 73\)](#). Deploy the updated application installer.

APPLICATION_RUNTIME_FAILURE

This error occurs when Amazon AppStream runs your application.

Cause

Your application has stopped.

Solution

Contact [Amazon AppStream support](#)

INTERNAL_FAILURE

This error occurs in Amazon AppStream.

Cause

An unknown error occurred in Amazon AppStream.

Solution

Contact [Amazon AppStream support](#)

SDK_VERSION_DETECTION_FAILED

This error occurs when Amazon AppStream runs the application installer.

Cause

Your application does not integrate with the libraries in the Amazon AppStream SDK.

Solution

Revise the application code to integrate the libraries in the Amazon AppStream SDK. Deploy the updated application.

S3_URL_INVALID

This error occurs when Amazon AppStream runs the application installer.

Cause

- The specified pre-signed URL to the Amazon S3 bucket with your application installer is incorrect.
- The application installer in the Amazon S3 bucket is not set to public.
- The expiry time you set when creating the pre-signed URL is inadequate.
- You used the HTTP protocol to create the pre-signed URL.

Solution

- Use the correct pre-signed Amazon S3 URL.
- Change the application installer in the Amazon S3 bucket to public. In the Amazon S3 bucket, click your application installer, click **Actions**, and then click **Make Public**.
- Create a pre-signed URL with a longer expiry time.
- Create a pre-signed URL with the HTTPS protocol.

Amazon AppStream REST API

The Amazon AppStream web service provides APIs you can call to manage applications hosted on Amazon AppStream and to manage client sessions connecting to those applications.

You can use this API to:

- Programmatically manage applications hosted on Amazon AppStream. For example, you can add new applications to Amazon AppStream and automate tasks you would otherwise perform through the console.
- Create tools or services to control access to your applications. The example found in [the section called “Build an Entitlement Service” \(p. 73\)](#) demonstrates one such service.

You can call an API provided by Amazon AppStream by either submitting a REST request, or by calling wrapper functions in the Amazon AppStream SDK. You can download the Amazon AppStream SDK from the links in [Downloads \(p. 10\)](#).

Topics

- [Hypertext Application Language \(p. 164\)](#)
- [Making HTTP Requests \(p. 165\)](#)
- [Signing Requests \(p. 168\)](#)
- [Handling Errors \(p. 169\)](#)
- [Resources \(p. 172\)](#)
- [Link Relations \(p. 181\)](#)

Hypertext Application Language

The Amazon AppStream web service is a resource-based API that uses Hypertext Application Language (HAL). HAL provides a standard way for expressing the resources and relationships of an API as hyperlinks. Using HAL, you use HTTP methods (GET, PUT, POST, DELETE) to submit requests and receive information about the API in the response. Applications can use the information returned to explore the functionality of the API.

For example, you can inspect the JSON returned in the response from the web service on an [Application \(p. 174\)](#) resource to discover the [session:entitle \(p. 185\)](#) link. By extracting the `href` property of that link, you can programmatically build the request needed to create a new client session for that application. For more information about HAL, see the [JSON Hypertext Application Language](#) draft.

Making HTTP Requests to Amazon AppStream

Amazon AppStream REST requests are HTTP requests as defined in RFC 2616. (For more information, go to <http://www.ietf.org/rfc/rfc2616.txt>.) This section describes the structure of an Amazon AppStream REST request. For detailed descriptions of the resources and references of the API, see [Resources](#) (p. 172).

A typical REST action consists of sending an HTTP request to Amazon AppStream and waiting for the response. Like any HTTP request, a REST request to Amazon AppStream contains a request method, a URI, request headers, and sometimes a query string or request body. The response contains an HTTP status code, response headers, and sometimes a response body.

Topics

- [Limits on Request Rates](#) (p. 165)
- [HTTP Header Contents](#) (p. 165)
- [HTTP Request Body](#) (p. 166)
- [HTTP Responses](#) (p. 166)

Limits on Request Rates

Amazon AppStream limits the rate at which you can submit requests:

- You can submit a maximum of two POSTs to the applications resource per second per AWS account.
- You can submit a maximum of four POSTs to an application's sessions resource per second per AWS account.

If you submit requests at a faster rate, Amazon AppStream may respond with HTTP 429 errors as explained in [API Error Codes \(Client and Server Errors\)](#) (p. 169).

HTTP Header Contents

Amazon AppStream requires the following information in the headers of an HTTP request:

Host (Required)

The Amazon AppStream endpoint. This value must be
`https://appstream.us-east-1.amazonaws.com.`

x-amz-date or Date (Required)

The date used to create the signature contained in the `Authorization` header. Specify the date in ISO 8601 standard format, in UTC time, as in the following example: `X-Amz-Date: 20130613T203622Z`.

You must include either `x-amz-date` or `Date`. (Some HTTP client libraries don't let you set the `Date` header). When an `x-amz-date` header is present, the system ignores any `Date` header when authenticating the request.

The time stamp must be within 15 minutes of the AWS system time when the request is received. If it isn't, the request fails with the `RequestExpired` error code to prevent someone else from replaying your requests.

Authorization (Required)

The information required for request authentication. For more information about constructing this header, see [Signing Requests](#) (p. 168).

Content-Type (Conditional)

Specifies JSON and the version, for example, `Content-Type: application/x-amz-json-1.0`.

Condition: Required for PUT and POST requests.

Content-Length (Conditional)

Length of the message (without the headers) according to RFC 2616.

Condition: Required if the request body itself contains information (most toolkits add this header automatically).

The following are example headers for an HTTP request to entitle a new client session.

```
POST /applications/e407fbc8-5c27-48e6-9412-a876167546e8/sessions HTTP/1.1
host: appstream.us-east-1.amazonaws.com
x-amz-date: 20120116T174952Z
Authorization: AWS4-HMAC-SHA256 Credential=AccessKeyID/20120116/us-east-1/ets/aws4_request,SignedHeaders=host;x-amz-date;x-amz-target,Signature=145b1567ab3c50d929412f28f52c45dbf1e63ec5c66023d232a539a4afd11fd9
content-type: application/x-amz-json-1.0
content-length: 56
{
  "opaqueData": "TYtYS00MaWQ9Y2JmOmM4hhZWNS00MjJ"
}
```

HTTP Request Body

Many Amazon AppStream API actions require you to include JSON-formatted data in the body of the request. The JSON conforms to the Amazon AppStream schema.

Note

JSON values in the request body are strings.

HTTP Responses

All Amazon AppStream API actions include JSON-formatted data in the response. The JSON conforms to the Amazon AppStream schema.

Note

JSON values in the response are strings.

Here are some important headers in the HTTP response and how you should handle them in your application, if applicable:

HTTP/1.1

This header is followed by a status code. Status code 200 indicates a successful operation. For information about error codes, see [API Error Codes \(Client and Server Errors\)](#) (p. 169).

Type: String

x-amzn-RequestId

A value created by Amazon AppStream that uniquely identifies your request, for example, K2QH8DNOU907N97FNA2GDLL8OBVV4KQNSO5AEMVJF66Q9ASUAAJG. If you have a problem with Amazon AppStream, AWS can use this value to troubleshoot the problem. We recommend that you log these values.

Type: String

Content-Length

The length of the response body in bytes.

Type: String

Content-Type

The type of the message's content. Usually `application/hal+json`.

Type: String

Date

The date and time that Amazon AppStream responded, for example, `Sun, 25 Mar 2012 12:00:00 GMT`. The format of the date must be one of the full date formats specified by RFC 2616, section 3.3.

Type: String

Signing Requests

Amazon AppStream requires that you authenticate every request you send by signing the request. To sign a request, you calculate a digital signature using a cryptographic hash function, which returns a hash value based on the input. The input includes the text of your request and your secret access key. The hash function returns a hash value that you include in the request as your signature. The signature is part of the `Authorization` header of your request.

After receiving your request, Amazon AppStream recalculates the signature using the same hash function and input that you used to sign the request. If the resulting signature matches the signature in the request, Amazon AppStream processes the request. Otherwise, the request is rejected.

Amazon AppStream supports authentication using [AWS Signature Version 4](#). The process for calculating a signature can be broken into three tasks:

- [Task 1: Create a Canonical Request](#)

Create your HTTP request in canonical format as described in [Task 1: Create a Canonical Request For Signature Version 4](#) in the *Amazon Web Services General Reference*.

- [Task 2: Create a String to Sign](#)

Create a string that you will use as one of the input values to your cryptographic hash function. The string, called the *string to sign*, is a concatenation of the name of the hash algorithm, the request date, a *credential scope* string, and the canonical request from the previous task. The *credential scope* string itself is a concatenation of date, region, and service information.

For the `X-Amz-Credential` parameter, specify:

- The code for the endpoint to which you're sending the request, for example, `us-east-1`. For a list of regions and endpoints for Amazon AppStream, see the [Regions and Endpoints](#) chapter of the *Amazon Web Services General Reference*. When specifying the code for the endpoint, include only the part between `appstream.` and `.amazonaws.com`
- `appstream` for the service abbreviation

For example:

```
X-Amz-Credential=AKIAIOSFODNN7EXAMPLE/20130501/us-east-1/appstream/aws4_request
```

- [Task 3: Create a Signature](#)

Create a signature for your request by using a cryptographic hash function that accepts two input strings: your *string to sign* and a *derived key*. The *derived key* is calculated by starting with your secret access key and using the *credential scope* string to create a series of hash-based message authentication codes (HMACs).

Handling Errors in Amazon AppStream

Topics

- [API Error Codes \(Client and Server Errors\) \(p. 169\)](#)
- [Catching Errors \(p. 171\)](#)
- [Error Retries and Exponential Backoff \(p. 172\)](#)

When you send requests to and get responses from the Amazon AppStream API, you might encounter two types of API errors:

- **Client errors:** Client errors are indicated by a 4xx HTTP response code. Client errors indicate that Amazon AppStream found a problem with the client request, such as an authentication failure or missing required parameters. Fix the issue in the client application before submitting the request again.
- **Server errors:** Server errors are indicated by a 5xx HTTP response code, and need to be resolved by Amazon. You can resubmit/retry the request until it succeeds.

For each API error, Amazon AppStream returns the following values:

- A status code, for example, 400
- An error code, for example, `ValidationException`
- An error message, for example, `Supplied AttributeValue is empty, must contain exactly one of the supported datatypes`

For a list of error codes that Amazon AppStream returns for client and server errors, see [API Error Codes \(Client and Server Errors\) \(p. 169\)](#).

API Error Codes (Client and Server Errors)

HTTP status codes indicate whether an operation is successful or not.

A response code of 2xx indicates the operation was successful. Other error codes indicate either a client error (4xx) or a server error (5xx).

The following table lists the errors returned by Amazon AppStream. Some errors are resolved if you simply retry the same request. The table indicates which errors are likely to be resolved with successive retries. If the value of the Retry column is:

- **Yes:** Submit the same request again.
- **No:** Fix the problem on the client side before submitting a new request.

For more information about retrying requests, see [Error Retries and Exponential Backoff \(p. 172\)](#).

HTTP Status Code	Error code	Message	Cause	Retry
400	Conditional Check Failed Exception	The conditional request failed.	Example: The expected value did not match what was stored in the system.	No

HTTP Status Code	Error code	Message	Cause	Retry
400	Incomplete Signature Exception	The request signature does not conform to AWS standards.	The signature in the request did not include all of the required components. See HTTP Header Contents (p. 165) .	No
400	Missing Authentication Token Exception	The request must contain a valid (registered) AWS Access Key ID.	The request did not include the required <code>x-amz-security-token</code> . See Making HTTP Requests to Amazon AppStream (p. 165) .	No
400	Validation Exception	Various.	One or more values in a request were missing or invalid; for example, a value was empty or was greater than the maximum valid value.	No
403	AccessDenied Exception	<ul style="list-style-type: none"> Deleting a system preset is not allowed: <code>account=<accountId>, presetId=<presetId></code>. General authentication failure. The client did not correctly sign the request. See Signing Requests (p. 168). 	You attempted to delete a system preset, the signature in a call to the Amazon AppStream API was invalid, or the IAM user whose credentials were used for this request is not authorized to perform the operation.	No
404	ResourceNot Found Exception	<ul style="list-style-type: none"> The specified <code><resource></code> could not be found: <code><resourceId></code>. 	Example: The application to which you're trying to create a new session doesn't exist or is still being created.	No
409	Resource InUse Exception	<ul style="list-style-type: none"> The <code><resource></code> was already in use: <code>accountId=<accountId>, resourceId=<resourceId></code>. 	Example: You attempted to delete an application that is currently in use.	No
429	Limit Exceeded Exception	<ul style="list-style-type: none"> The account already has the maximum number of sessions allowed: <code>account=<accountId>, maximum number of sessions=<maximum></code> 	The current AWS account has exceeded limits on Amazon AppStream objects. .	

HTTP Status Code	Error code	Message	Cause	Retry
429	Provisioned Throughput Exceeded Exception	You exceeded your maximum allowed provisioned throughput.	Example: Your request rate is too high. The AWS SDKs for Amazon AppStream automatically retry requests that receive this exception. Your request is eventually successful unless your retry queue is too large to finish. Reduce the frequency of requests. For more information, see Error Retries and Exponential Backoff (p. 172) .	Yes
429	Throttling Exception	Rate of requests exceeds the allowed throughput.	You are submitting requests too rapidly; for example, requests to entitle new sessions.	Yes
500	Internal Failure	The server encountered an internal error trying to fulfill the request.	The server encountered an error while processing your request.	Yes
500	Internal Server Error	The server encountered an internal error trying to fulfill the request.	The server encountered an error while processing your request.	Yes
500	Internal Service Exception		The service encountered an unexpected exception while trying to fulfill the request.	Yes
500	Service Unavailable Exception	The service is currently unavailable or busy.	There was an unexpected error on the server while processing your request.	Yes

Sample Error Response

The following is an HTTP response indicating that the value for `inputBucket` was null, which is not a valid value.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: b0e91dc8-3807-11e2-83c6-5912bf8ad066
x-amzn-ErrorType: ValidationException
Content-Type: application/json
Content-Length: 124
Date: Mon, 26 Nov 2012 20:27:25 GMT

{"message": "1 validation error detected: Value null at 'InstallS3Bucket' failed to satisfy constraint: Member must not be null"}
```

Catching Errors

For your application to run smoothly, you need to build in logic to catch and respond to errors. One typical approach is to implement your request within a `try` block or `if-then` statement.

The AWS SDKs perform their own retries and error checking. If you encounter an error while using one of the AWS SDKs, you should see the error code and description. You should also see a `Request ID` value. The `Request ID` value can help troubleshoot problems with Amazon AppStream support.

Error Retries and Exponential Backoff

Numerous components on a network, such as DNS servers, switches, load balancers, and others can generate errors anywhere in the life of a given request.

The usual technique for dealing with these error responses in a networked environment is to implement retries in the client application. This technique increases the reliability of the application and reduces operational costs for the developer.

Each AWS SDK supporting Amazon AppStream implements automatic retry logic. The AWS SDK for Java automatically retries requests, and you can configure the retry settings using the `ClientConfiguration` class. For example, in some cases, such as a web page making a request with minimal latency and no retries, you might want to turn off the retry logic. Use the `ClientConfiguration` class and provide a `maxErrorRetry` value of 0 to turn off the retries.

If you're not using an AWS SDK, you should retry original requests that receive server errors (5xx). However, client errors (4xx, other than a `ThrottlingException` or a `ProvisionedThroughputExceededException`) indicate you need to revise the request itself to correct the problem before trying again.

In addition to simple retries, we recommend using an exponential backoff algorithm for better flow control. The idea behind exponential backoff is to use progressively longer waits between retries for consecutive error responses. For example, you might let one second elapse before the first retry, four seconds before the second retry, 16 seconds before the third retry, and so on. However, if the request has not succeeded after a minute, the problem might be a hard limit and not the request rate. For example, you may have reached the maximum number of pipelines allowed. Set the maximum number of retries to stop around one minute.

Resources

The Amazon AppStream API includes the following resources.

Topics

- [AppStream](#) (p. 172)
- [Applications](#) (p. 173)
- [Application](#) (p. 174)
- [Application Errors](#) (p. 176)
- [Application Error](#) (p. 177)
- [Application Status](#) (p. 177)
- [Sessions](#) (p. 179)
- [Session](#) (p. 179)
- [Session Status](#) (p. 180)

AppStream

The root of the Amazon AppStream service.

Links

Relation	Description	Methods	Templated
self	The root resource of Amazon AppStream.	GET	No.
appstream:applications	A link to the applications resource for the service.	GET	No.

Properties

The AppStream resource has no properties.

Applications

The Applications resource is a collection resource that contains zero or more references to your existing applications, and links that guide you on ways to interact with your collection. The collection offers a paginated view of the contained applications.

Links

Relation	Description	Methods	Templated
self	The collection of applications you have hosted on Amazon AppStream.	GET	No.
application:by-id (p. 181)	Retrieves an individual Application resource based on the specified identifier.	GET	Yes. Requires the application identifier.
application:create (p. 181)	Adds a new application to Amazon AppStream. You must have previously uploaded the application package to Amazon S3 and created a pre-signed URL. For more information about generating pre-signed URLs, see Share an Object with Others in the <i>Amazon Simple Storage Service Developer Guide</i> .	POST	No.
item (p. 187)	An array of links to the current page of Application resources.	GET	No.

Relation	Description	Methods	Templated
next (p. 187)	The next page of items in a collection. If there are no further pages of items, this link is not returned in the response.	GET	No.
first (p. 187)	The first page of items in a collection. This link is returned only when on pages other than the first one.	GET	No.

Properties

The Applications resource has no properties.

Application

An application you have added to Amazon AppStream. You are only able to access applications added to your AWS account.

Links

Relation	Description	Methods	Templated
self	An application hosted on Amazon AppStream.	GET	No.
application:status	The current status of this application.	GET	No.
application:update (p. 183)	Update selected metadata for this application.	POST	No.
application:errors	The collection of errors for this application.	GET	No.
application:archive	Archives the application.	PUT	No.
application:reactivate	Activates a previously archived application.	PUT	No.
application:delete	Deletes an application that is in the <code>Error</code> state.	DELETE	No.
application:sessions	The collection of sessions associated with this application.	GET	No.
session:by-id (p. 185)	The session which has the specified identifier.	GET	Yes. Requires the session identifier.
session:entitle (p. 185)	Create a new client session for this application.	POST	No.

Relation	Description	Methods	Templated
collection (p. 187)	The collection of applications you have hosted on Amazon AppStream. The collection includes this application.	GET	No.

Properties

The following properties of the application are set when you create the application, either by using the REST API or the Amazon AppStream console.

Name	Description
id	The application identifier. This is unique across all your applications in Amazon AppStream.
name	The name of the application.
description	(Optional) The description of the application. This description is used to describe the application in Amazon AppStream, it is not displayed to end users.
installerUrl	A pre-signed URL that points to the location in Amazon S3 that contains the installation package for the application. For more information on creating a pre-signed URL, see Share an Object with Others in the <i>Amazon Simple Storage Service Developer Guide</i> .
installerParameters	The parameters required by the application installer to install that application on a host managed by Amazon AppStream.
launchCommand	The command to run in order to launch the application after it is hosted on Amazon AppStream. Important This command should not include any command-line parameters.
logBucket	The name of the Amazon S3 bucket where the <code>.zip</code> files contains the logs are uploaded. The bucket must already exist and allow Amazon AppStream to use the <code>PutObject</code> method.

Name	Description
logPaths	<p>A string array whose elements contain the log filenames or directory path where logs are stored. The filenames and paths must meet the following requirements:</p> <ul style="list-style-type: none"> Log files must be located in the <code>C:\</code> directory. Log paths must be fully qualified paths. Do not use relative paths. You can use wildcards. Log paths must be less than the path length limits. See Naming Files, Paths, and Namespaces. Log file names cannot contain Windows reserved characters. See Naming Conventions. Your application must have read access to the paths. <p>Note The total size of logs in the <code>.zip</code> file is subject to the Amazon S3 key size limits.</p>
applicationErrorCount	The total number of errors associated with this application. This does not include errors associated with client sessions, which are enumerated in <code>sessionErrorCount</code> .
activeSessions	The total number of client sessions currently connected to this application.
availableSessions	The additional capacity available to accept client sessions.
sessionErrorCount	The total number of errors associated with client sessions connected to this application.
createdDate	The date the application was created, in ISO 8601 format.
lastUpdatedDate	The date the application metadata was last updated in Amazon AppStream, in ISO 8601 format.

Application Errors

The collection of [Application Error](#) resources associated with an application hosted on Amazon AppStream.

Links

Relation	Description	Methods	Templated
self	The collection of errors associated with an application.	GET	No.
item (p. 187)	An array of links to the current page of Application Error resources.	GET	No.

Relation	Description	Methods	Templated
next (p. 187)	The next page of items in a collection. If there are no further pages of items, this link is not returned in the response.	GET	No.
first (p. 187)	The first page of items in a collection. Only available if on a page other than the first.	GET	No.

Properties

The Application Errors resource does not have any properties.

Application Error

An error associated with the current application.

Links

Relation	Description	Methods	Templated
self	The error state.	GET	No.
collection (p. 187)	The collection of errors associated with the current application. The collection includes this error.	GET	No.

Properties

Name	Description
id	The identifier of the error. For more information about the errors returned by the Amazon AppStream service, see Handling Errors in Amazon AppStream (p. 169) .
state	The current state of the error. This can be one of the following: <code>New</code> , <code>Read</code> , or <code>Deleted</code> .
type	The type of error. This can either be <code>Application</code> or <code>Session</code> .
message	A message that describes the error.
errorDate	The date the error occurred, in ISO 8601 format.

Application Status

The current status of an application hosted on Amazon AppStream.

Links

Relation	Description	Methods	Templated
self	The status of an application.	GET	No.
up (p. 187)	The application to which this status applies.	GET	No.

Properties

Name	Description
state	<p>The state of the current application. This can be one of the following values:</p> <ul style="list-style-type: none"> • Active—The application is ready to accept client sessions. • Archived—The application must be reactivated before accepting client sessions. • Archiving—The application is in the process of archiving. No new sessions can be entitled, but existing sessions will continue until these sessions finish. To end the existing sessions, explicitly terminate the existing session. • Blocked—The application is waiting on dependencies. • Building—Amazon AppStream is allocating resources to host the application. • Deleting—The application is in the process of being deleted. • Error—The application failed to build or deploy properly. • New—The application has just been created. • Unknown—The application state cannot be determined.
buildStep	<p>This property is present when the state is either Building or Error. It indicates either the currently processing step if the state is Building, or the step that was processing if the state is Error. It can be one of the following values:</p> <ul style="list-style-type: none"> • Copying—Copying the installer data into the hosting environment. • Installing—Installing the application into the hosting environment. • Preparing—Preparing the host environment for application installation. • Provisioning—Provisioning the hosting environment to run your application.

Sessions

The list of sessions associated with the current application. You can only access sessions for applications associated with your AWS account.

Links

Relation	Description	Methods	Templated
self	The collection of sessions associated with the current application.	GET	No.
session:by-id (p. 185)	The session with the specified identifier.	GET	Yes. Requires the session identifier.
item (p. 187)	An array of links to the current page of Session resources.	GET	No.

Properties

The sessions resource has no properties.

Session

A resource representing an individual client session of an application hosted on Amazon AppStream.

Links

Relation	Description	Methods	Templated
self	A session of the specified application	GET	No.
session:status (p. 186)	The status of the current session.	GET	No.
session:terminate (p. 186)	Terminate the current session.	PUT	No.
collection (p. 187)	The list of all client sessions for the specified application.	GET	No.

Properties

Name	Description
id	Unique identifier for the session.
entitlementUrl	The URL clients use to redeem an entitlement for this session and connect to the application.

Name	Description
opaqueData	Data to pass to the application. This data is not used by the client, entitlement service, or Amazon AppStream, it is used by the application. An example of opaqueData would be a user identifier, which the application would then use to load previous state information (such as high score or current map level) for that user from a database.
errorCount	The number of errors associated with this session.
startDate	The time at which the session began, in ISO 8601 format.
endDate	The time at which the session ended, in ISO 8601 format. If this is null, the session is active and the client is currently connected.

Session Status

Returns the status of the current session.

Links

Relation	Description	Methods	Templated
self	The status of the current session.	GET	No.
up (p. 187)	The session to which this state applies.	GET	No.

Properties

Name	Description
state	<p>Current state of the session. This can be one of the following values:</p> <ul style="list-style-type: none">• Unknown—the session state cannot be determined.• Entitled—the session has been entitled, but the client has not yet redeemed the entitlement.• Reserved—the server side of the session is ready to receive the client, but the client has not yet connected.• Active—the session is actively streaming the application to a client.• Completed—the session ended by either client or server action.• Terminating—the session was terminated by the developer and is in the process of ending.• Terminated—the session was terminated by the developer.

Link Relations

The Amazon AppStream API provides the following link relations that you can use to access and modify Amazon AppStream resources.

Topics

- [appstream:applications](#) (p. 181)
- [application:by-id](#) (p. 181)
- [application:create](#) (p. 181)
- [application:update](#) (p. 183)
- [application:archive](#) (p. 184)
- [application:reactivate](#) (p. 184)
- [application:delete](#) (p. 185)
- [application:status](#) (p. 185)
- [application:errors](#) (p. 185)
- [application:sessions](#) (p. 185)
- [session:by-id](#) (p. 185)
- [session:entitle](#) (p. 185)
- [session:status](#) (p. 186)
- [session:terminate](#) (p. 186)
- [Common Link Relations](#) (p. 187)

appstream:applications

You can get the [Applications](#) resource for all your applications by performing a GET on the href of this link.

Output

Returns the [Applications](#) resource that represents the collection of your [Application](#) resources.

application:by-id

You can use a Get request and this API to retrieve the application associated with the specified identifier.

Output

Returns the [Application](#) associated with the specified identifier.

application:create

You can use a POST request on the href of this link to add an application to Amazon AppStream. You must have previously uploaded your application's installation package to Amazon S3 and generated a pre-signed URL for the package's location in Amazon S3. For more information about generating pre-signed URLs, see [Share an Object with Others](#) in the *Amazon Simple Storage Service Developer Guide*.

Input

In order to add an application to Amazon AppStream, you must pass in the following fields during your POST request.

Input Field	Description
name	The name of the application.
description	(Optional) The description of the application. This description is used to describe the application in Amazon AppStream, it is not displayed to end users.
installerUrl	A pre-signed URL that points to the location in Amazon S3 that contains the installation package for the application. For more information on creating a pre-signed URL, see Share an Object with Others in the <i>Amazon Simple Storage Service Developer Guide</i> .
installerParameters	The command to run to install the application on an Amazon AppStream host managed by Amazon AppStream. This should include all necessary command-line parameters.
launchCommand	The command to run in order to launch the application after it is hosted on Amazon AppStream. This should include all necessary command-line parameters.
logBucket	The name of the Amazon S3 bucket where the .zip files contains the logs are uploaded. The bucket must already exist and allow Amazon AppStream to use the <code>PutObject</code> method.
logPaths	<p>A string array whose elements contain the log filenames or directory path where logs are stored. The filenames and paths must meet the following requirements:</p> <ul style="list-style-type: none"> Log files must be located in the <code>C:\</code> directory. Log paths must be fully qualified paths. Do not use relative paths. You can use wildcards. Log paths must be less than the path length limits. See Naming Files, Paths, and Namespaces. Log file names cannot contain Windows reserved characters. See Naming Conventions. Your application must have read access to the paths. <p>Note The total size of logs in the .zip file is subject to the Amazon S3 key size limits.</p>

Output

The newly created [Application](#).

Example Request

```
POST /applications HTTP/1.1
Host: appstream.us-east-1.amazonaws.com
Accept: application/hal+json
Version=4
X-Amz-Algorithm=AWS4-HMAC-SHA256
X-Amz-Credential=...%2Fus-east-1%2Fappstream%2Faws4_request
X-Amz-Date=2013-11-06T19%3A18%3A42.323Z
X-Amz-SignedHeaders=content-type%3Bhost%3Bx-amz-date
X-Amz-Signature=...
{
  "name": "SampleApp",
  "description": "A sample application hosted on Amazon AppStream.",
  "installParams": " ",
  "installS3Bucket": "https://s3-us-west-2.amazonaws.com/stx-server-client-bundles/WhitewaterServer-04-20-13.EXE",
  "launchCommand": "c:\\app\\SampleApp\\sampleapp.exe"
  "logBucket": "MyS3Bucket",
  "logPaths": [ "C:\\MyLog1.log", "C:\\MyLog2.log", C:\\myapp\\logs\\*logs]
}
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
{
}
```

application:update

You can use a POST request on the href of this link to update the metadata fields associated with an application hosted on Amazon AppStream. Updating the state affects the active applications. Updating the state does not affect the active sessions.

Input

In order to update the state of an application, you must pass in the following field during your POST request.

Input Field	Description
name	The new name of the application.
description	The new description of the application. This is used internally by Amazon AppStream and not displayed to end users.
launchCommand	The new command to launch the application. This command is run on the streaming server after it finishes allocating. The command starts the application on the server.

Input Field	Description
logBucket	The name of the Amazon S3 bucket where the .zip files contains the logs are uploaded. The bucket must already exist and allow Amazon AppStream to use the <code>PutObject</code> method.
logPaths	<p>A string array whose elements contain the log filenames or directory path where logs are stored. The filenames and paths must meet the following requirements:</p> <ul style="list-style-type: none"> Log files must be located in the <code>C:\</code> directory. Log paths must be fully qualified paths. Do not use relative paths. You can use wildcards. Log paths must be less than the path length limits. See Naming Files, Paths, and Namespaces. Log file names cannot contain Windows reserved characters. See Naming Conventions. Your application must have read access to the paths. <p>Note The total size of logs in the .zip file is subject to the Amazon S3 key size limits.</p>

Output

The updated [Application](#).

application:archive

You can use a PUT request and this API to archive an application and release the AWS resources allocated to host it. Any sessions that are currently active on the application will continue, but the application does not accept new sessions. When all the currently active sessions have concluded, the application reaches the `Archived` state.

An application that is archived can later be restarted with [application:reactivate](#).

Output

The updated [Application Status](#). This resource will initially have a state of `Archiving`, but the state will eventually be `Archived`. You can periodically poll this resource to see its progress.

application:reactivate

You can use a PUT request and this API to restore a previously archived application and make it ready to accept client sessions.

Output

The updated [Application Status](#). This resource will initially have a state of `Building`, but the state will eventually be `Active`. You can periodically poll this resource to see its progress.

application:delete

To delete an [Application](#) that is in the `ERROR` state, call the href in the link using the DELETE method.

Output

Returns the [Applications](#) resource.

application:status

You can use a GET request on the href of this link to retrieve the [Application Status](#) associated with the current [Application](#).

Output

The [Application Status](#) for this [Application](#).

application:errors

You can use a GET request on the href of this link to retrieve the [Application Errors](#) associated with the current [Application](#).

Output

The [Application Errors](#) for this [Application](#).

application:sessions

You can GET the href of this link to retrieve the [Sessions](#) resource for associated with the current application.

Output

The associated [Sessions](#) resource.

session:by-id

You can use a GET request with this API to retrieve the [Session](#) associated with the specified identifier.

Output

The [Session](#) associated with the specified identifier.

session:entitle

You can use a POST request and this API to entitle a new session for the current application. Amazon AppStream returns the identifier of the newly created session. The new session contains the `entitlementUrl` property which your entitlement service should pass back to the client. The client will then use this URL to redeem its entitlement from Amazon AppStream and get the IP address it will use to connect to the application.

Links

Relation	Description	Methods	Templated
self	The session:entitle link relation.	GET	No.

Input

When you entitle a new session, you can pass in the following field during your POST request.

Input Field	Description
opaqueData	(Optional) Data to pass to the application. This data is not used by the client, entitlement service, or Amazon AppStream, it is used by the application. An example of opaqueData would be a user identifier, which the application would then use to load previous state information (such as high score or current map level) for that user from a database.

Output

The newly entitled [Session](#) (p. 179).

session:status

You can use a GET request on the href of this link to retrieve the [Session Status](#) associated with the current [Session](#).

Output

The [Session Status](#) for this [Session](#).

session:terminate

You can use a PUT request and this API to terminate a session entitlement and return the session to the pool of available sessions. Any client connected to the session will be disconnected.

Links

Relation	Description	Methods	Templated
self	The session:terminate link relation.	GET	Yes. Requires the session identifier.

Output

The updated [session:status](#) (p. 186). This should be set to `Terminating` or `Terminated`.

Common Link Relations

The following link relations may be applied to [Applications](#) or [Sessions](#) depending on the context of the Amazon AppStream request. For more information about common link relations, see <http://www.iana.org/assignments/link-relations/link-relations.xhtml>.

Topics

- [collection](#) (p. 187)
- [first](#) (p. 187)
- [item](#) (p. 187)
- [next](#) (p. 187)
- [up](#) (p. 187)

collection

The collection link relation identifies a target resource that represents a collection of which the context resource is a member. For example, an application resource may have a collection link relation that points to the collection of all applications associated with your AWS account.

first

The first page of items in a collection. This can be a page of applications, sessions, or errors, depending on the context of the request.

item

An item in a collection. This can be an application, session, or error, depending on the context of the request. For example, the [Applications](#) resource has an item relation that is an array of individual [Application](#) resources.

next

The next page of items in a collection. This can be a page of applications or sessions, depending on the context of the request.

up

The parent of the current context. For example, the up link relation of the [Application Status](#) resource is the [Application](#) to which the status applies.

Product Updates

Topics

- [Release for June 20, 2014 \(Latest\) \(p. 188\)](#)
- [Release for May 30, 2014 \(p. 189\)](#)
- [Release for May 9, 2014 \(p. 189\)](#)
- [Release for April 22, 2014 \(p. 190\)](#)
- [Release for March 28, 2014 \(p. 191\)](#)
- [Release for March 7, 2014 \(p. 191\)](#)
- [Release for February 14, 2014 \(p. 192\)](#)
- [Release for January 24, 2014 \(p. 192\)](#)

Release for June 20, 2014 (Latest)

This release includes the following changes:

Up to 20% performance improvement on 64-bit iOS devices

We added support for ARM64 which improves the performance on 64-bit iOS devices by up to 20%. If you are updating an existing iOS client application, you should use the new `libXStxClient.a` file and update the `<SDK_dir>/3rdparty` folder to the version in the SDK and set Architectures in your project Build Settings to `$(ARCHS_STANDARD)` to include ARM64 support.

Simplified sample streaming application code

The code for the sample streaming application has been simplified to under 600 lines, down from more than 2,000 lines. The SDK also contains a Visual Studio solution file so you just need Visual Studio 2010 or later to compile the code. In the previous SDK version, you needed both CMake for Windows and Visual Studio to compile the code.

Learn more about the simplified sample streaming application by completing [Option 3: Deploy a Streaming Application on Amazon AppStream \(p. 27\)](#).

Remote debugging in standalone mode

You can debug your streaming application running in Amazon AppStream standalone mode by using Remote Tools for Visual Studio on your Amazon EC2 instance. Learn more about remote debugging in standalone mode at [Debug Your Streaming Application in Amazon AppStream Standalone Mode \(p. 72\)](#).

Other Updates

- We fixed an issue where an invalid path specified in the log page prevents the next step in the console even if logging was not specified.

Release for May 30, 2014

This release includes the following changes:

Utilization Log is in Coordinated Universal Time (UTC)

Amazon AppStream now uses UTC time instead of epoch time in the utilization log. The UTC time allows better coordination and consistency between other AWS services and Amazon AppStream.

Utilization Log Contains GPU Metrics

You can now see GPU metrics in the utilization log when you select logging for your streaming application. The utilization log contains GPU metrics, such as memory information, power utilization, and temperature to help you monitor and diagnose your streaming application.

Learn more about the GPU metrics at [Enable Logging on a Streaming Application \(p. 147\)](#).

Simplified the Application Resource Properties

The applicationManifest property is no longer in the [Application \(p. 174\)](#) resource. The properties within applicationManifest property are now in the [Application \(p. 174\)](#) properties.

This change simplifies the [Application \(p. 174\)](#) resource structure.

Other Updates

- The latency rate is now equal or better than the previous SDK release. Some customers experienced increased latency rates after using the previous SDK release.
- The example Android client application no longer hangs on exit.
- Bandwidth adaptation is improved when the frame rate varies.

Release for May 9, 2014

This release includes the following changes:

Logging for Utilization Metrics

You can now see utilization metrics in the logs when you select logging for your streaming application. This comma-separated value file contains operational metrics, such as CPU utilization, memory information, and disk read and write speeds.

Learn more about the utilization metrics at [Enable Logging on a Streaming Application \(p. 147\)](#).

YUV444 Color Subsampling Support

Amazon AppStream now supports streaming video to client applications and devices at the YUV444 color subsampling rate. If your streaming application, client application, and devices support YUV444, you can specify this higher rate in your streaming application.

Learn more about YUV444 at [Stream Video to a Client \(p. 60\)](#).

Other Updates

- Amazon AppStream now has less server overhead, giving more CPU resources to streaming applications and improving performance.
- The Amazon AppStream console **Summary** page now has a link to the Amazon S3 bucket that stores your log file.
- Amazon AppStream now creates a log file when the streaming application unexpectedly stops. In the previous version, a log file was not created after the streaming application stops.
- The Amazon AppStream console features improved text.

Release for April 22, 2014

This release includes the following changes:

Standard and User-defined Logging

You can configure your streaming application to save standard and user-defined logs to an Amazon S3 bucket. When the streaming application terminates, Amazon AppStream collects the logs into a `.zip` file and then uploads this file to your own bucket or to a bucket that Amazon AppStream creates.

Learn more about the logging feature at [Enable Logging on a Streaming Application \(p. 147\)](#).

Updated OpenSSL Version

We updated the OpenSSL cryptography library in Amazon AppStream to a version that fixes the Heartbleed security bug.

Other Updates

- We now report an error if the application installer fails in the middle of the process. Previously, we did not report an error when this condition occurs.

Release for March 28, 2014

This release includes the following changes:

New Billing Practice

Your bill now includes session time up to the point when your streaming application unexpectedly terminates. Previously, your bill did not include any session in which your streaming application unexpectedly terminated.

SDK Version Detected from the Files instead of the Registry

The version of the Amazon AppStream SDK that you use is based on the files rather than a registry setting. This change ensures that Amazon AppStream implements the correct version for your streaming application.

Other Updates

- The **Cancel** button on the console now takes you to the **Welcome** page.

Release for March 7, 2014

This release includes the following changes:

Amazon AppStream Is Available to Anyone with an AWS Account

During Limited Preview, only a limited number of users had access to Amazon AppStream. Now allow anyone with an AWS account can try Amazon AppStream.

Improved Deployment Error Messages

Error messages now display the reason why a deployment failed. The error messages are documented in [Error Codes](#) (p. 160).

Example Client Application for Mac OS X

The Amazon AppStream SDK now includes a precompiled client application and the source code to build your own client application for Mac OS X.

Learn more about creating a sample application client in [Build a Client for OS X](#) (p. 101).

Increase Your Service Limits

By default, you can have up to ten simultaneous sessions connect to your streaming application. When you want your streaming application to accept more simultaneous sessions, you can request to increase the service limit.

Learn more about increasing your service limits at [Increase Your Amazon AppStream Service Limits \(p. 152\)](#).

Other Updates

- When you delete a streaming application from the **Streaming Application** page, it no longer appears there. Previously, a deleted streaming application remained in the **Streaming Application** page with a **Deleted** status.
- Amazon AppStream displays an improved error message it cannot access the pre-signed Amazon S3 URL to the application installer. Previously, Amazon AppStream displayed a generic error message that did not define the problem.
- The Amazon AppStream console now correctly displays the launcher command.
- Amazon AppStream now displays an error if the streaming application stops after the client connects. Previously, this condition did not trigger an error message.

Release for February 14, 2014

This release includes the following changes:

Improved Deployment Experience

The Amazon AppStream console now displays a review page that shows your settings prior to deployment. On this page you can change the settings before deployment so that your streaming application will start correctly.

Silent Installer Requirement

To install your streaming application, you need to create an installer that installs your streaming application and dependency files without any user intervention.

Learn more about the silent installer requirement at [Build an Application Installer \(p. 73\)](#).

Other Updates

- Amazon AppStream displays an error message if the application installer you uploaded to the Amazon S3 bucket does not comply with the requirements in [Build an Application Installer \(p. 73\)](#).
- A race condition no longer results from errors thrown by the `GetEntitlement` function.

Release for January 24, 2014

This release includes the following changes:

Single Download for the Amazon AppStream SDK

All of the different SDK packages are now available as a single downloadable package that contains the libraries for your Windows streaming application and your Android, iOS, and Windows client applications. Previously, you had to download a package for the Windows sample streaming application and client application, a package for the Android client application, and another package for the iOS client application.

Learn more at [Downloads \(p. 10\)](#).

Improved Console Load Time

The Amazon AppStream console now loads 45% faster than the previous version. The faster load time means faster deployment for your streaming application.

Support for Android Hardware Decoding

Amazon AppStream now supports hardware decoding for 2013 or later Android devices that use the Qualcomm processor, such as the Amazon Kindle and Google Nexus. The hardware decoder support means more consistent streaming performance over using the software decoder.

Other Updates

- Amazon AppStream terminates the Amazon EC2 instances if the application installer did not completely install the streaming application. Previously, the EC2 instance would start and run even if the application installer could not complete the installation.
- Error messages now follow the style of error messages in other AWS services.
- You can now register a streaming application reactivated from an archived state.
- The Amazon AppStream console now features separate text boxes for the launch parameters and the path to the launcher to prevent application installer error. Previously, you entered the launch parameters and path to the launcher in the same text box.
- Amazon AppStream now allows an instance to run even if the instance is in pre-production mode.

Document History

The following table describes the important changes to the documentation in this release of Amazon AppStream.

Change	Description	Release Date
Initial Release	This is the first release of the Amazon AppStream Developer Guide.	November 13, 2013