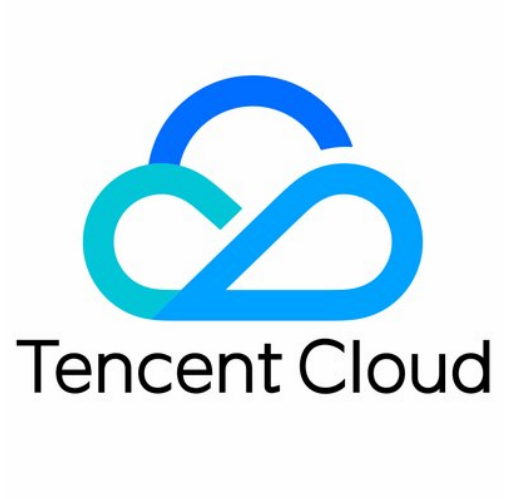
Cloud Application Rendering

Best Practices

Product Documentation



Contents

Best Practices
How to Implement Application Upload and Version Update via CAR-CLI

Last updated：2024-06-04 15:19:49

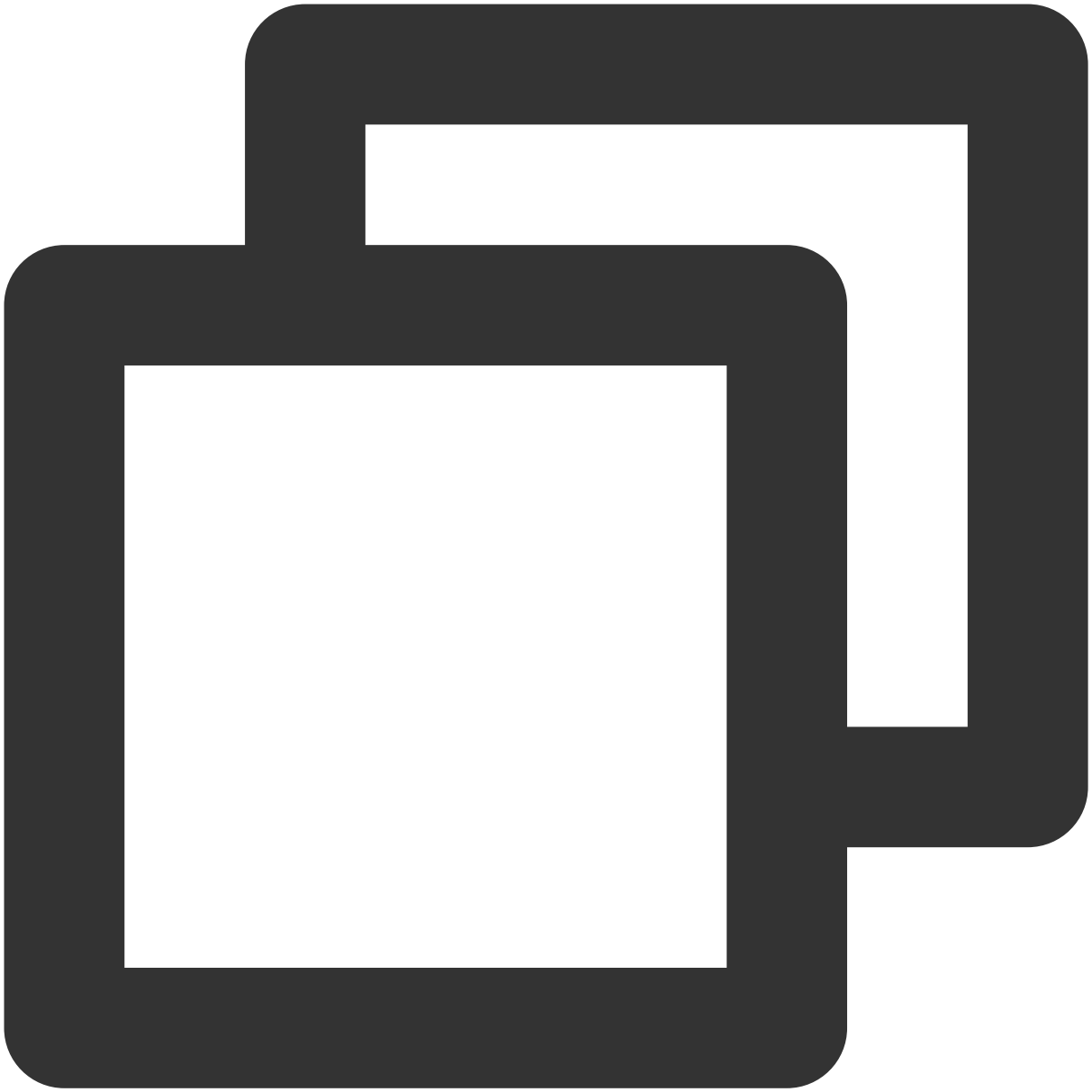
CAR-CLI is a command line tool for managing application versions. You can use CAR-CLI to quickly and easily query, upload, update, and delete application versions by calling TencentCloud APIs. This document describes the basic features of CAR-CLI and how to use it to implement an automated pipeline for updating application versions.

Basic features

Note:
You need to place the configuration file in the same directory as the CLI tool and replace with the `SecretId`, `SecretKey`, and `AppId` of your Tencent Cloud account. CAR-CLI supports the macOS, Windows, and Linux operating systems. The following examples are based on Linux.

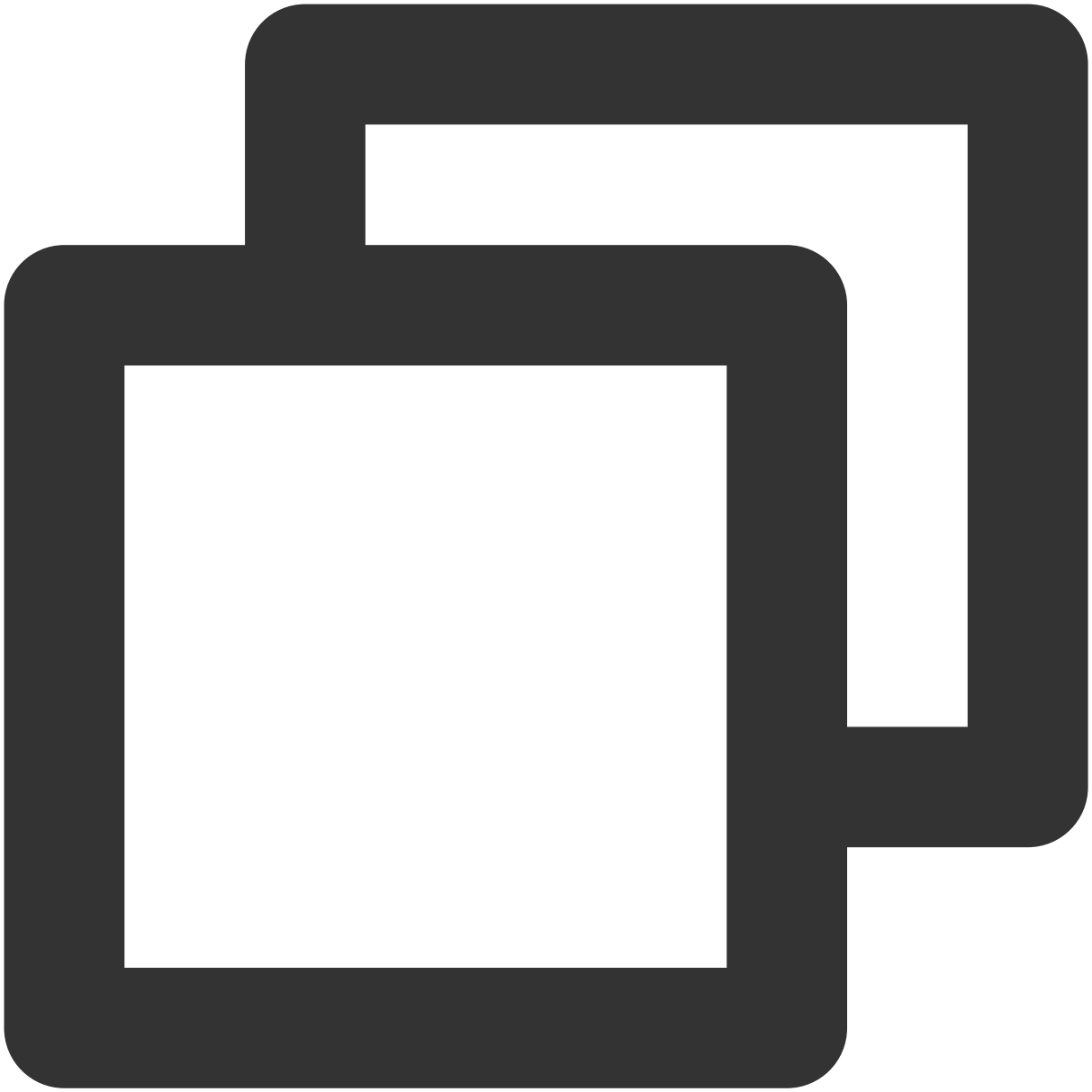
View help information

Run the following command to view the tool's help information.



```
./car help
```

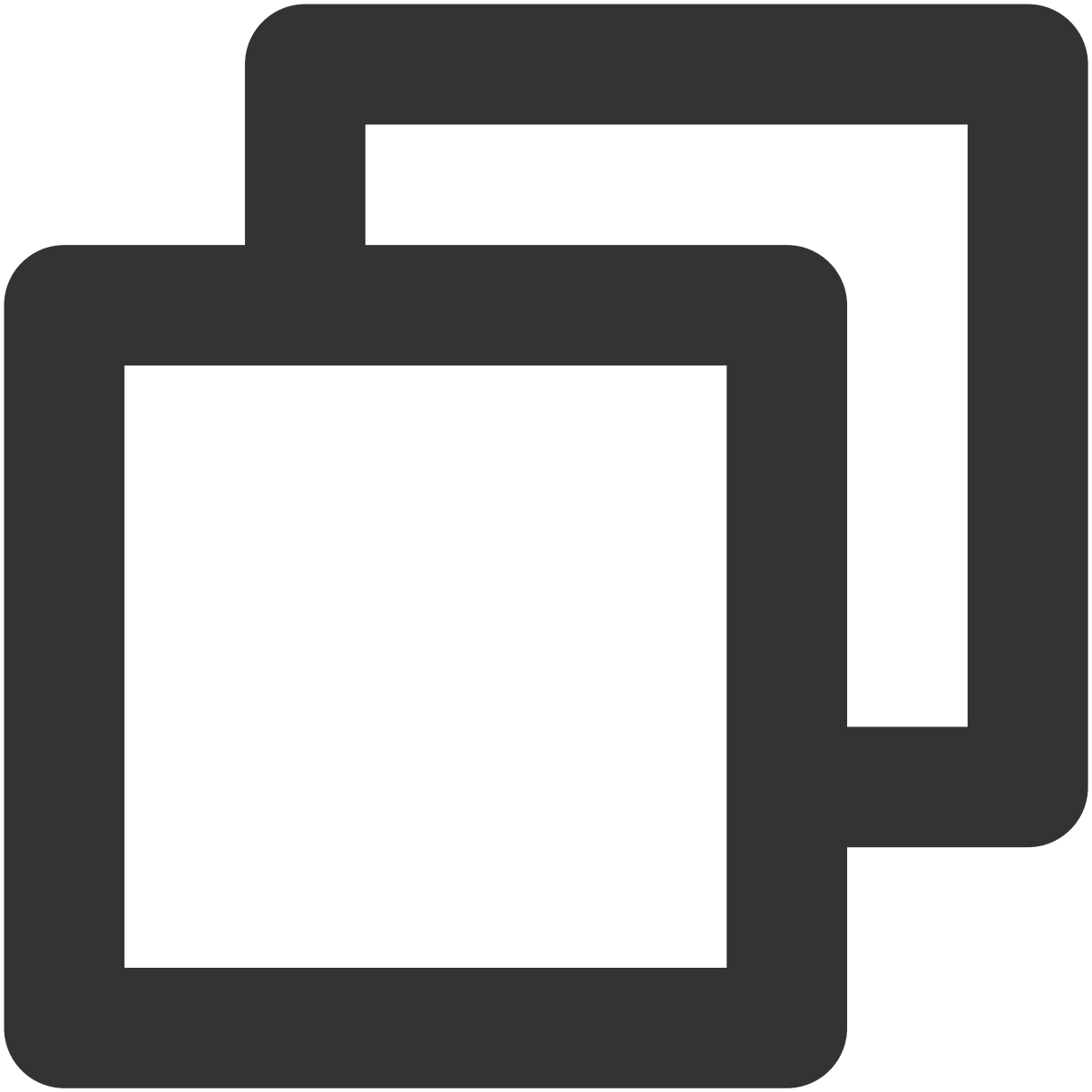
For example, run the following command to view the help information for the application creation command.



```
./car create-application help
```

Create an application

Run the following command to create an application. You need to enter the application name, which can contain up to 16 Chinese characters, letters, digits, or hyphens (-). If this command is executed successfully, the application ID of the new application will be returned.

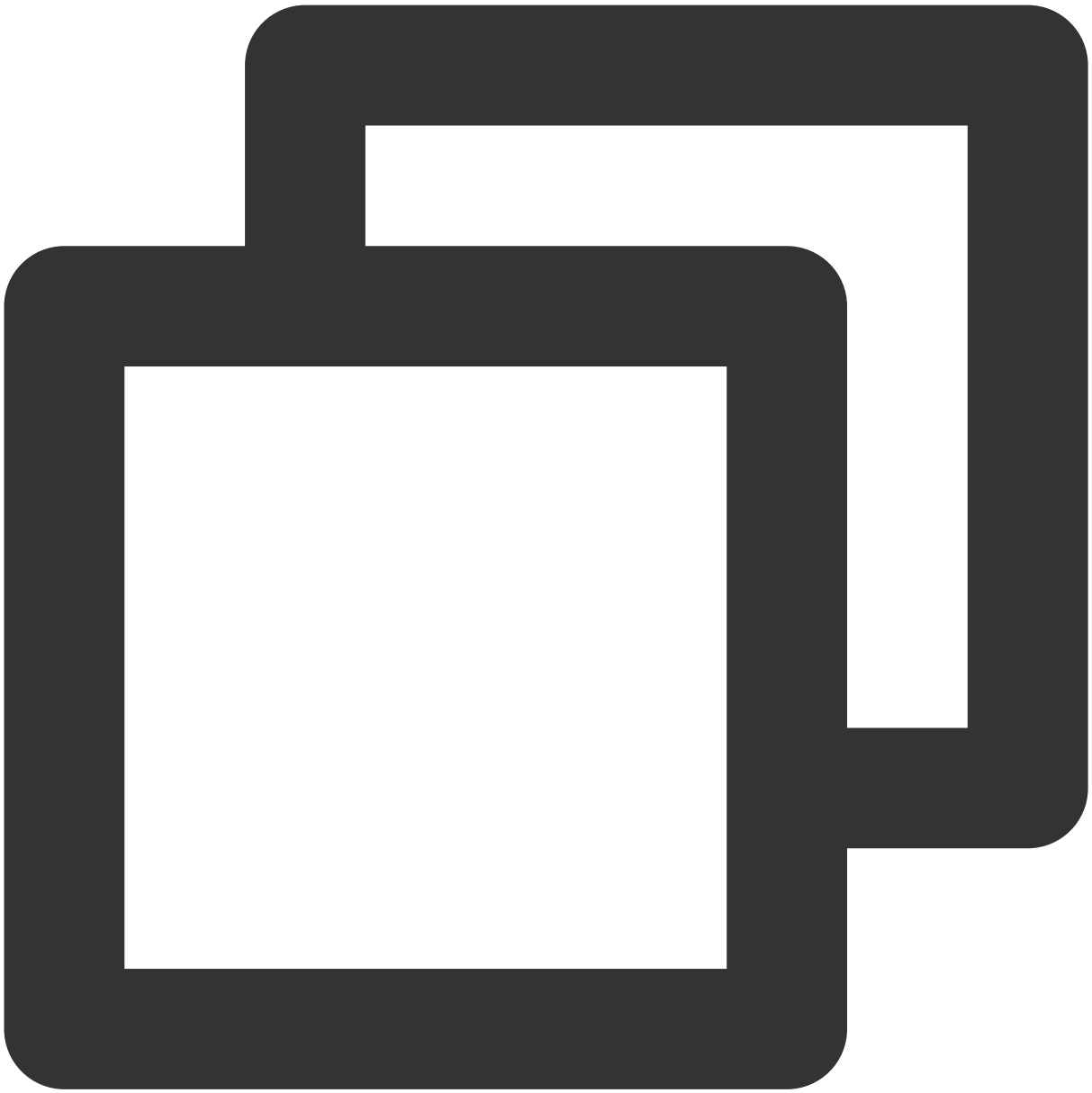


```
./car create-application --name xxx
```

Create an application version

Run the following command to create an application version. Up to five versions can be created for an application. You need to enter the application ID, version name, and application file format (ZIP, RAR, or 7z). If there are any versions in `ApplicationUpdateCreating`, `ApplicationUpdateNoReleased`, or `ApplicationUpdateCreateFail` status, this command will be rejected. After the version is created

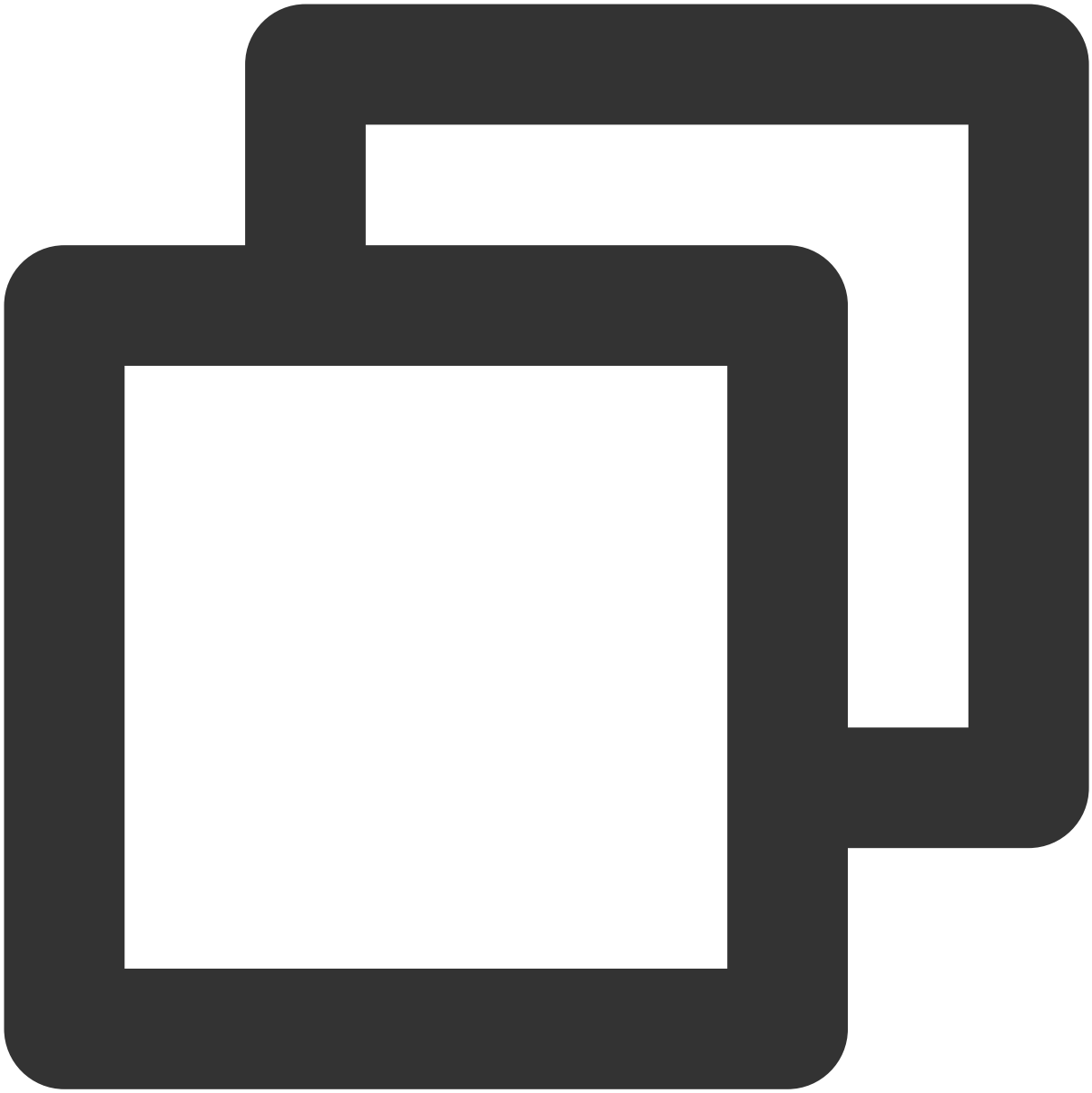
successfully, you also need to upload the application version file and release the new version. If this command is executed successfully, the application version ID will be returned.



```
./car create-application-version --app-id app-xxx --name xxx --type zip
```

Upload an application version file

Run the following command to upload an application file for a new version. You need to enter the application ID, local path of the application file, and version ID. If the upload fails during the creation of an application or an application version, you can still proceed with this command. You must ensure that the local file path is not changed.



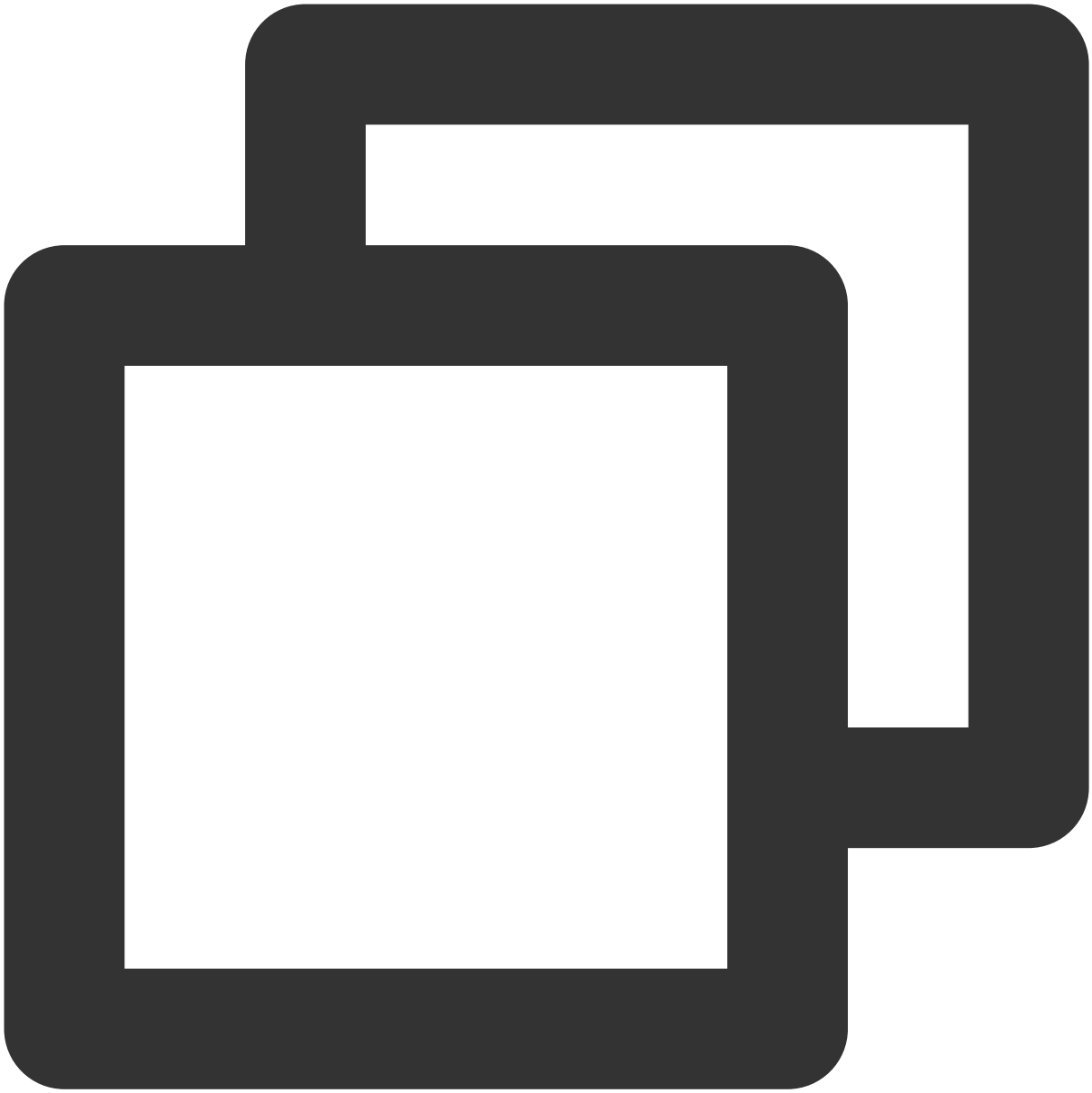
```
./car upload-application-version-file --app-id app-xxx --path /data/xxx.zip --versi
```

Note:

After the application version file is uploaded successfully, if its version name and format are different from those used for creating an application version, the version name and format will be replaced with those of the uploaded file.

Release an application version

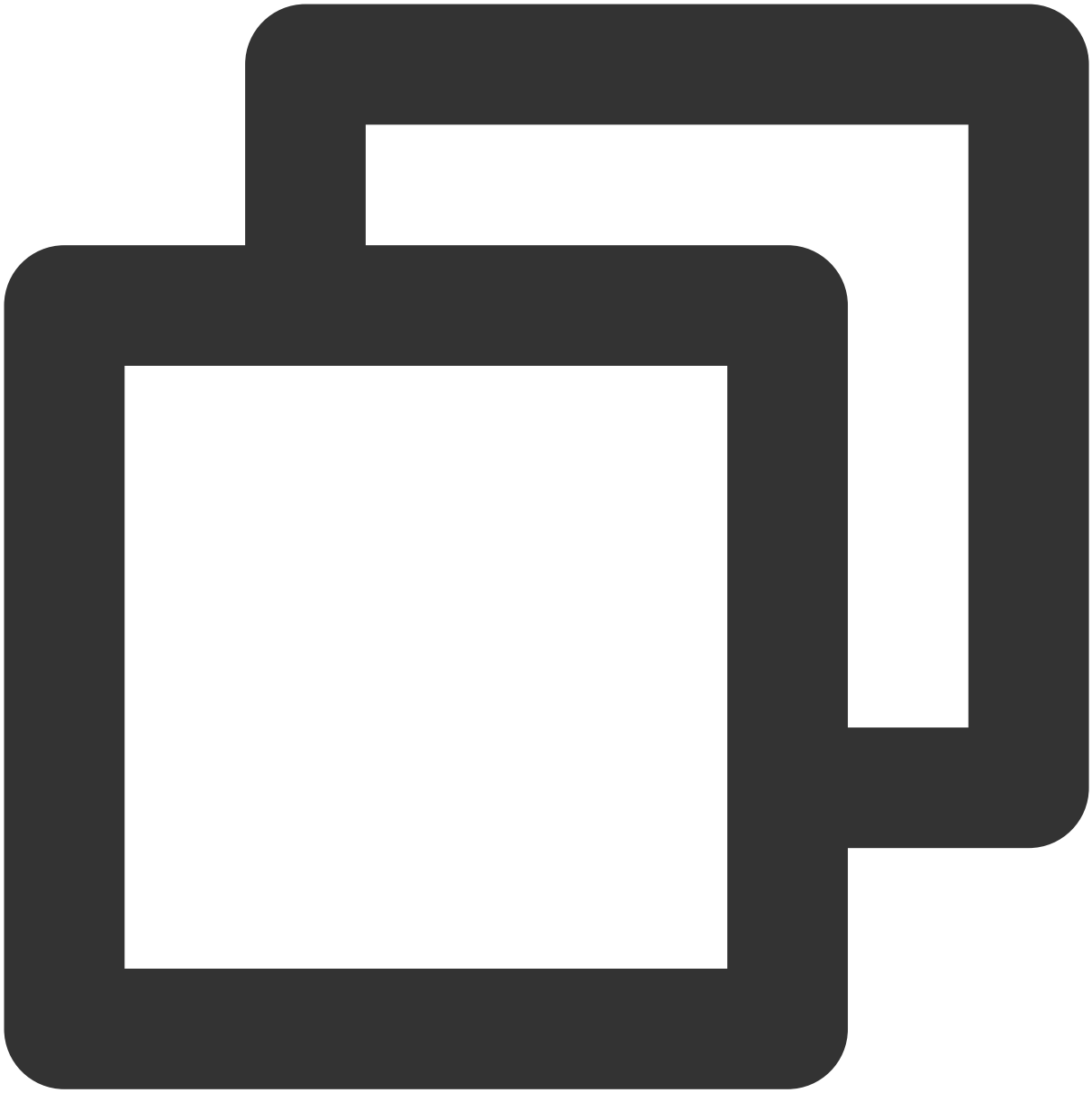
Run the following command to release an application version. You need to enter the application ID and version ID.



```
./car set-version-online --app-id app-xxx --version-id ver-xxx
```

Delete an application version

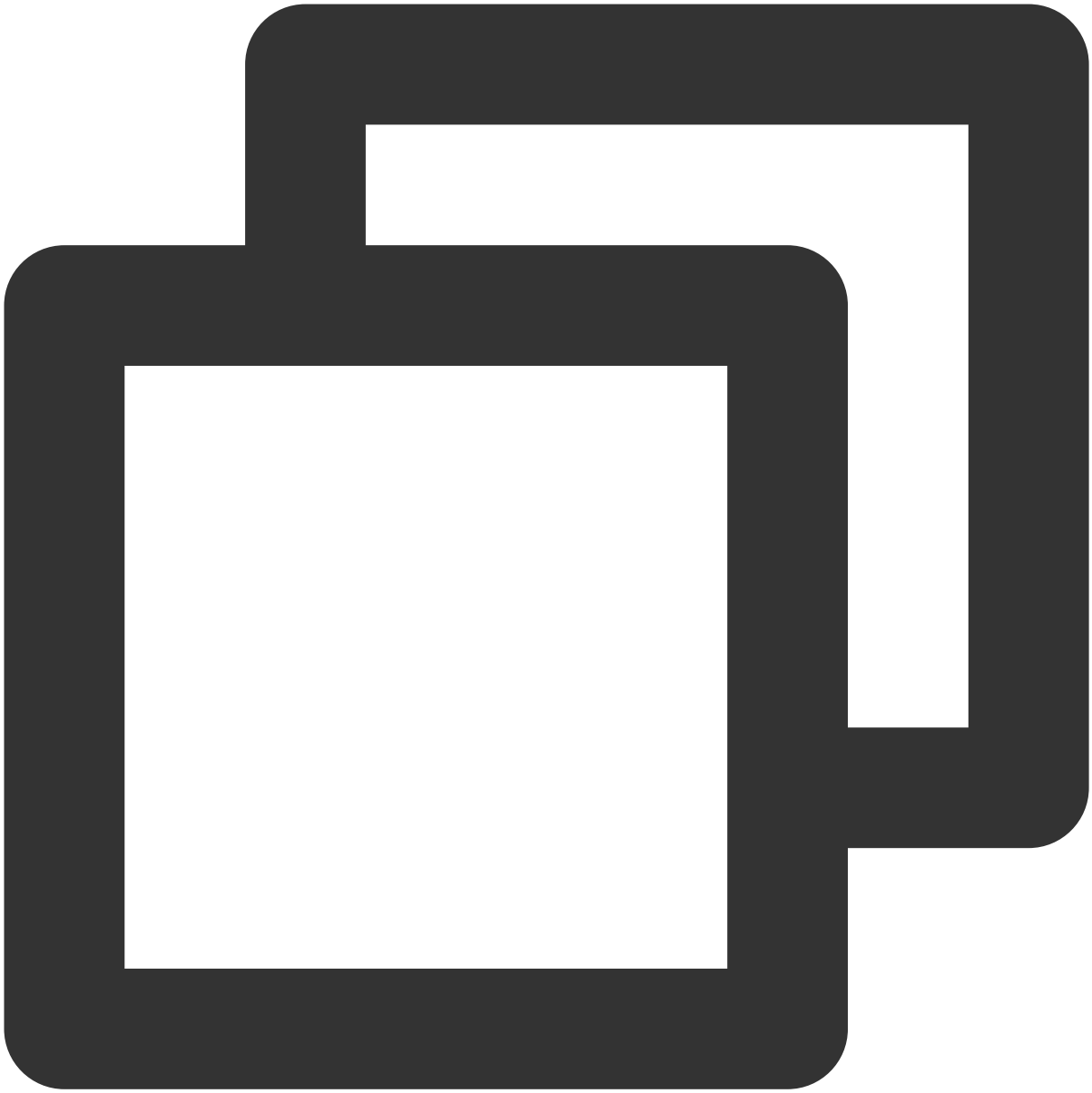
Run the following command to asynchronously delete an application version. You need to enter the application ID and version ID. To check whether the version has been deleted successfully, you can query the list of application versions.



```
./car delete-application-version --app-id app-xxx --version-id ver-xxx
```

Display the list of application versions

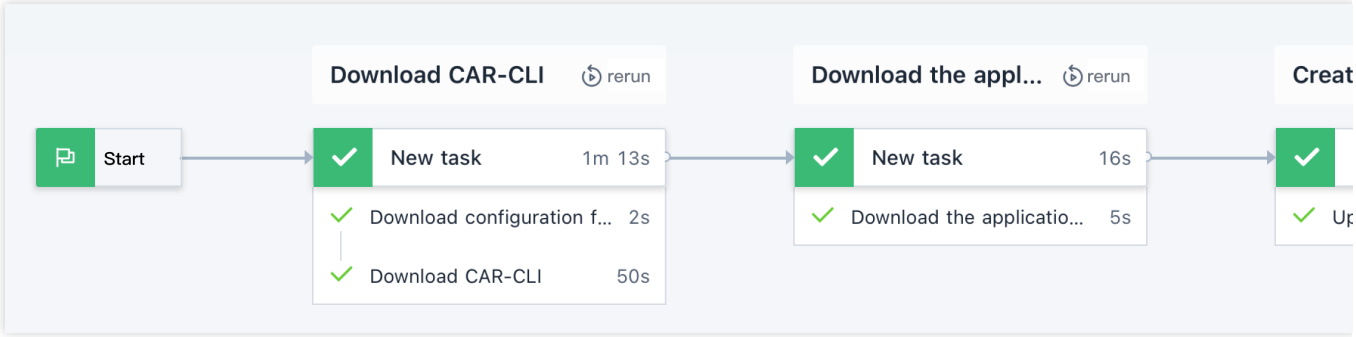
Run the following command to display the list of application versions. You need to enter the application ID. To obtain the output version ID and version status of the oldest version, use the `grep` and `awk` commands.



```
./car describe-application-version --app-id app-xxx
./car describe-application-version --app-id app-xxx | grep -v "Inuse" | awk '{print
```

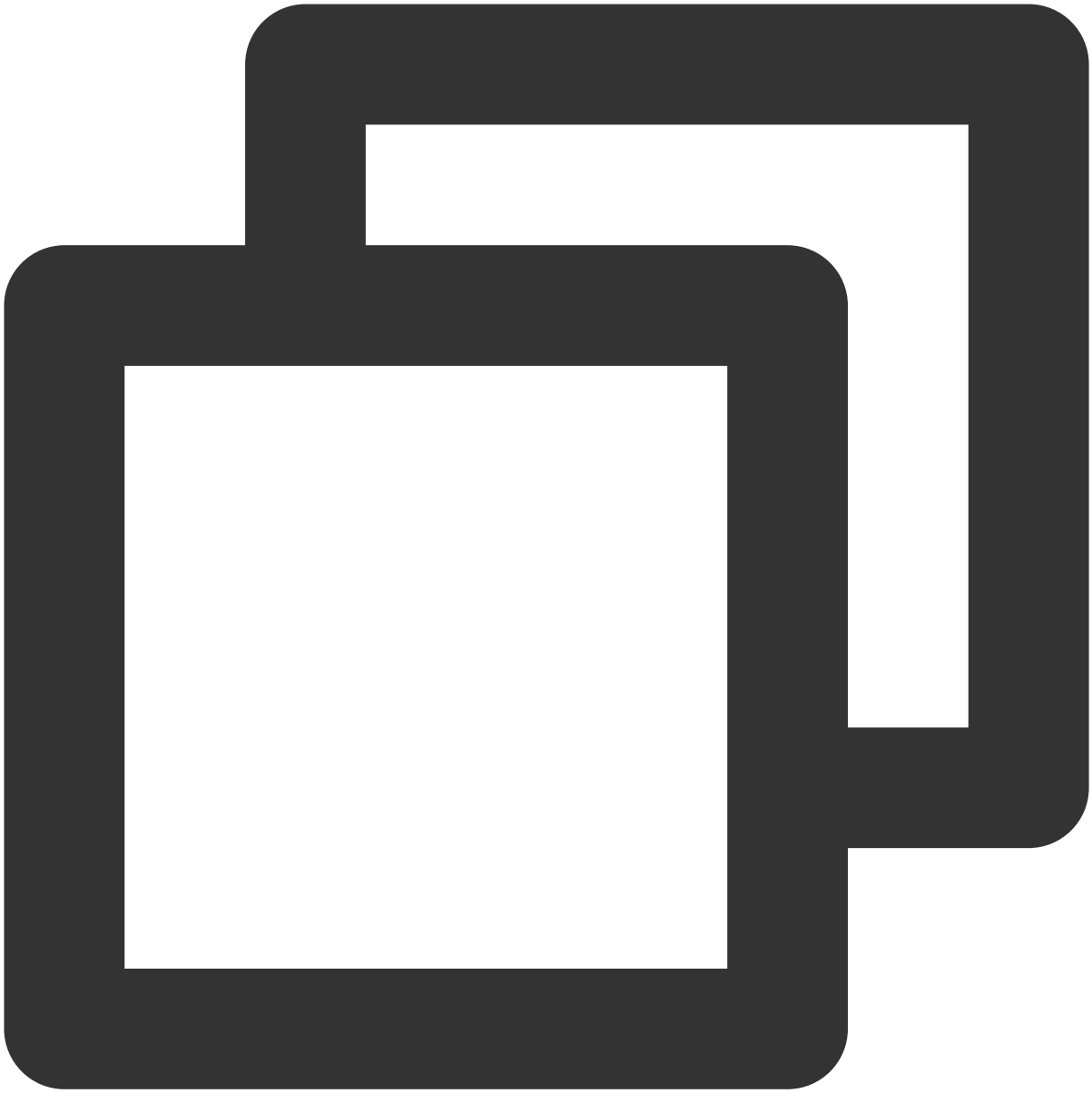
Pipeline for updating application versions

This section uses a Tencent Cloud CODING pipeline as an example to show you how to implement a pipeline for updating application versions.



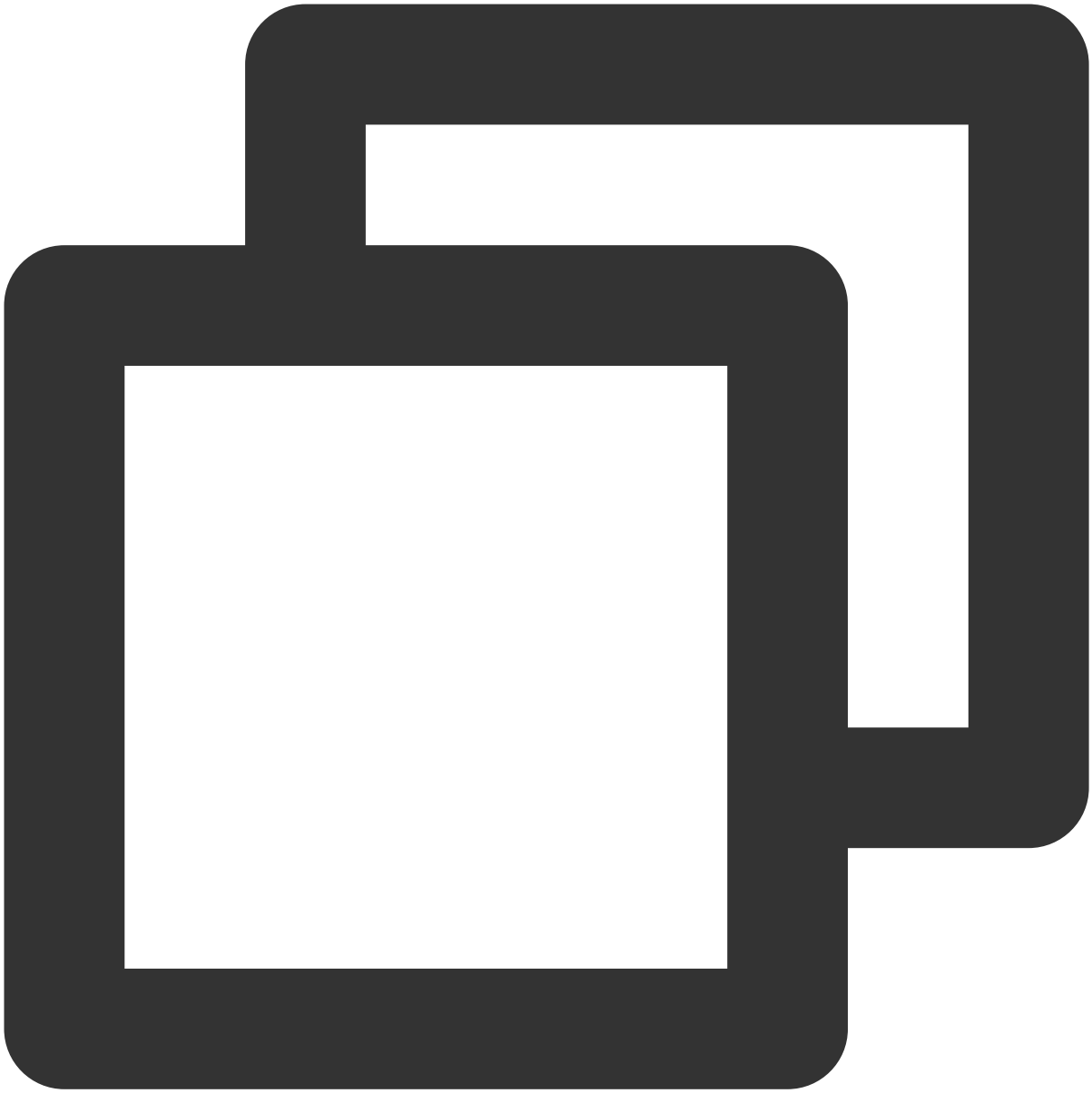
Download CAR-CLI

Load the remote configuration file into the current working directory.



```
cd workdir

wget "your config file url"
```

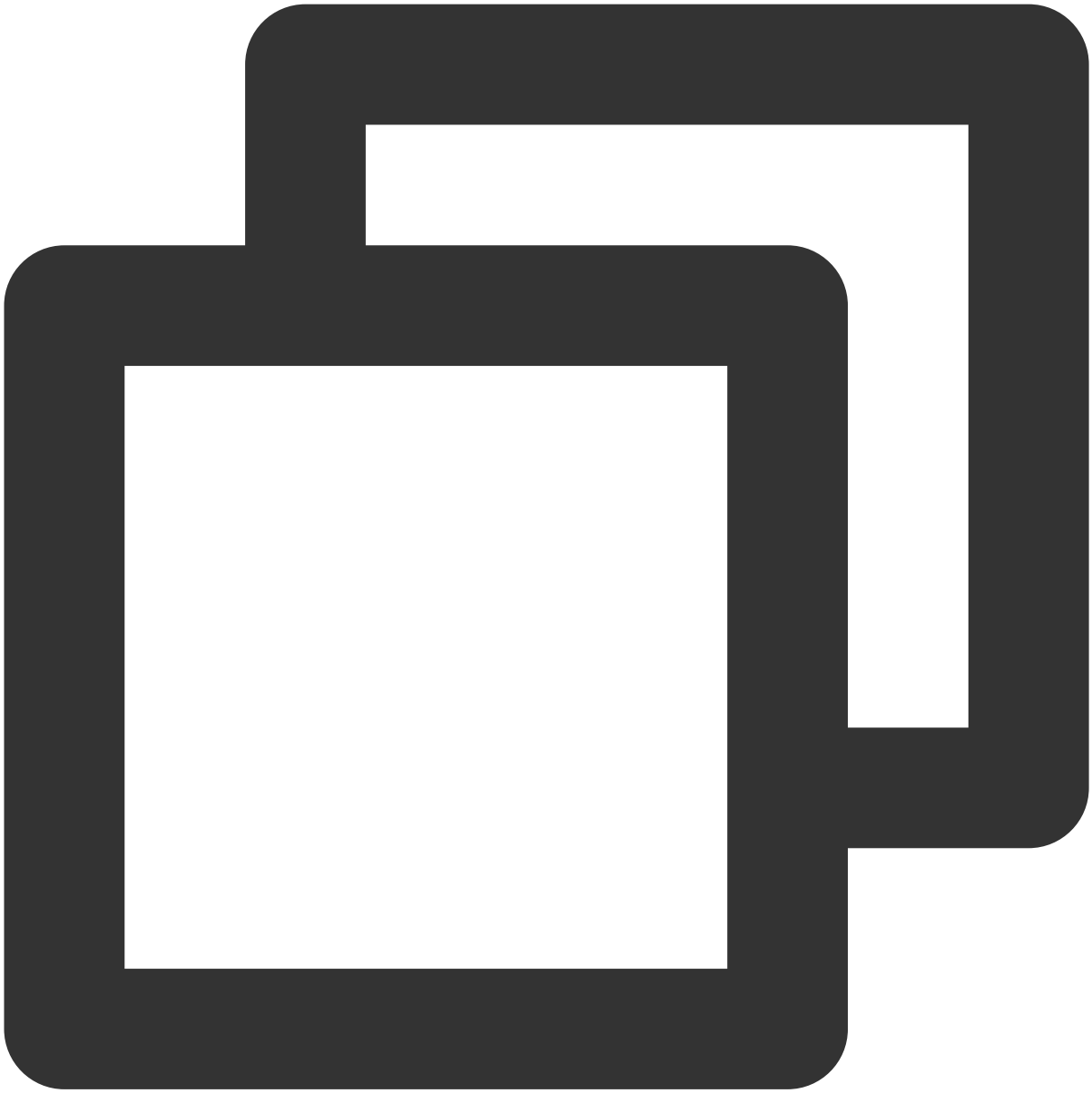
Run the following scripts to download the CAR-CLI tool.



```
cd workdir

mkdir pkg && cd pkg
wget https://github.com/tencentyun/car-cli/releases/download/v1.0.0/car.zip
unzip car.zip
mv ./car/linux/car ../
cd ..

chmod +x car
```

Download the application file of the new version

Configure the following scripts to download the application file of the new version.



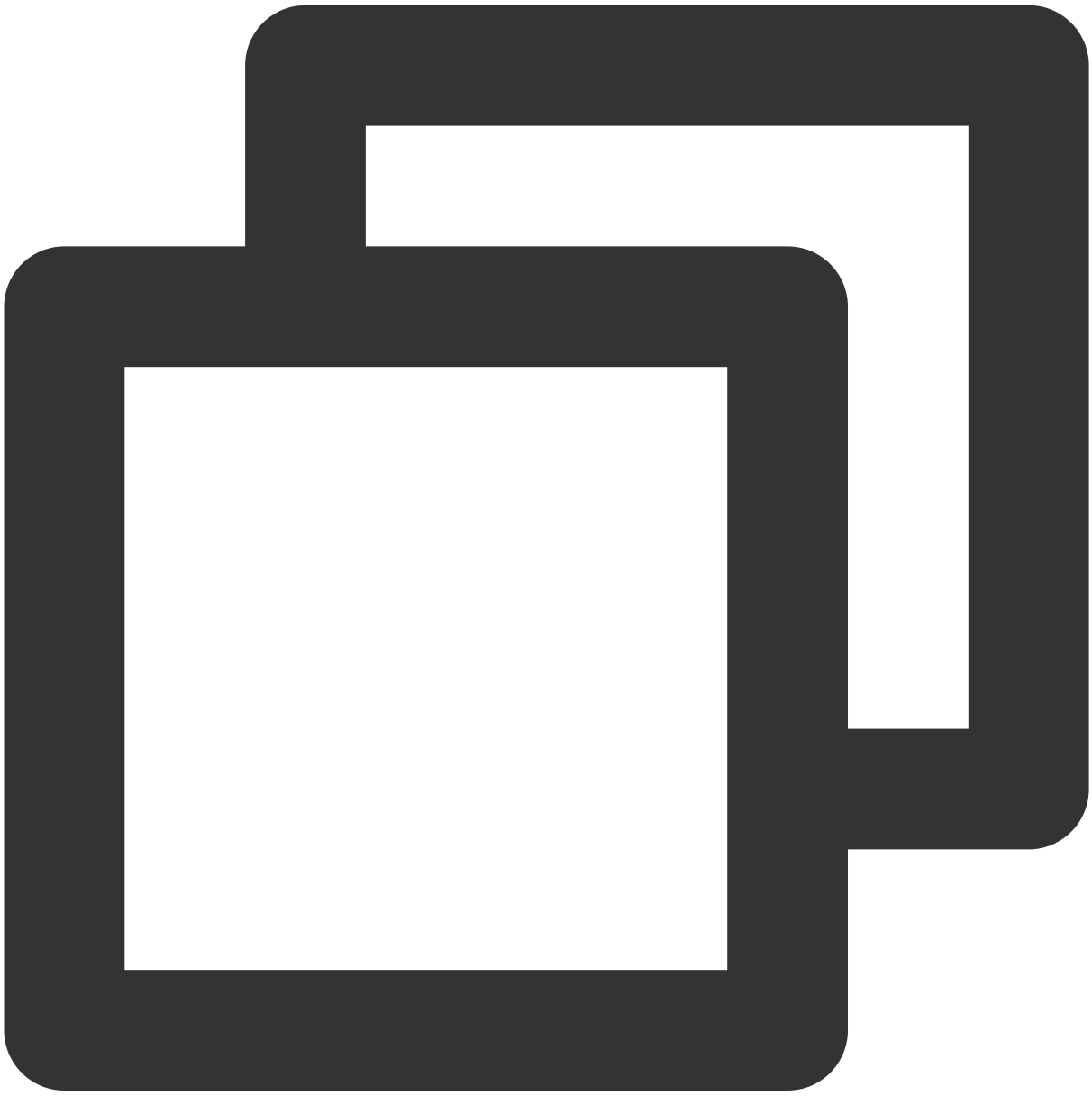
```
cd workdir

wget $PackageURL       # Configure `PackageURL` in pipeline environment variables
```

Create an application version

Create a new version for an existing application, upload the new application version file, and finally release the new version.

Note:

If the number of application versions exceeds 5, this script will automatically delete the oldest versions which are not in use.



```
cd workdir

# Query the list of application versions
output=$(./car describe-application-version --app-id $ApplicationID)   # Configure
```

```
lineCount=$(echo "$output" | wc -l)

# If the number of application versions exceeds 5, delete the oldest versions which
if [ $lineCount -ge 5 ];then
    versionID=$(echo "$output" | grep -v "Inuse" | awk '{print $1}' | head -n 1)
    ./car delete-application-version --app-id $ApplicationID --version-id $versionI
fi

# Deleting an application version is an async operation, so you need to regularly c
waitTimes=0
while [ $lineCount -ge 5 ]
do
    output=$(./car describe-application-version --app-id $ApplicationID)   # Replac
    lineCount=$(echo "$output" | wc -l)

    waitTimes=$((waitTimes+1))
    if [ $waitTimes -gt 20 ]
    then
        echo "Error: Waiting too long to delete application version."
        exit 1
    fi

    sleep 1
done

# Query the application file name and type by `PackageURL`
fileName=$(basename $PackageURL)          # Configure `PackageURL` in pipeline envir
echo $fileName
fileType="${PackageURL##*.}"
echo $fileType

# Create an application version
output=$(./car create-application-version --app-id $ApplicationID --name $fileName

# Upload the new application version file to the cloud
./car upload-application-version-file --app-id $ApplicationID --version-id $output

# Release the new version
./car set-version-online --app-id $ApplicationID --version-id $output
```

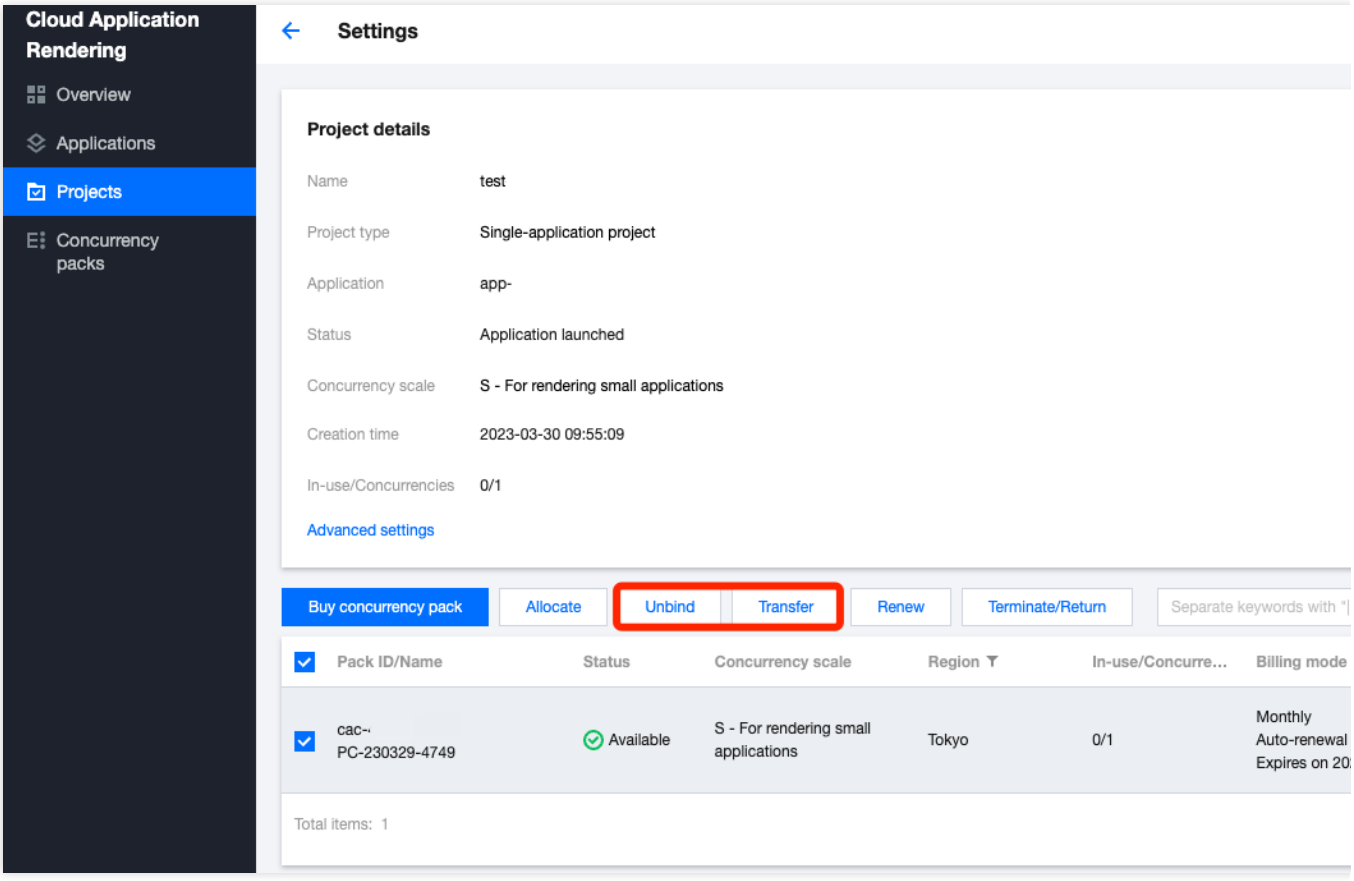
How to Implement Concurrency Sharing

Last updated：2024-01-26 12:00:38

CAR allows you to flexibly allocate and share concurrencies so as to reduce costs.

Option 1. Concurrency transfer

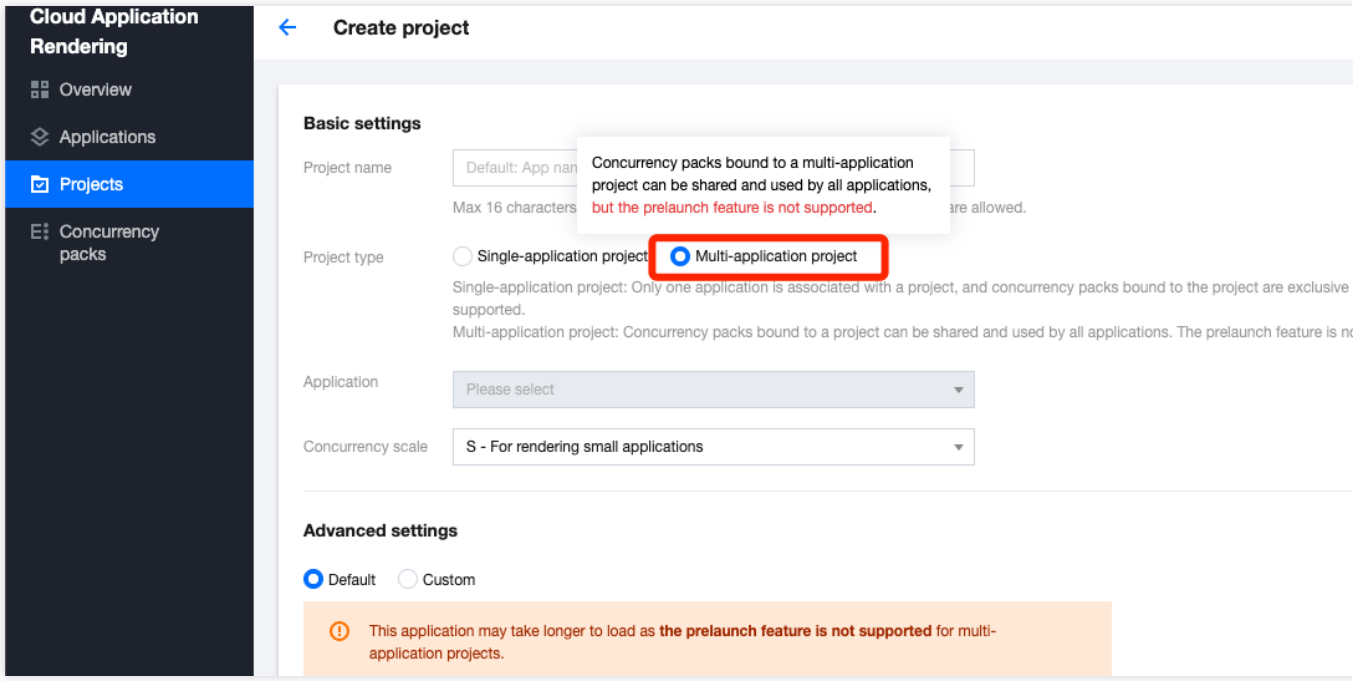
CAR concurrencies are decoupled from applications. Assume that you have projects A and B and there is a pack of 100 concurrencies under project A. If project A is removed or does not need so many resources, you can unbind the concurrency pack from project A and transfer it to project B in the Projects page of the console at any time.



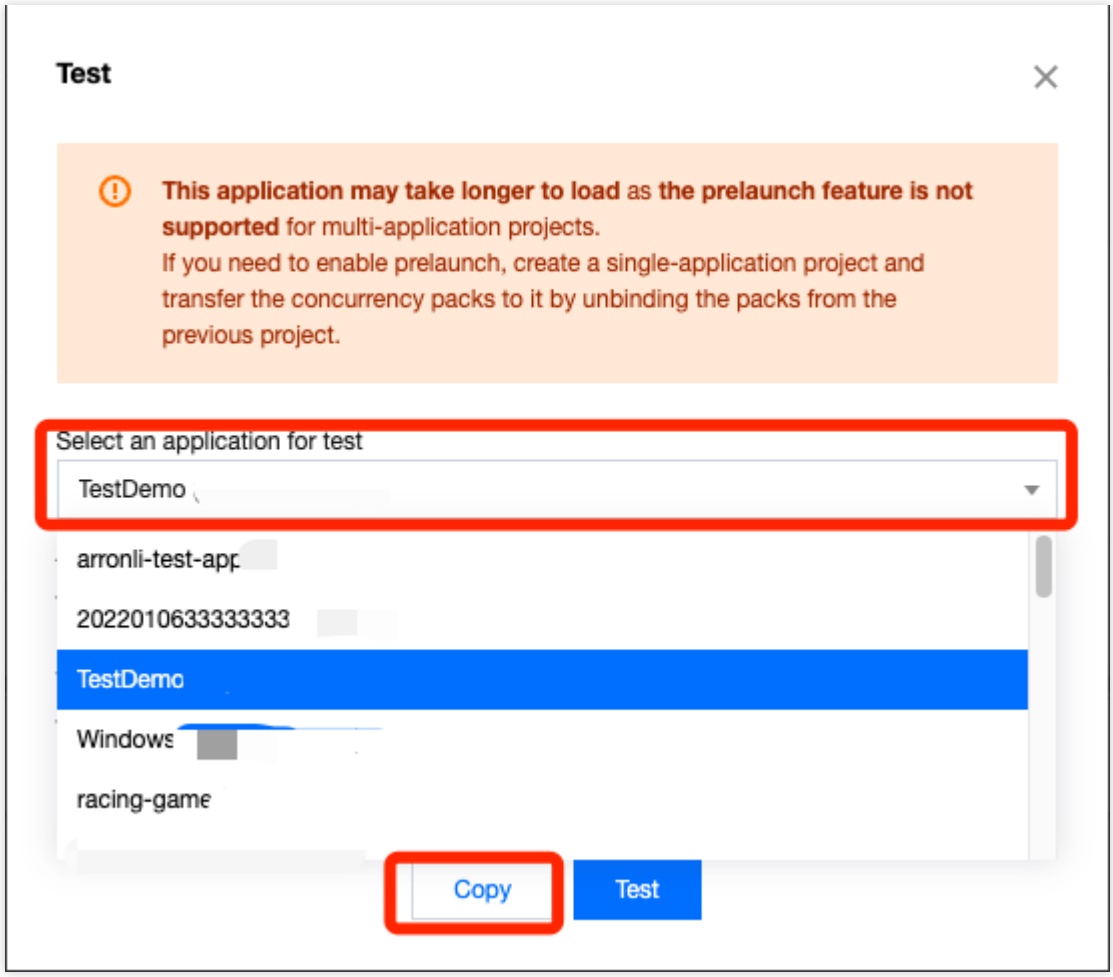
Option 2. Multi-application projects

CAR supports two types of projects as detailed in Appendix: Detailed description of project types.

Single-application project: The project is associated with only one application. Concurrency packs under the project can be used by the application. The availability of resources is guaranteed, and you can enable the prelaunch feature to quickly load the application when a user connects to it.
Multi-application project: Like a resource pool, concurrency packs under the project can be shared by all applications. When a user makes a request to connect to an application, idle concurrencies can be scheduled from the pool in real time.
The following example shows how to use a multi-application project to implement concurrency sharing.
1. Go to the Projects page in the console and create a multi-application project.



2. In the **Projects** page, find the project and click **Test** under the **Operation** column. Then select the target applications to generate test passwords and save the generated information locally.
3. Start a test. If you select the passwords generated for different applications, the corresponding applications will be started, but they all will use the concurrencies bound to the same project.



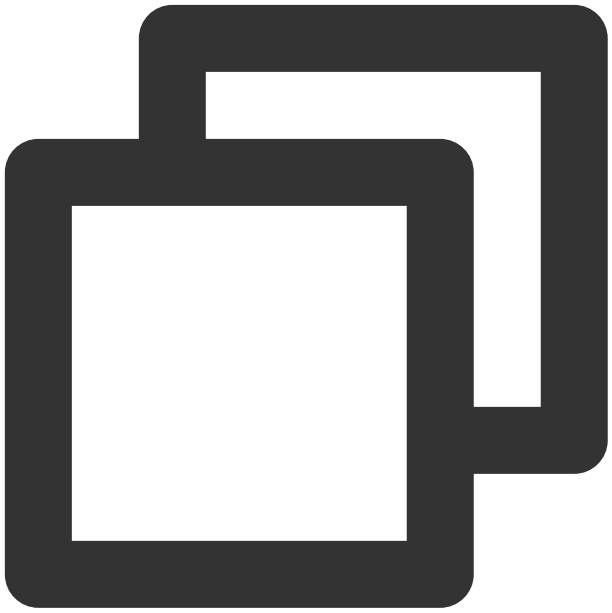
Note:

A test password is valid for seven days. Each time you click **Test**, a new password will be generated, but the previously generated passwords are still valid if they haven't expired.

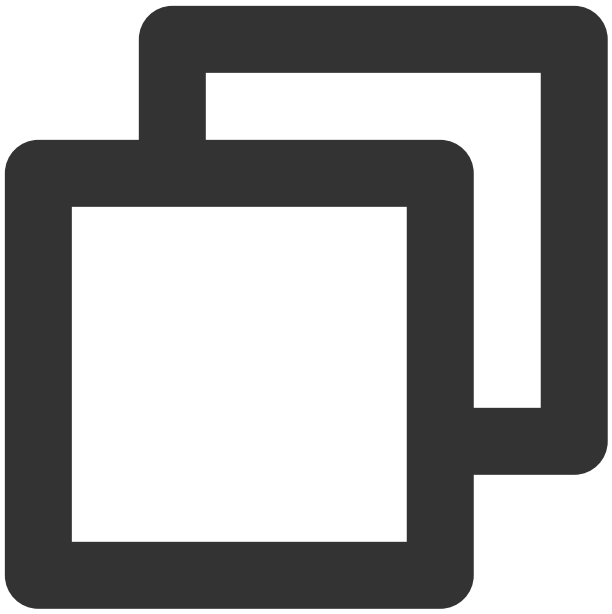
While a password is still valid, you don't need to generate a new one to perform operations such as purchasing a concurrency pack and updating the application version.

4. If you want to integrate the SDK (for the integration demo and directions, see Step 3. Integrate CAR PaaS), when you call ApplyConcurrent to request a concurrency, you need to pass in both the `ProjectId` and `ApplicationId` parameters.

Assume that the ID of the multi-application project is `cap-abcdefgh` and applications A (`app-12345`) and B (`app-67890`) need to share resources. Taking the integration demo for JavaScript as an example, you need to pass in the following parameters for the frontend of applications A and B respectively:



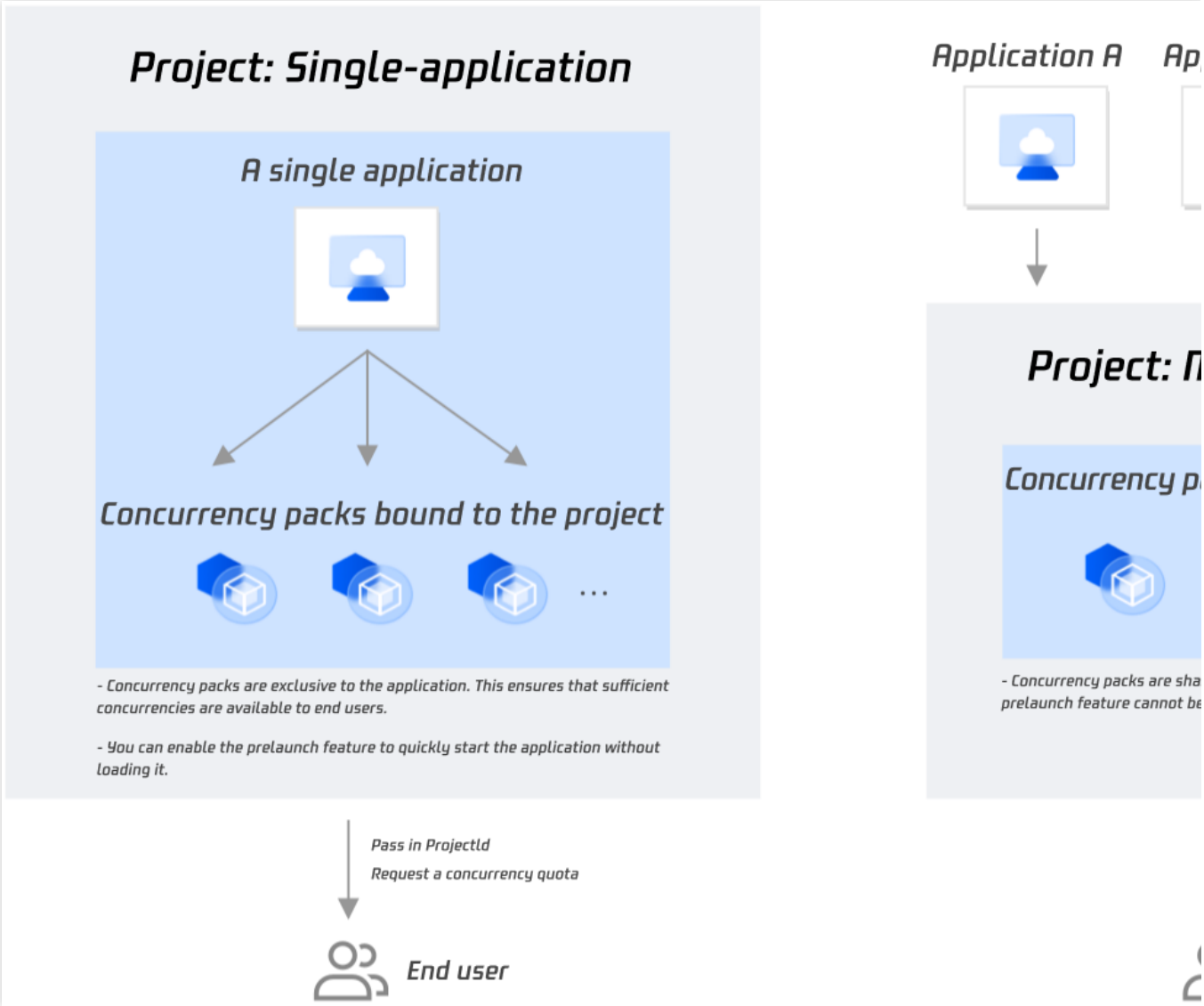
```
//Application A

const { data } = await axios.post(url, {
    ProjectId: 'cap-abcdefgh',
    ApplicationId: 'app-12345',
    UserId: 'user-id', //Random UserId
    ClientSession: TCGSDK.getClientSessic
});
```



```
//Application B
```

```
const { data } = await axios.post(url, {
    ProjectId: 'cap-abcdefgh',
    ApplicationId: 'app-67890',
    UserId: 'user-id', //Random UserId
    ClientSession: TCGSDK.getClientSessic
});
```

Appendix: Detailed description of project types



Project Type	Description	Main Strengths	Use case

Single-application	The project is associated with only one application, and concurrency packs bound to the project can be used only by that application. When you call the ApplyConcurrent API to request a concurrency quota, you need to pass in the `ProjectId` parameter.	You can enable the prelaunch feature to quickly load the application when a user connects to it.	A virtual exhibition application requires at least 100 concurrencies and uses the prelaunch feature to allow users to quickly access the application without waiting for it to load.
Multi-application	Concurrency packs that are bound to the project are shared by all the applications under the project, but the prelaunch feature cannot be enabled. When you call the ApplyConcurrent API to request a concurrency, you need to pass in the `ProjectId` and `ApplicationId` parameters.	Multiple applications can share concurrencies to reduce costs.	Multiple exhibition applications which are independent of each other need to share the concurrency pool, so that each application can get idle concurrencies from the pool as needed when there are user access requests.

Note:

To maintain a balance between user experience and cost optimization, you can create a single-application project and a multi-application project and use them together:

When there are user requests, you can first schedule the concurrencies in the single-application project. If prelaunch is enabled for the project, users can quickly use the application without waiting for it to load.

When available concurrencies in the single-application project become insufficient (ApplyConcurrent() returns a prompt indicating that **there are no idle concurrencies**), you can then schedule concurrencies in the multi-application project.

For more information, see Starting an Application and Queue Feature.

How to Implement Mobile Chinese Input

Last updated：2024-06-14 11:51:02

As a cloud 3D application (in .exe format) runs in the cloud Windows Server environment, if the default Windows input method of the cloud desktop is used, users on mobile devices may have a poor experience. Therefore, for use on a mobile device, we recommend that you follow the steps below:

1. Listen for the onInputStatusChange callback to determine whether the cloud input box is focused.

Note:

If the cloud application uses a custom input box, the `onInputStatusChange` callback may not take effect.

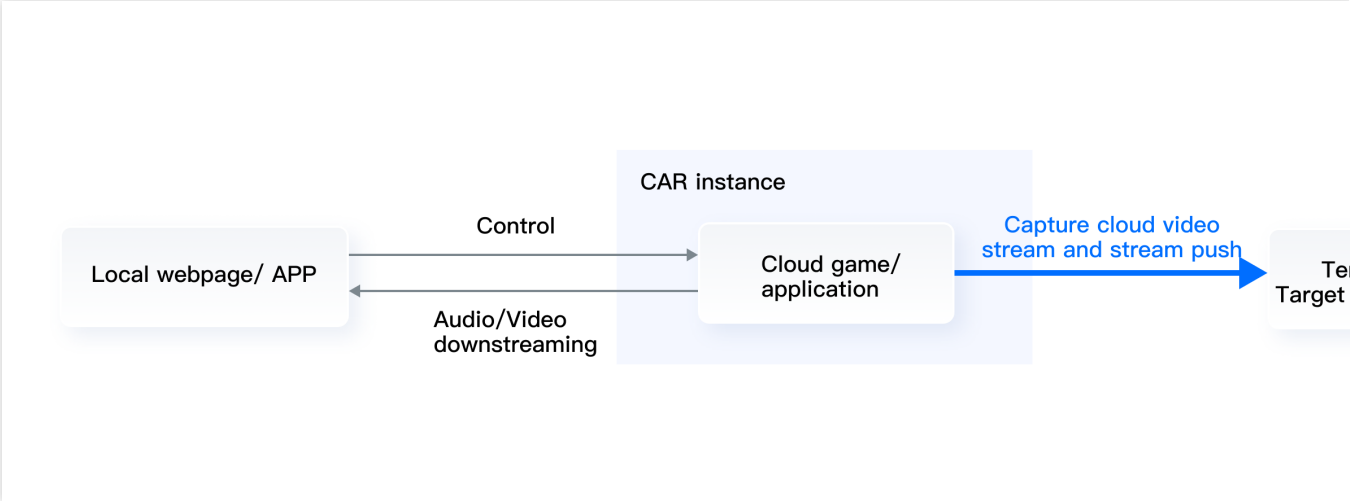
2. When the cloud input box is focused, an input box will be displayed on web. After the native input method on the user's mobile phone is called to enter text, the TCGSDK.sendText() API can be called to pass in the text characters to the cloud application.

3. For use in landscape mode, the user needs to be asked to **enable auto-rotate on their mobile phone** and use the input method in landscape mode, so as to avoid the issue where the rendered application image is displayed in landscape mode but the local input method is still in portrait mode.

How to Push Cloud Video Streams to CSS

Last updated：2024-01-26 12:00:38

When cloud applications run in scenarios such as virtual meetings, virtual exhibitions and interactive games, it is often necessary to push video streams to a live room at the same time so that more viewers in the live room can watch the rendered application without using additional CAR concurrencies.
Cloud-based stream pushing supports two methods of transmission: (1) Pushing streams to CSS by binding a CSS domain name; (2) Transmitting the target stream pushing address to transmit visuals to a specified address.



Procedure

1. Application Deployment

Perform operations such as uploading an application, creating a project, purchasing concurrencies, and testing the project on the CAR console to run your application. For more information, see User Guides.

2. Frontend and Backend Deployment

Deploy your frontend client and backend service as instructed in CAR PaaS Integration or Integration Demo.

3.Cloud-based Stream Pushing

Method 1: Pushing Streams to CSS

1. Before using the stream pushing to CSS function, ensure that CSS is activated.
2. On the CAR console, bind the Tencent CSS stream pushing domain name.

3. Initiate a cloud rendering concurrency and run the application. (Optional) If 24-hour uninterrupted live streaming is required, you can set the **RunMode** to **RunWithoutClient** during the session creation, thereby ensuring the continuous cloud application running and uninterrupted live streaming even in the absence of a client connection.
4. Invoke the push starting interface to push the images on the CAR concurrency to CSS in real time.
For step details, refer to Pushing Streams to CSS**.**
Note:
For the billing method of the Tencent CSS stream pushing , see the billing description.

Method 2: Pushing Streams to a Specified Address

Pushing streams to a specified address currently only supports the RTMP protocol. This function must be enabled on the console before it can be invoked through the interface.
1. On the function page of the cloud-based stream pushing on the console, select the tab indicating stream pushing to a specified address, and activate the service.
2. Initiate a cloud rendering concurrency and run the application. (Optional) If 24-hour uninterrupted live streaming is required, you can set the RunMode to RunWithoutClient during the session creation, thereby ensuring the continuous cloud application running and uninterrupted live streaming even in the absence of a client connection.
3. Invoke the stream pushing interface to push the streams of the cloud-based visuals to the specified address in real time.
For step details, refer to Pushing Streams to a Specified Address**.**
Note:
For the billing method of pushing streams to a specified address, see the billing description.

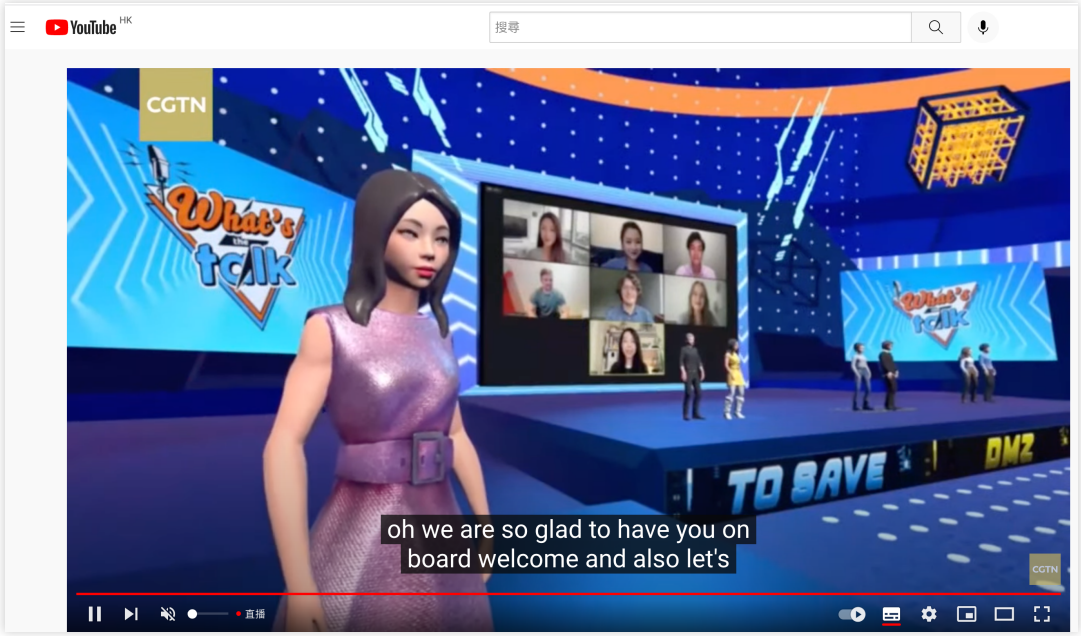
Customer Success Cases

Bullet Screen Interactive Game

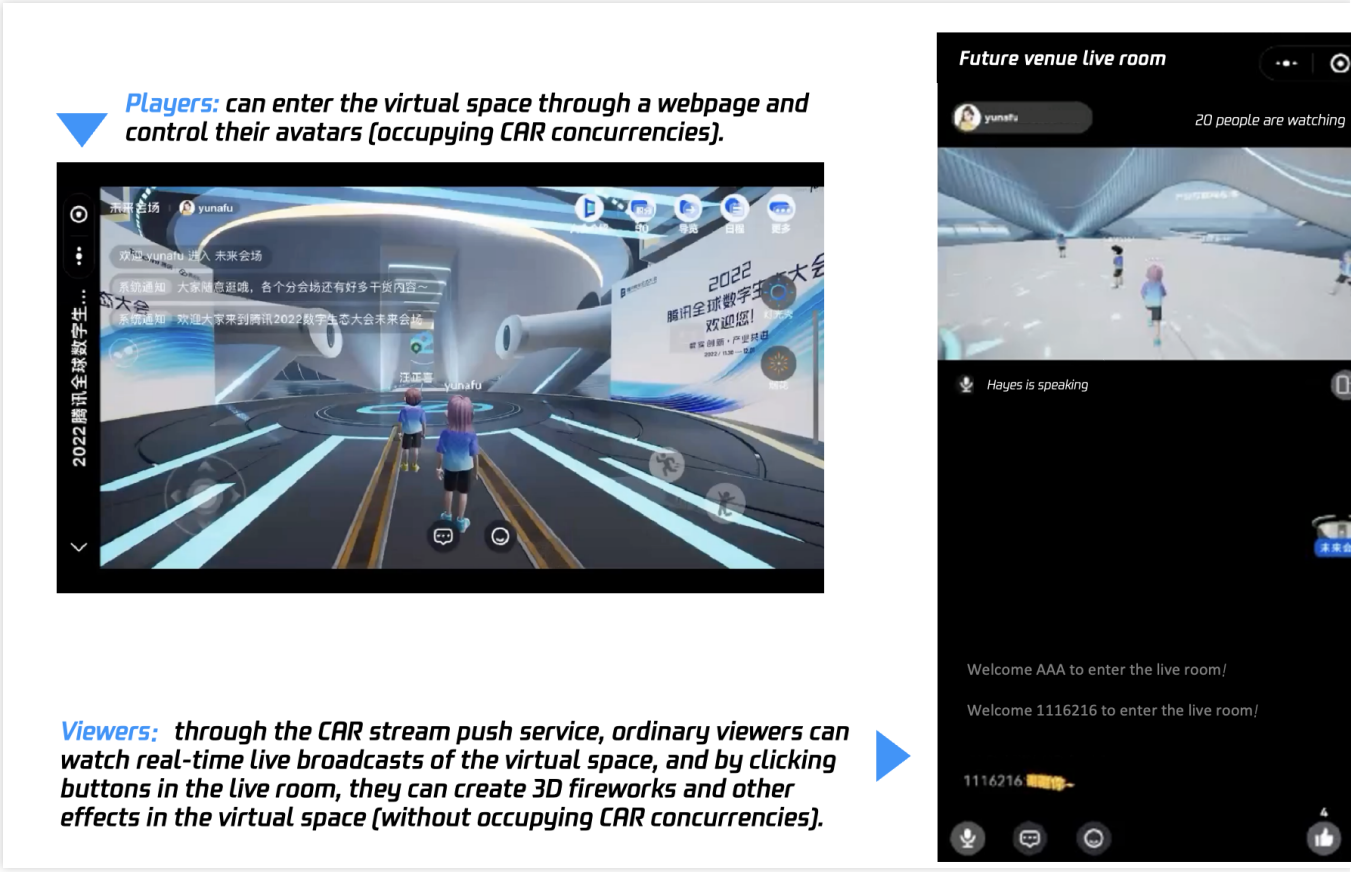
A live streaming platform employs cloud rendering to operate bullet screen games, and push streams of the visuals to the live broadcast room. In the conventional broadcasting mode, bullet screen games require local installation and operation, setting a high threshold for broadcasters. By using cloud-based rendering for broadcasting, there is no need to download or install the games. Broadcasters can simply initiate the live bullet screen games via mobile applications. This mode is widely applicable to short video, voice chat, social, and live streaming applications.

Virtual event live streaming

1. A customer in the radio and TV industry runs their 3D virtual event app via CAR. The streaming host can control the app and direct the event from their browser. The audience can enter the virtual event where the host is located by opening a link, and they can view the stream and interact in real time with no downloads or installations required. Meanwhile, the customer uses the CAR stream push service to push the cloud application stream to CSS and other online video-sharing platforms with low latency, enabling tens of thousands of other audience members who have not entered the virtual space to watch the entire event in the live room (without using additional concurrencies).



2. With the support of CAR, Tencent Cloud created a venue mini program featuring immersive convergence for the 2022 Tencent Global Digital Ecosystem Summit. Using the mini program, users could quickly enter the 3D venue, interact, and watch talks from experts from an aerial view in the live room (without using additional concurrencies). They could not only interact with other audience members through text and emojis but also control the 3D special effects in real-time by clicking the Light Show/Fireworks button. With the CAR stream push service, the venue also allowed audience members to send invitations to invite friends to join a live viewing room. Everything the audience members did, saw, and heard in the venue could be shared to their friends' devices.



On-screen commenting interaction in the live room

DouYu hosted a virtual tournament viewing, where the **virtual space was streamed in real-time to the live room. Audience members could enter the virtual space by clicking a link while watching the live stream. Those who had not entered the virtual space could also create virtual avatars through on-screen comments and join the dance floor to compete for camera attention (without using additional concurrencies).** For more information, see How to Implement Live Room Interaction with On-Screen Comments.

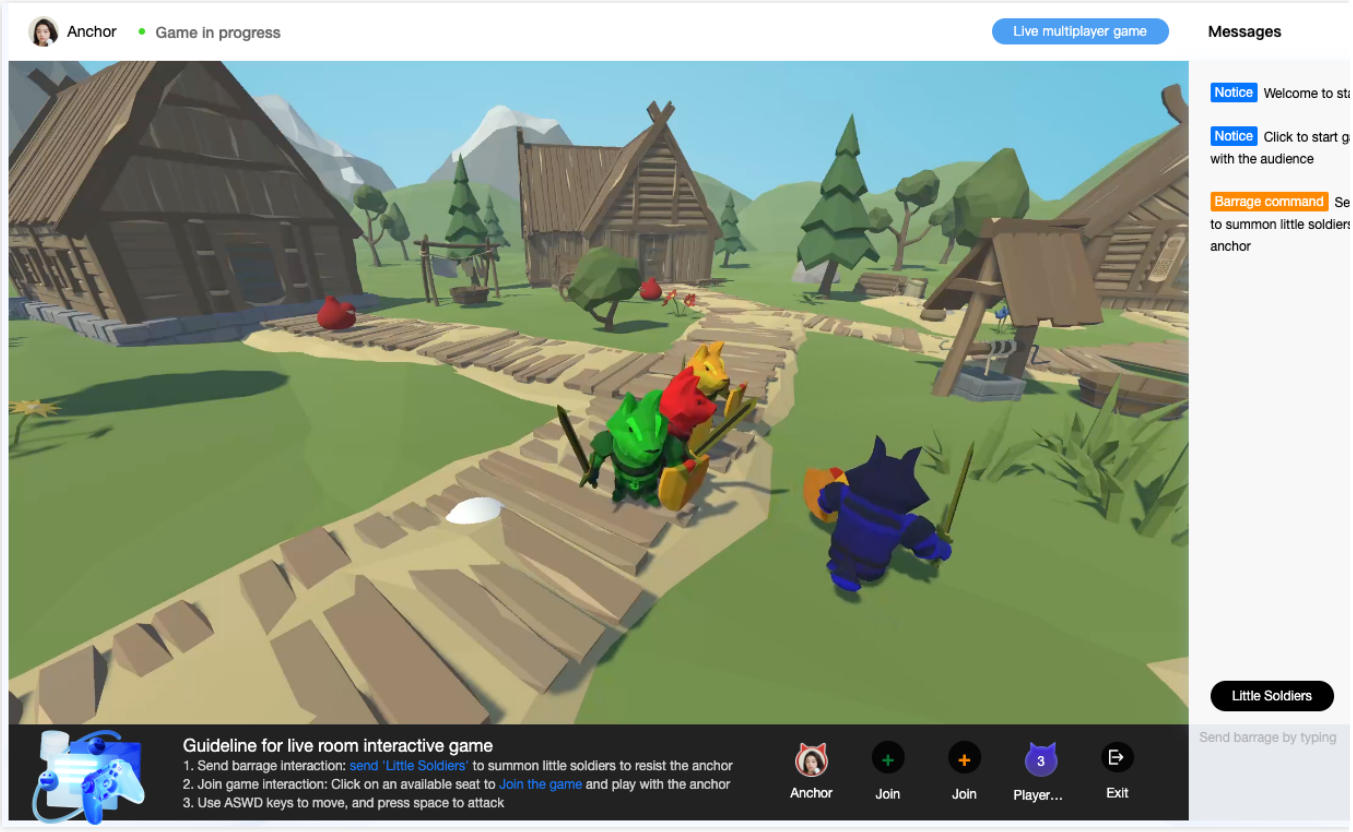
How to Implement Multi-Person Interaction

Last updated：2024-03-18 14:49:15

Customer Scenarios and Practices

This feature is suitable for scenarios such as multi-person interaction and live streaming, which can upload single-player/multiplayer games to the cloud. The streaming host is able to easily start a game on any device. By integrating interactive features in the live room, audience members can take a role in a multiplayer game by sending gifts or being invited. CAR pushes game video streams to the cloud, so that the live stream of the game can be watched in the live room in real time. This feature allows you to implement online game competition, online team battling, and online game sparring. It can also help increase the popularity and gift revenue of the live room.

You can experience live multiplayer interaction in Technology Experience Center - Live multiplayer game.



Scenario 1. Multi-person interactive live streaming

Use case: A live streaming service provider needs to implement in-game interaction between the streaming host and the audience during live streaming.

Implementation logic:
When starting a stream, the streaming host connects to the cloud game and creates a room, and other players can send a request to the host or use the invitation link provided by the host to enter the room for multiplayer gaming. Other audience members can connect to the live stream and watch the in-game interaction between the host and other players.

Scenario 2. Multiplayer arena game

Use case: A cloud game company needs to implement a cloud multiplayer arena game.
Implementation logic:
The admin creates a cloud game room.
Other players can queue up to play. The winner will continue playing the game, and the loser will switch to a viewer role or exit the game.

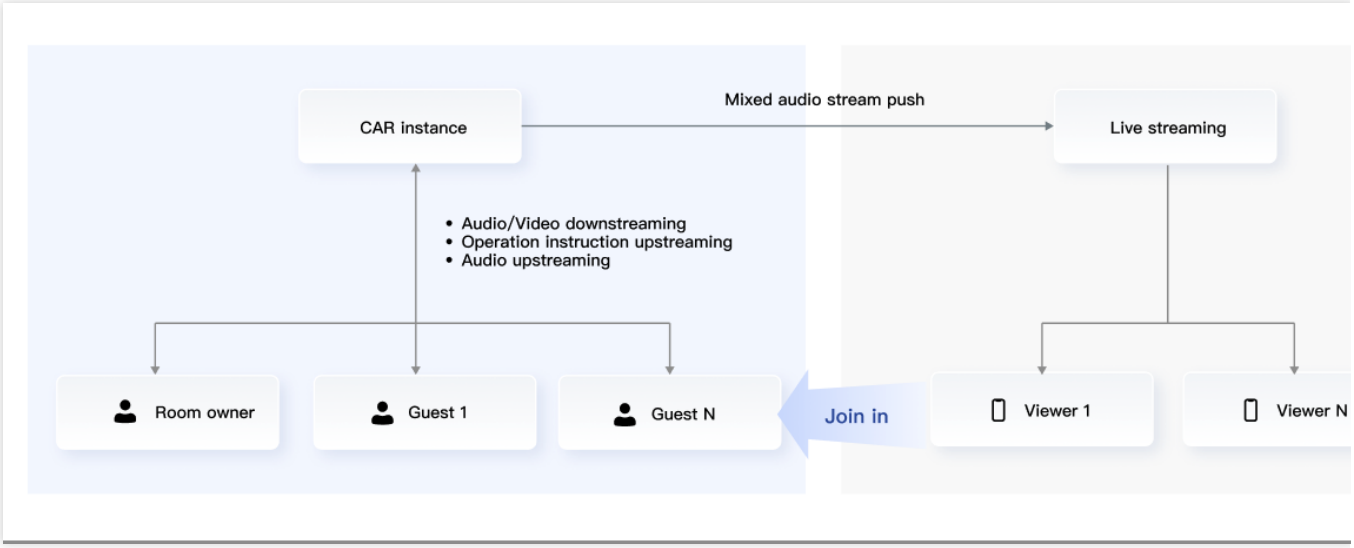
Quick Connection

First, you need to develop an application that can run stably on Windows and implement required features in the application.
For example, the application should include basic game scenarios, support at least 1-4 players to join a room, and be controlled using a controller.

After you have an application, you can perform the following steps.
1. Activate CAR.
2. Deploy an application as instructed in User Guides.
3. Deploy your frontend and backend service and successfully start the application in the cloud. For details, see Integration Demo.
After completing the above steps, you can refer to the following instructions to integrate multiplayer interaction. To push cloud video streams to a live room, see here.

Multiplayer Interaction Concepts

The multiplayer interaction can be abstractly understood as a CAR room and a live stream (optional). After a room is created by a CAR player (room owner), other players (interactive audience) can enter the room through the room owner's `UserId` , and all the users in the room can see the same cloud image through connection via CAR.



User types

Room owner: The player that creates a room

The room owner determines whether others have control over click, keyboard/mouse, and controller operations.

The room owner manages the mic status of everyone in the room.

Interactive audience: Users that enter the room and can have the role of a `Player` or a `Viewer` .

An interactive audience member needs to request control from the room owner.

When granted game control by the room owner, the member has the `Player` role.

When not granted control by the room owner, the member has the `Viewer` role and has no game control.

Interactive audience members can turn the local mic on or off.

Streaming audience: Users that can watch the live stream but are not connected to a concurrency instance

A streaming audience member can only watch the live streamed image and has no other permissions.

CAR role description

Host: The original creator of a room

The host can switch the role of interactive audience members. When the host exits, the room is immediately terminated, and all interactive audience members automatically exit.

Player: An interactive audience member who has game control

The default maximum number of players is 7, which should actually be the same as the maximum number of players supported by the game.

Viewer: An interactive audience member who does not have game control and can only watch the game

The default maximum number of viewers is 7.

Note:

The room owner can switch a member's role to player to let the member control the application or to viewer to let them watch only.
An interactive audience member can be a player or a viewer. The room owner can control which role each member has.

Process

Service activation

Activate this value-added feature on the Multiplayer page in the console.

The room owner creates a room

1. The room owner's client sends a request to the business backend to start CAR, and the business backend calls the ApplyConcurrent() API to apply to reserve a CAR concurrency.
2. The business backend calls the CreateSession() API to create a session, where `HostUserId` must be the same as `UserId` and `Role` can be `Player` or `Viewer` .

Note:
The process of creating a room in multi-person interaction mode is the same as the process of creating a session by calling the TencentCloud API, except that `HostUserId` needs to be entered in the latter case.

An interactive audience member enters the room

1. The member's client sends a request to the business backend to start CAR.
2. The business backend calls the CreateSession() API to create a session, where `HostUserId` must be the same as the room owner's `UserId` and `Role` can be `Player` or `Viewer` .

Note:
Only the room owner needs to call `ApplyConcurrent()` , and members do not.

Live stream push

Refer to the best practice guide How to Push Cloud Rendering Screens to The Live Room.

Switch the role

1. The interactive audience member's client calls the TCGSDK.submitSeatChange() API to request a switch of the role to player or viewer.

Note:
A switch from viewer to player requires the information of the seat, which must be vacant.
A switch from viewer to viewer can be made directly without a request.
A switch from player to viewer can be made directly without a request.

A switch from player to player requires the information of the seat, and a seat switch between players will result in a role switch in the game.
In other words, a switch to player requires the seat information, while a switch to viewer does not.
2. The room owner's client calls back the onMultiPlayerChange() API to get the member's request to switch the role and calls the TCGSDK.seatChange() API to switch the member's role.
3. The member's client calls back the onMultiPlayerChange() API to get the response to the request for a role switch.

Switch the mic status

1. The room owner's client calls the TCGSDK.changeMicStatus() API to switch the status of the local mic or another user's mic.
2. An interactive audience member's client calls the TCGSDK.changeMicStatus() API to switch the local mic status.

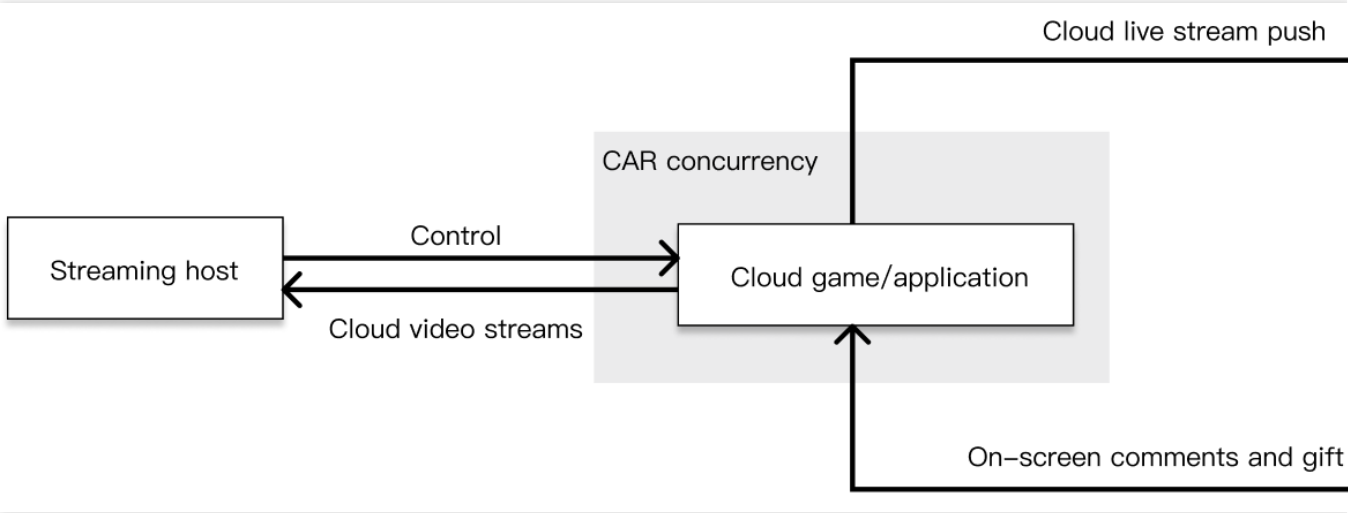
How to Implement Live Room Interaction with On-Screen Comments

Last updated：2024-01-26 12:00:38

Traditionally, streaming hosts can only run games in their local system, which has high requirements for hardware configurations such as the computing power of the graphics card, meaning that not everyone can become a game streaming host. Additionally, the local device needs to be continuously powered on and running, resulting in high maintenance costs.
By hosting the application in the cloud, CAR enables games and live streaming applications to run in the cloud while pushing a stream of the game video to specified platforms in real time. This provides smooth and clear gameplay and video streaming without needing to install game software or configur the runtime environment locally.

Concepts

Currently, on-screen commenting and interactive gameplay features in many live rooms are developed based on the game engine. For example:
Battle game: Users can select a faction and join the battle by sending comments. Sending gifts can trigger more powerful equipment.
Cloud dancing: Users can join the dance floor with a virtual avatar by sending a specific comment. They can also send gifts to switch songs or change the avatar's appearance.
Survival/Cultivation game: Users can send comments to generate characters, survive, and level up in the game. Gift rewards can accelerate growth.
The above scenarios are all based on the Windows system. The streaming host manually installs and runs an application locally, and then the app client captures the video stream and pushes it to the live room.
CAR helps to quickly start a stream for games developed with engines such as UE and Unity, with no local downloads or installations required.
The application runs in the cloud and automatically retrieves the specified on-screen comments and gift information to trigger various in-game interactions in real time. The game video is captured and pushed to the specified live streaming platform from the cloud.



Strengths

On-cloud running: Applications and games can run in the cloud. Games developed with engines such as UE and Unity can be quickly deployed and loaded in seconds, regardless of the size of the game. Game content does not need to be repeatedly redeveloped for different architectures, which saves costs and time.
Cloud stream push: The cloud video streams are directly pushed to the specified platform to save local bandwidth and device performance, making video images smoother and clearer.
Offline stream push: Applications/Games can run in the clientless mode on the cloud 24/7 without needing to depend on a local device.
Multi-person control: Applications/Games can be controlled by multiple users on the cloud, allowing for remote multi-player gaming.

Directions

First, you need to develop an application that can run stably on Windows and implement required features in the application.
For example:
Basic gaming scenarios, including joining methods for audience members and logic for wins and losses during gameplay.
The capability to retrieve on-screen comments and gift information from the live room. This can be developed via open APIs provided by each live streaming platform.
After you have an application, you can perform the following steps.
1. Activate CAR.
2. Deploy an application as instructed in User Guides.

3. Deploy your frontend and backend service and successfully start the application in the cloud. For details, see Integration Demo.

Performance optimization:

In most cases, games that use on-screen commenting use CAR to start streams through mobile phones, which have varying screen resolutions. To better achieve a full-screen display without black bars, we recommend you use Adaptive Resolution.

When a game starts, it needs to be associated with the live room where the streaming host is located. In most cases, the host needs to enter a room number when starting a stream locally. This step may create a barrier to the host. We recommend game developers support obtaining the room number through startup parameters or listening ports. To use a listening port, you need to use the Data Channelfeature. To use a startup parameter, you need to call the CreateSession API and set the `ApplicationParameters` value.

If the technical conditions permit, we recommend you use a data channel. This is because startup parameters must be passed before the game starts. In this case, the game cannot be prefetched, resulting in a slower startup speed compared to using a data channel.

4. Start the stream push. Two stream push methods are supported:

Push a cloud video stream to CSS by binding a push domain. For more information, see Pushing Streams to CSS.

Push a cloud video stream to a third-party address according to the destination URL you enter. For more information, see Pushing Streams to Third-Party Addresses.

If you do not need to view the application in the frontend window on your local computer, you can use the clientless mode. Set `RunMode` to `RunWithoutClient` when calling CreateSession, which allows the application to keep running on the cloud even when there are no client connections.

Note:

If you need help with the production of games or interactive content, please submit a ticket for a project assessment.

Selection of Concurrency Specifications

To ensure the smooth operation of bullet screen games, it is essential to select appropriate CAR concurrency specifications. In the scenarios of bullet screen games, the application's hardware requirements do not directly depend on the number of viewers in the live room. Instead, they are associated with the hardware configuration requirements of the game itself, the complexity of the images, the number of bullet screens, and the number of units on the same screen. Given the varying popularity of platforms and anchors, the configuration requirements often differ significantly.

Therefore, when choosing CAR concurrency specifications, it is recommended to conduct stress tests/gray tests for each game and select the appropriate concurrency specifications in conjunction with the characteristics of the live streaming platforms.

Related Content

For more immersive live room interactions triggered by on-screen comments, see How to Implement Multi-Person Interaction.