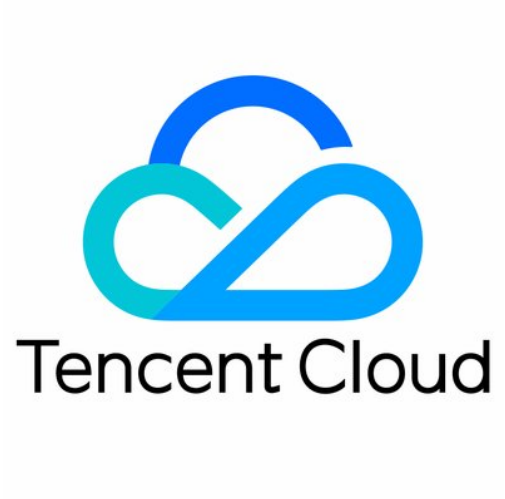
Cloud Application Rendering

Getting Started

Product Documentation



Contents

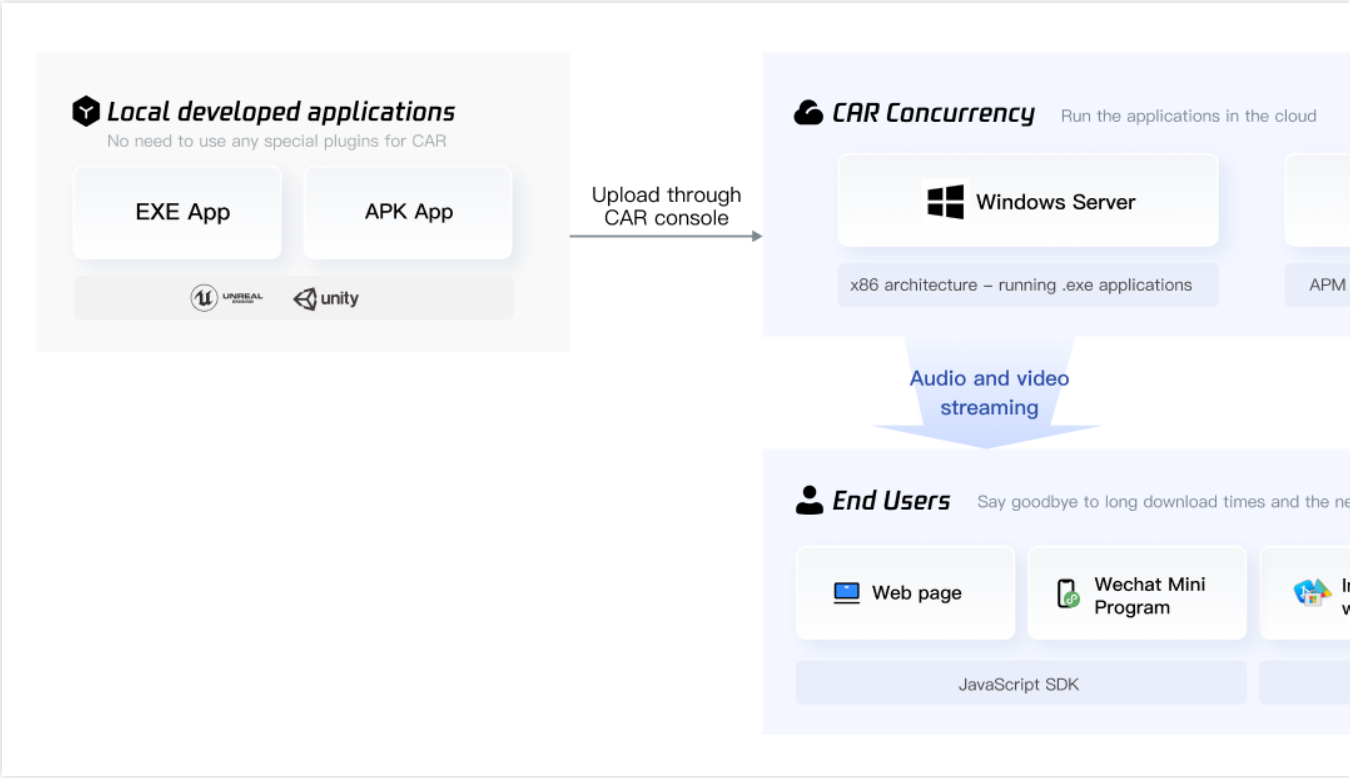
Getting Started
- Basic Technical Concepts
- User Guides
- Technical Integration
  - Starting an Application
  - Closing the Application
  - Queue Feature
  - Data Channel
  - Heartbeat Connection
  - Application Reconnection
  - Adaptive Resolution
  - Enabling Mic
  - Enabling Camera
  - Mouse/Keyboard/Touch Interactive Processing
- Integration Demo

Getting Started
Basic Technical Concepts

Last updated：2024-01-26 11:54:09

What is Cloud Application Rendering (CAR)

Cloud Application Rendering is a PaaS product with features like
Application self-upload and update,
Concurrency (Virtual Machine) self-management,
Automatic concurrency scheduling based on global end-user location for optimal proximity,
Ultra-low latency audio and video streaming,
JavaScript, iOS, and Android SDK
and so on.
It runs your developed EXE/APK applications on cloud concurrencies, allowing end-users to experience them through web pages without downloading. The provided terminal SDKs transmit audio and video results to end users' devices and send real-time user operations back to the concurrencies. To integrate CAR, you only need to ensure that the application can run normally in a Windows Server 2019 / 2022 or Android environment, without using any special plugins, making cloudification easy.

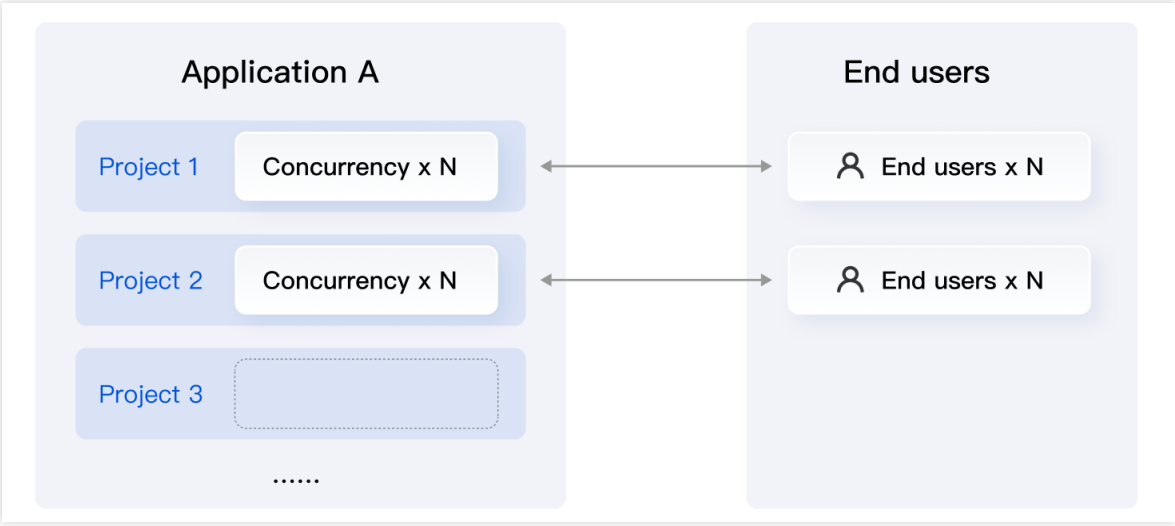


CAR has three basic concepts: application, project, and concurrency.

Application: Your developed application. Upload it to the console to run it in the cloud concurrency.

Project: Group concurrencies through projects and associate them with applications. Create different projects for an application to meet various testing and deployment scenarios without repeatedly uploading the application.

Concurrency: The only billing item for CAR. One concurrency equals one virtualized cloud instance, including virtual computing resources such as CPU, bandwidth, disk, and GPU, used to run your application in the cloud.



Note：

For more information on concurrency configuration and specifications, please refer to Billing.
At present, **only PC desktop applications (.exe) can be uploaded through the console, and only x86 concurrency (Windows Server 2019/2022 system) is available**. If you want to deploy a .apk application and use ARM concurrency (Android system), please contact your Tencent Cloud sales representative.

TencentCloud API

TencentCloud API: TencentCloud API 3.0 is the basis of Tencent Cloud open ecosystem and boasts strengths such as ease of automation and remote call, high compatibility, and low system requirements. It enables you to quickly manipulate Tencent Cloud products with only a small amount of code, and improves the efficiency for frequently called features. You can also combine different TencentCloud APIs to implement more advanced features. For more information on TencentCloud API 3.0, see Introduction.
Access key: An access key is a TencentCloud API key, which is a security credential used for authentication when you access TencentCloud APIs. It consists of a `SecretId` and `SecretKey`. If you don't have an API key yet, you need to create one in **Manage API Key**; otherwise, you cannot call TencentCloud APIs. You can get the access key on the Manage API Key page in the console.

Running Environment

Cloud software environment of a CAR concurrency instance

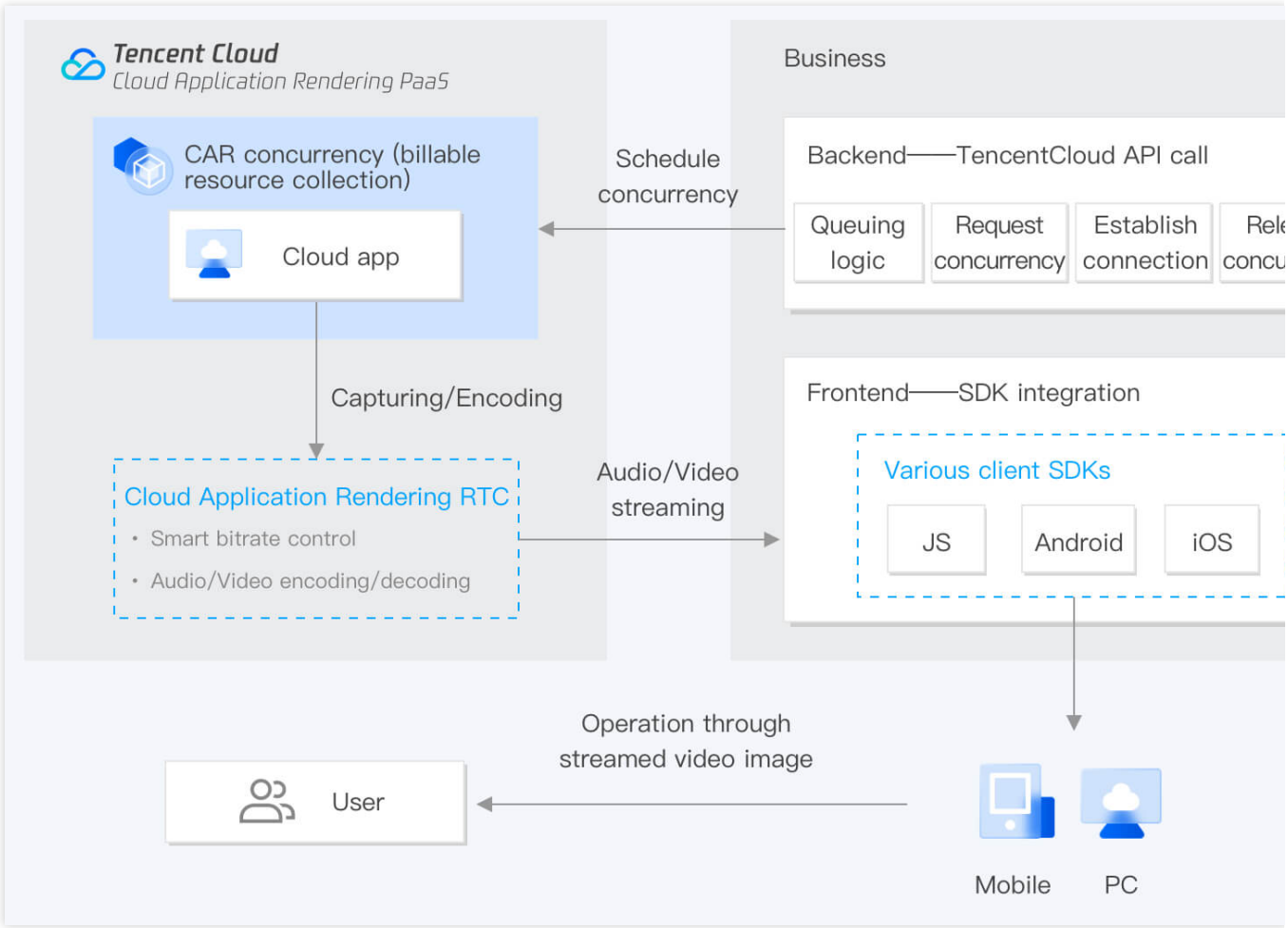
Deployment prerequisites: The uploaded application will run in the purchased concurrency instance. The installation package of the application must support portable deployment and installation; that is, application operations don't rely on modifications of the registry or other system configurations.
System limits: PowerShell and CMD services on Windows cannot be called.
Port limits: Port listening is not supported because the public IP provided by the service is not fixed. Subnet broadcasting is not supported. If you have such needs, we recommend that you use the public network for communication.
Network conditions: The service does not prevent external access from your application.

CAR service environment

Network conditions: The network of CAR relies on UDP data reception and sending, so you need to open UDP port 60000-60100 to all source IPs on the client. If there are no special security concerns, we recommend you open all UDP ports.
If Wi-Fi or your company intranet cannot access CAR service, you can try testing it on a 4G network. If CAR service can be accessed on a 4G network, it indicates that there may be network restrictions on the Wi-Fi or company intranet.

Service Integration Capabilities

CAR provides the ability to quickly launch applications without development. You can configure a standardized front-end page through the console and launch it for use. As a frontend/backend integrated PaaS product, CAR also provides backend APIs and various client SDKs. You can use our integration demo to build your own business backend services and business client programs to meet your unique needs.



Business backend:

You need to set up your own backend service and connect to the backend TencentCloud APIs provided by CAR to perform operations such as requesting a concurrency and creating a session.

We provide a backend demo for reference to help you set up your own backend. For detailed directions on how to deploy the demo, see Getting Started. For the detailed sequence diagram and how to integrate features such as data channel, adaptive resolution, and enabling the mic, see Technical Integration.

Deploying your own backend is a necessary step for using CAR services:

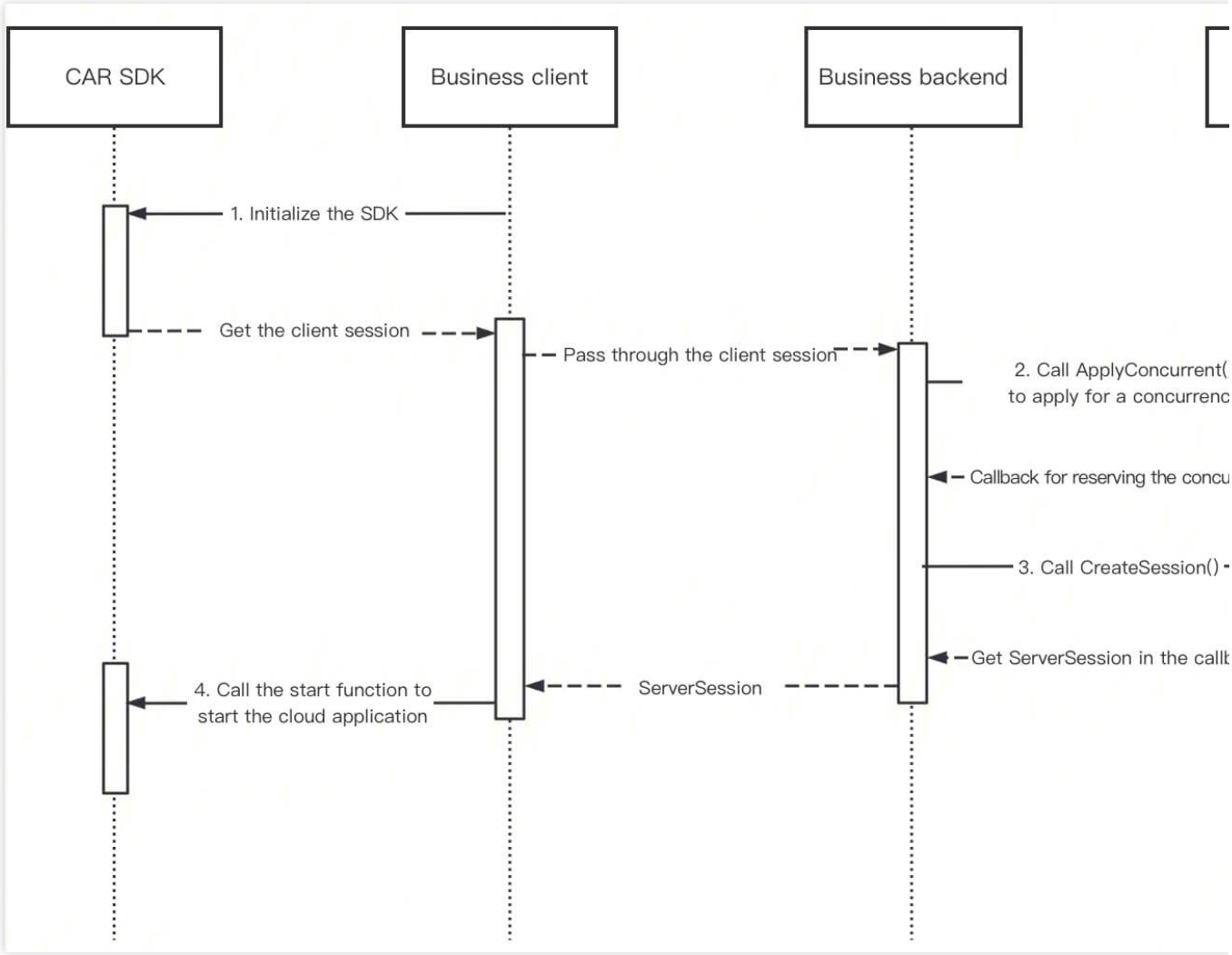
For the security of your assets and services, the `SecretId` and `SecretKey` of your Tencent Cloud account (which can be obtained on the Manage API Key page in the console) that are needed in order to access the CAR

TencentCloud API service must be processed on your backend service. In addition, management of user sessions and other features such as user queue must be implemented on your backend service.

Business client:

CAR provides SDKs for JavaScript, Android, and iOS. You need to set up your own client program and connect to these SDKs so that Android, iOS, and web end users can access your cloud-rendered services.

We provide the demo for JavaScript, demo for Android and demo for iOS for reference to help you set up your own client service. For detailed directions on how to deploy the demos, see Getting Started. The SDK provides many APIs, and you can customize your development based on the demos and SDK API documentation.

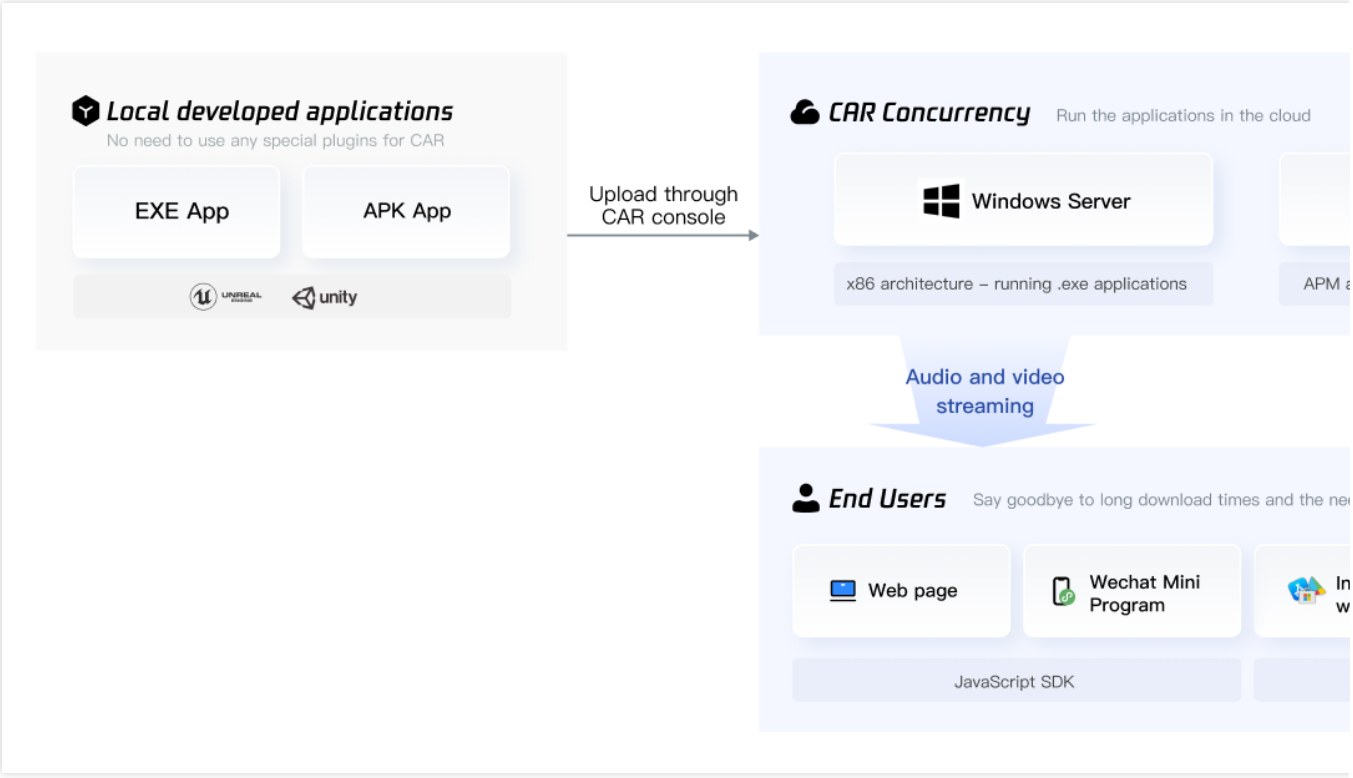


User Guides

Last updated：2024-01-26 11:54:09

What is Cloud Application Rendering (CAR)

Cloud Application Rendering is a PaaS product with features like
Application self-upload and update,
Concurrency (Virtual Machine) self-management,
Automatic concurrency scheduling based on global end-user location for optimal proximity,
Ultra-low latency audio and video streaming,
JavaScript, iOS, and Android SDK
and so on.
It runs your developed EXE/APK applications on cloud concurrencies, allowing end-users to experience them through web pages without downloading. The provided terminal SDKs transmit audio and video results to end users' devices and send real-time user operations back to the concurrencies. To integrate CAR, you only need to ensure that the application can run normally in a Windows Server 2019 / 2022 or Android environment, without using any special plugins, making cloudification easy.



Before reading this document, make sure you understand the basic technical concepts of CAR.

Sign up for a Tencent Cloud account and activate CAR

1. If you don't have a Tencent Cloud account yet, sign up for one first.
2. After applying to activate CAR, you can start using the CAR console.

CAR Integration

Before integration, we hope you can first understand the three basic concepts of CAR:

Application: Your developed application. Upload it to the console to run it in the cloud concurrency.

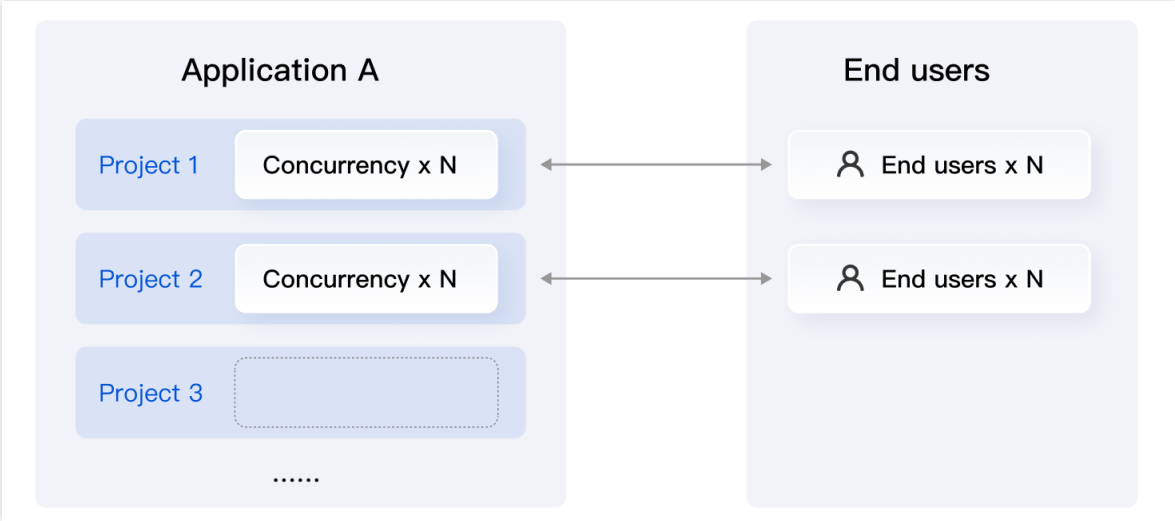
Project: Group concurrencies through projects and associate them with applications. Create different projects for an application to meet various testing and deployment scenarios without repeatedly uploading the application.

Concurrency: The only billing item for CAR. One concurrency equals one virtualized cloud instance, including virtual computing resources such as CPU, bandwidth, disk, and GPU, used to run your application in the cloud.

Note:

For more information on concurrency configuration and specifications, please refer to Billing.

At present, **only PC desktop applications (.exe) can be uploaded through the console, and only x86 concurrency (Windows Server 2019/2022 system) is available**. If you want to deploy a .apk application and use ARM concurrency (Android system), please contact your Tencent Cloud sales representative.

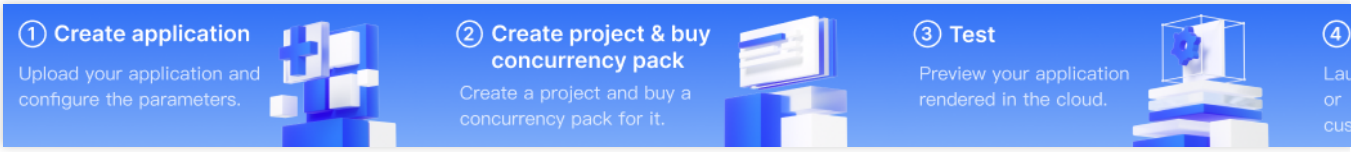


In the console, it is recommended that you follow the sequence of **(1) Creating an application, (2) Creating a project & purchasing concurrency packs** to obtain a project associated with a specific application and allocated with available concurrency. Next, you can carry out a **(3) Performance test** on this project, experience the basic

effect of your cloud application, and choose the most suitable concurrency scale. After completing the performance test, you can **(4) Launch the project**. We support:

Quick Launch (no development required): Quickly generate a standardized front-end page through simple configurations in the console.

By using the provided backend and client demo for integration, you can build your own backend services and client programs to meet your unique business requirements.



Step 1. Create application

1. Upload an application:

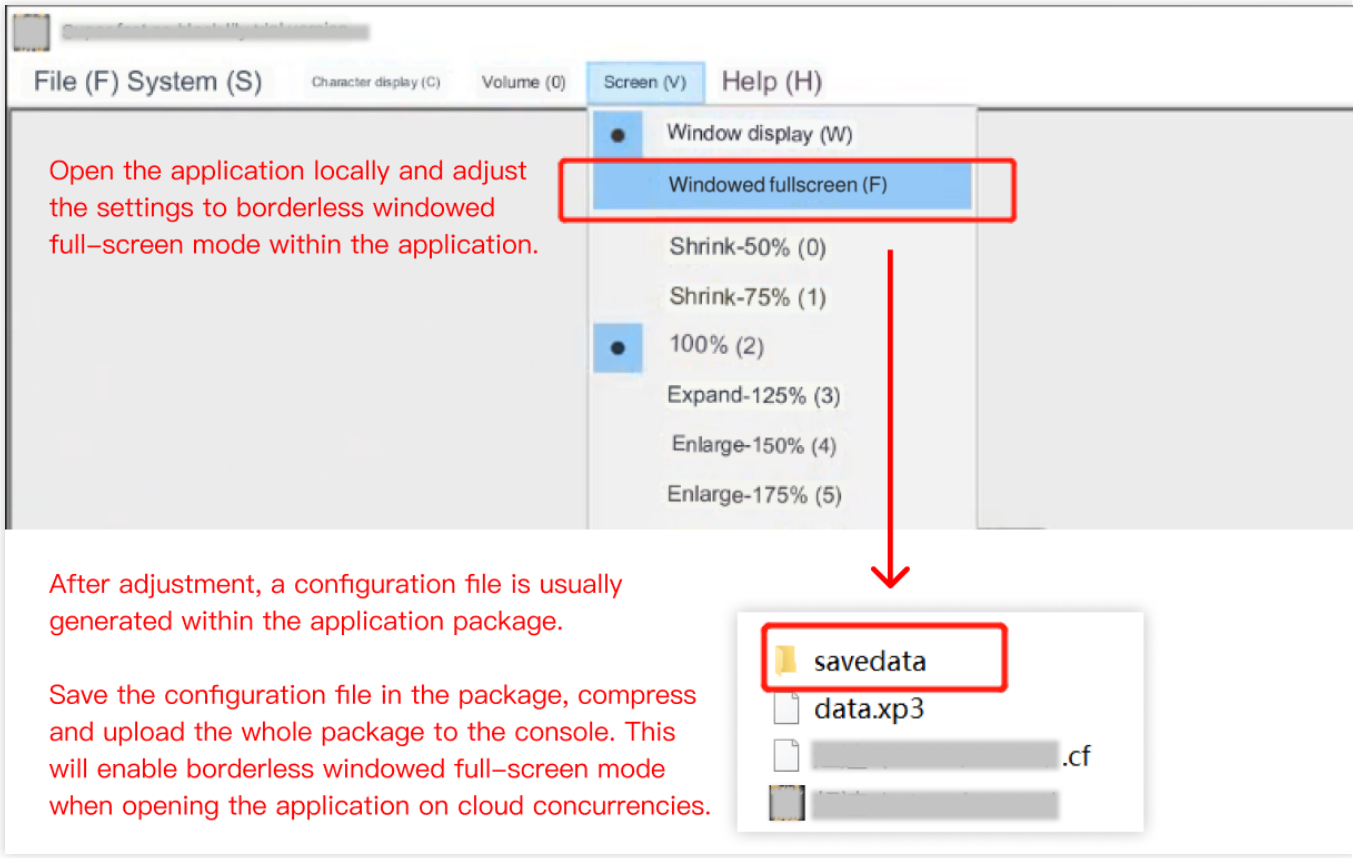
1.1

Before

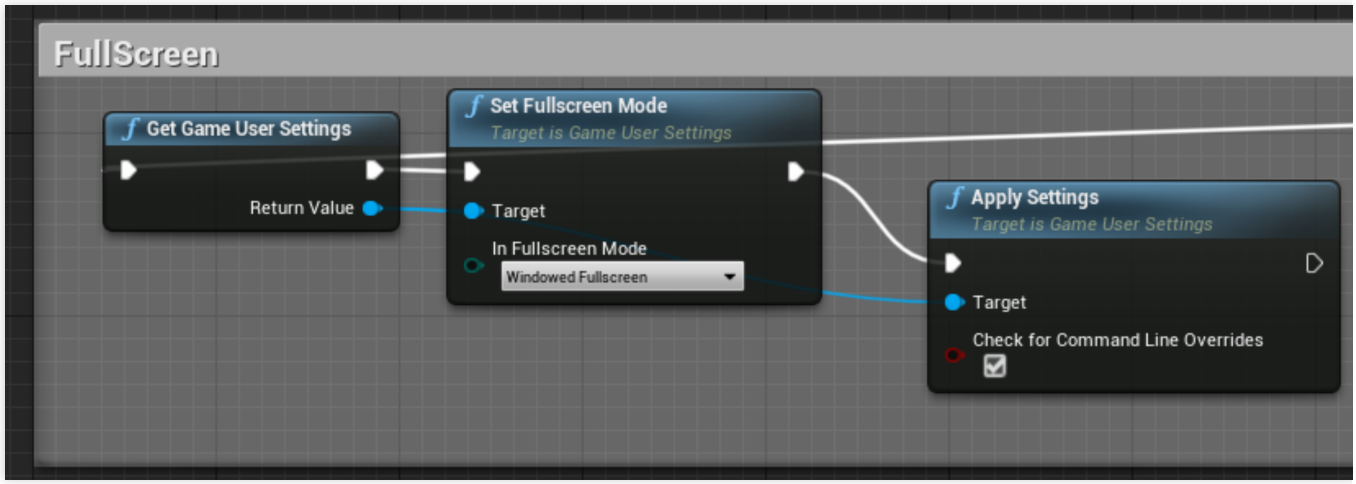
you upload an application to CAR, **we recommend that you first set the application to a borderless windowed mode that can adapt the desktop resolution** so that you can later use the adaptive resolution feature. This feature can make the application display full-screen, adapt to the resolution of end-user devices, and have no black borders on the screen.

This operation needs to be performed at the application layer, not in the console.

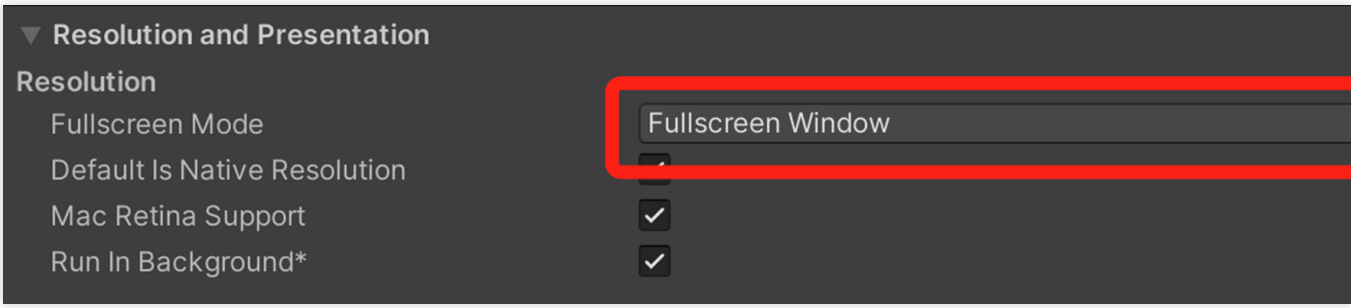
For some applications, you can adjust the display mode in the settings and upload the generated configuration file along with the package to the console.



You can also adjust the relevant display configurations during application development. For Unreal Engine, You can add a blueprint as shown below to `BeginPlay` in `Map` . `In Fullscreen Mode` should be set to `Windowed Fullscreen` .



For the Unity application, refer to the following image for instructions on how to configure the fullscreen mode.



Alternatively, you can add the application startup parameter `-screen-fullscreen 1` . For Unity application parameters, refer to the Unity documentation.

`-screen-fullscreen`	Override the default full-screen state. This n

Note:

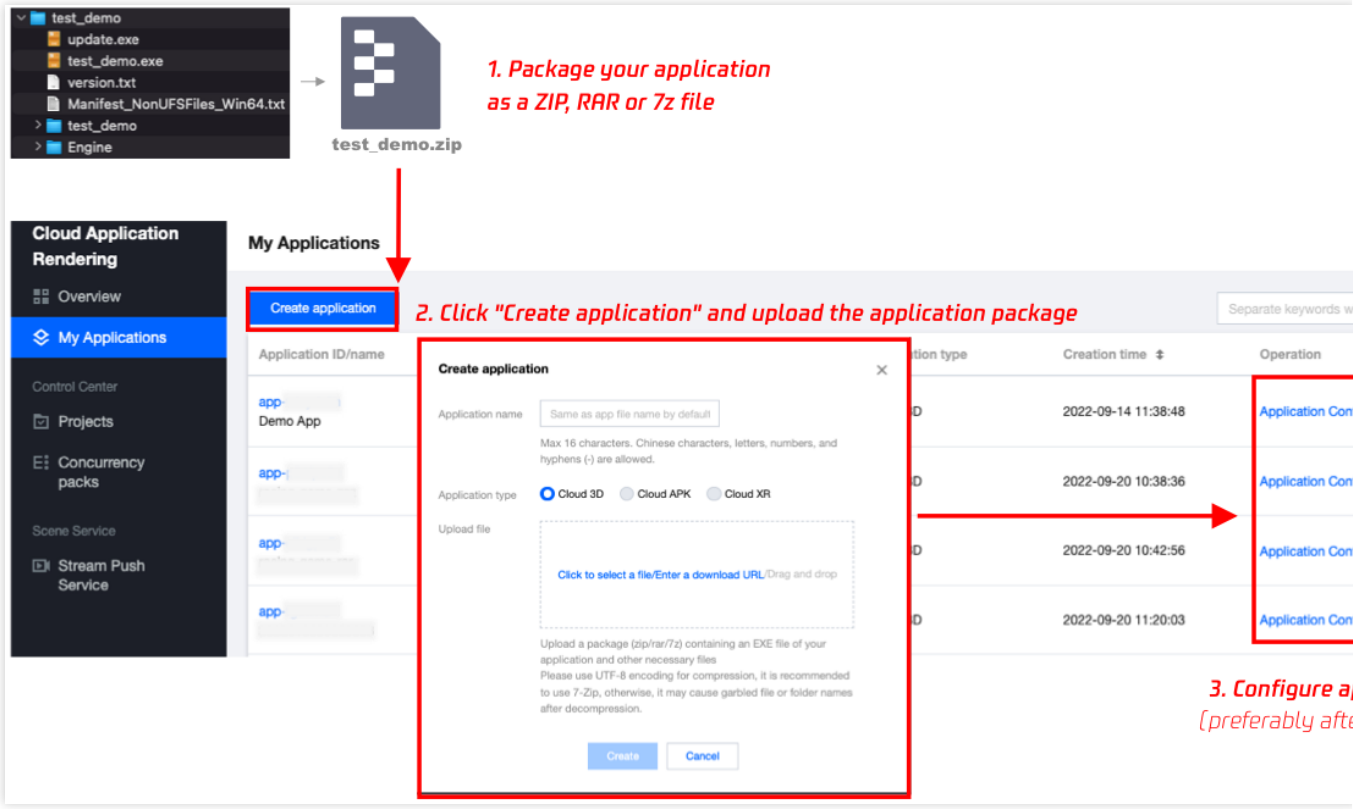
An application is not in the borderless windowed mode when displayed in fullscreen mode.

The Adaptive Resolution feature is achieved by modifying the "default resolution of cloud desktop". However, when the application is in full-screen mode, the desktop resolution is controlled by the application. In this case, a crash may occur when the desktop resolution is modified.

How do I differentiate between borderless windowed mode and fullscreen mode?

You can press **Alt+Tab**. Borderless windowed applications do not cause the monitor to flicker when the window is switched, while fullscreen applications do cause the monitor to flicker.

1.2 Package the application as a ZIP, RAR, or 7z file. Go to the **My Applications** page in the CAR console, upload the application, and wait for the application to be created.



Please use UTF-8 encoding for compression (recommended to use 7-Zip) to prevent garbled file or folder names after decompression.

Cloud rendering supports three types of applications: Cloud 3D, Cloud XR, and Cloud APK.

Cloud 3D: Traditional 3D (non-AR/VR class) applications, operated generally with a keyboard and mouse, such as virtual event applications and PC games. The application running on the cloud needs to be in the EXE format and can be adapted to a variety of platforms such as web pages, mini-programs, Android/iOS applications, etc.

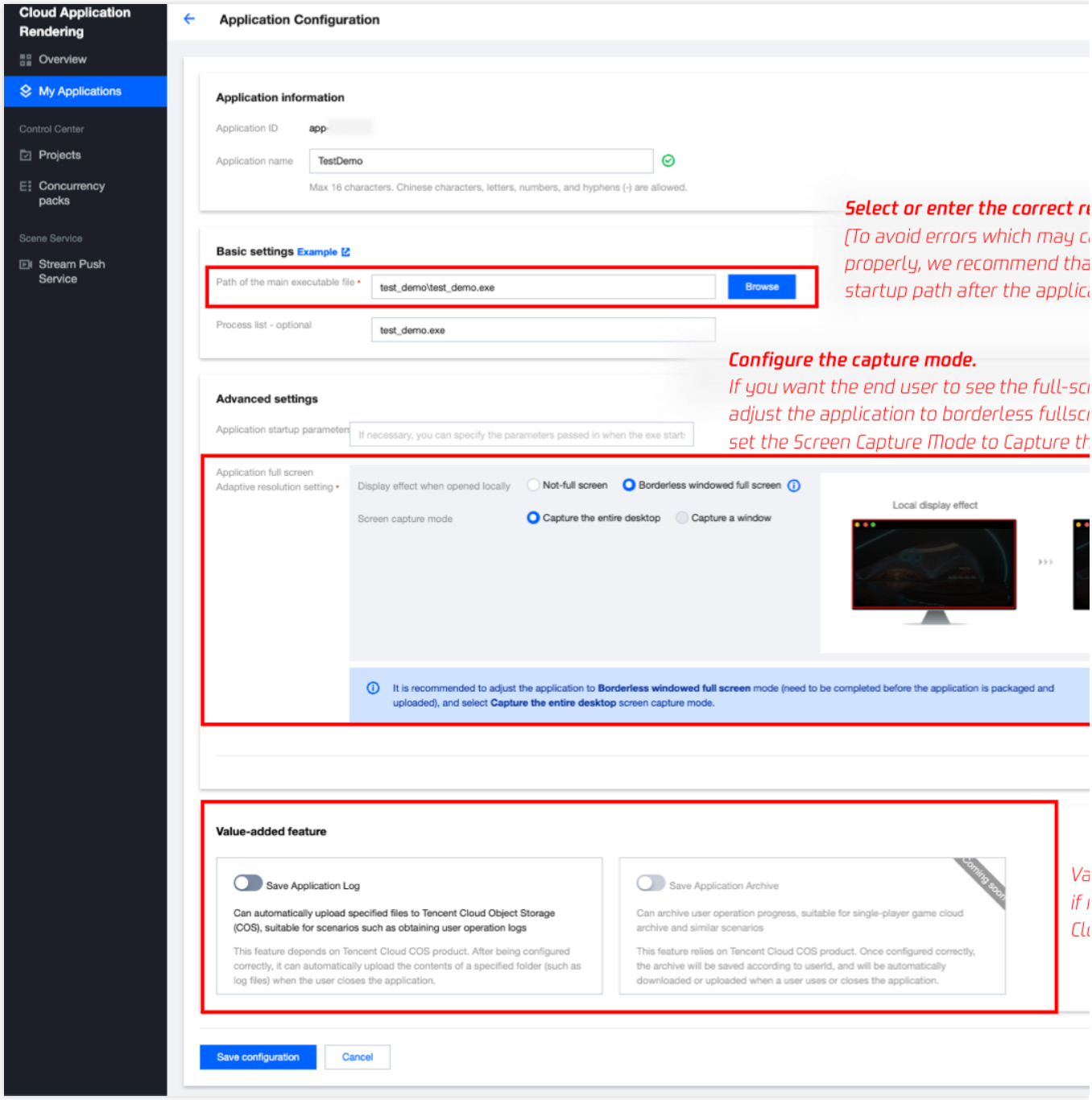
Cloud XR (beta): VR/AR/MR applications, generally operated with dual-hand controllers. The XR application running on the cloud needs to be in the EXE format, and can be adapted to Pico/Oculus, mobile phones/tablets, PCs, holographic terminals, and other platforms.

Cloud APK (beta): Applications in the APK format.

Note:

Cloud APK and Cloud XR are currently in beta, and self-upload is not currently supported. If you want to use, please contact your Tencent Cloud sales rep.

2. Configure the application

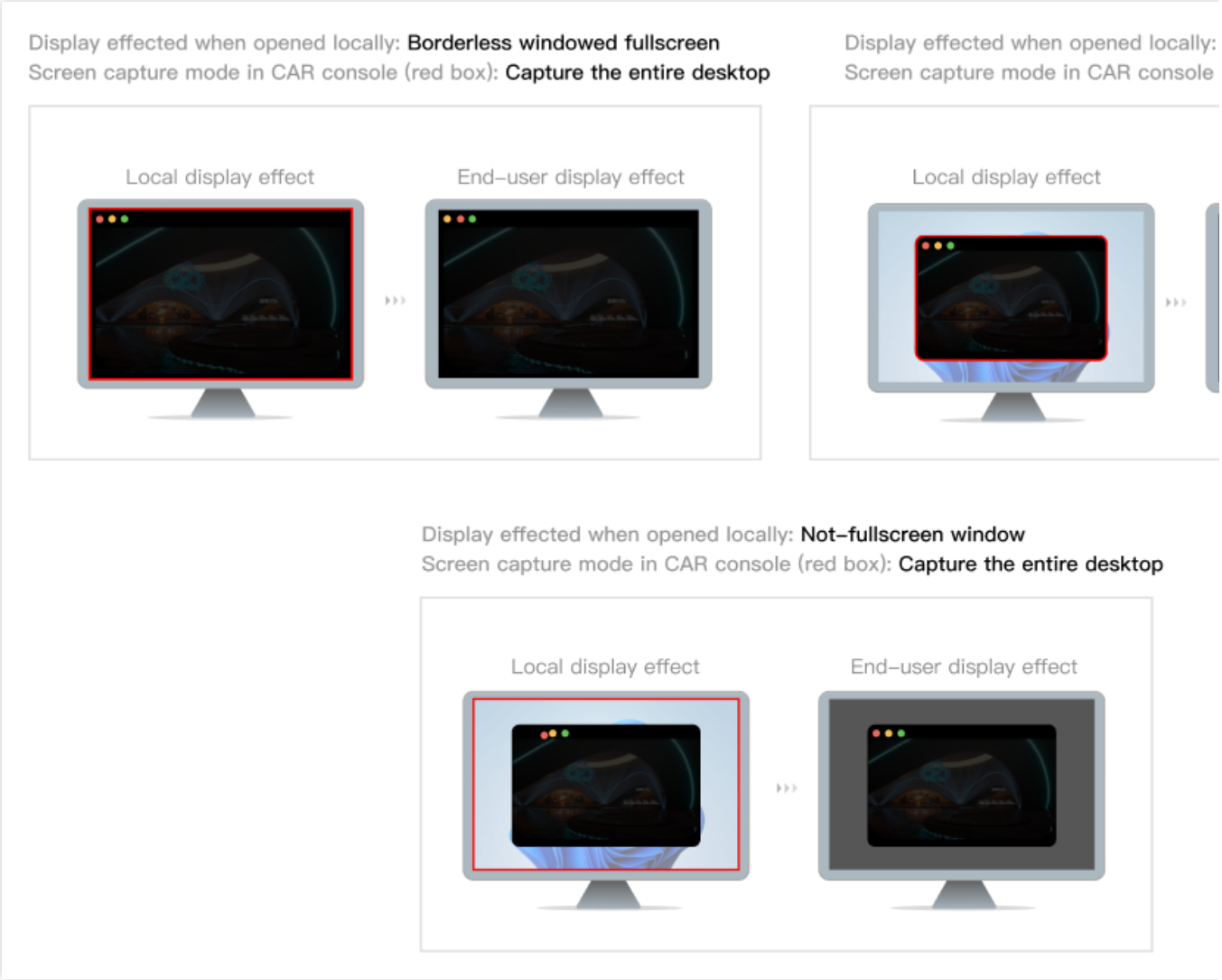


Path of the main executable file: To avoid filling in errors that may cause the application to fail to start, it is recommended that you wait for the application to be created and then click **Browse** to select the right path.

Application full-screen/adaptive resolution settings:

If you have adjusted the application to borderless fullscreen mode according to Step 1.1, set the `Screen Capture Mode` to `Capture the entire desktop`.

If your application only has a not-fullscreen window mode and cannot be adjusted to borderless fullscreen mode, you can use the `Capture a window` mode. You need to fill in the correct `Window title` and `Class name` according to the instructions. If the window title isn't customized during development, the window title for `Demo.exe` will be `Demo`. If your application is built by Unreal Engine, enter `UnrealWindow` for the Class name.



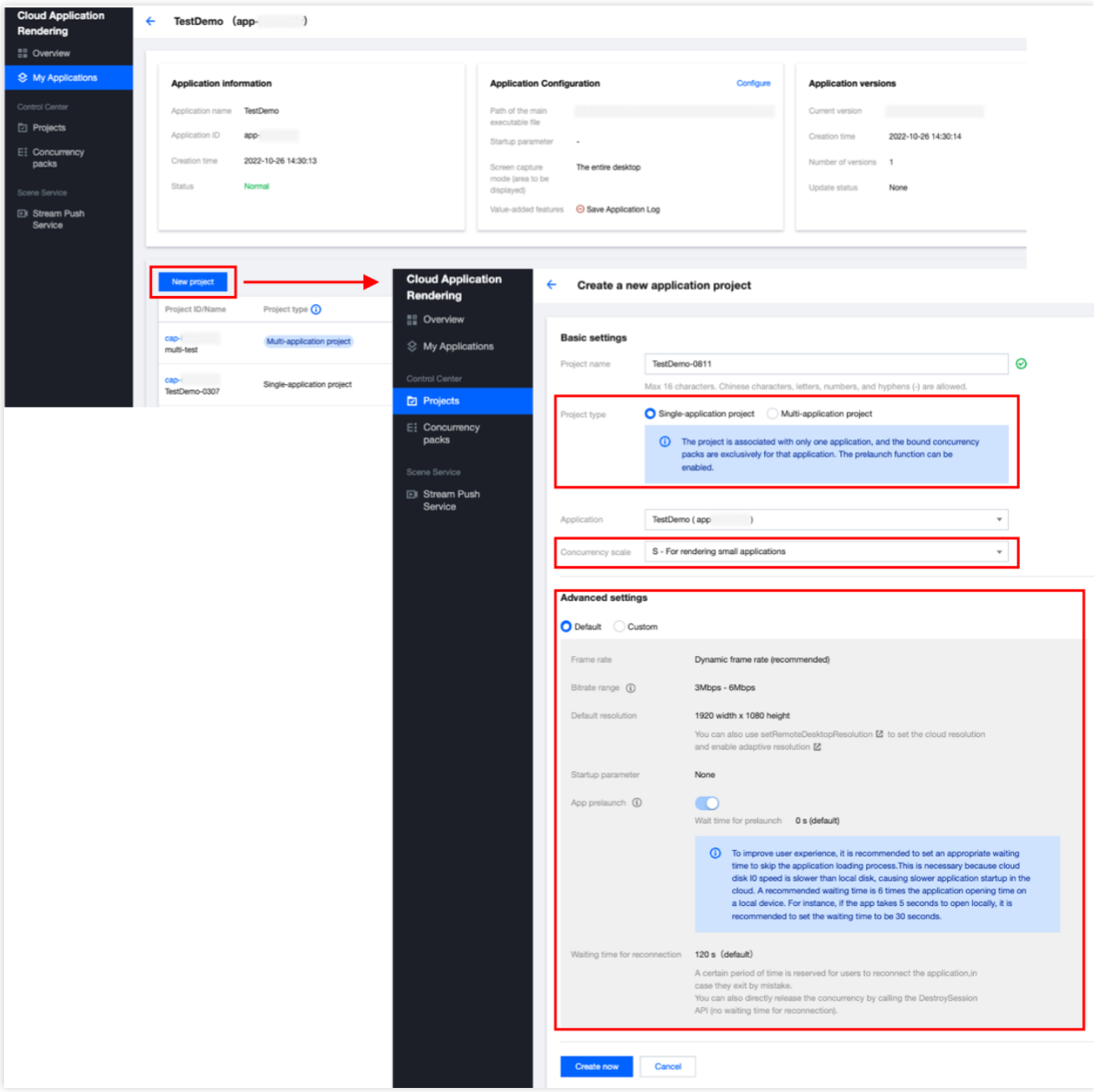
For detailed configuration instructions, see Application Configuration.

Step 2. Create a project and purchase concurrency packs:

Test Project and Free Concurrency Pack for New Users:

Before officially purchasing any concurrency pack, you can freely test the applications you uploaded using the Test Project. Free resources are limited, so each test is restricted to 2 minutes. For more information, please refer to the default Test Project and the free concurrency pack for new users.

1. If you wish to test other concurrency scales or purchase a concurrency pack for official use, please create a new project.



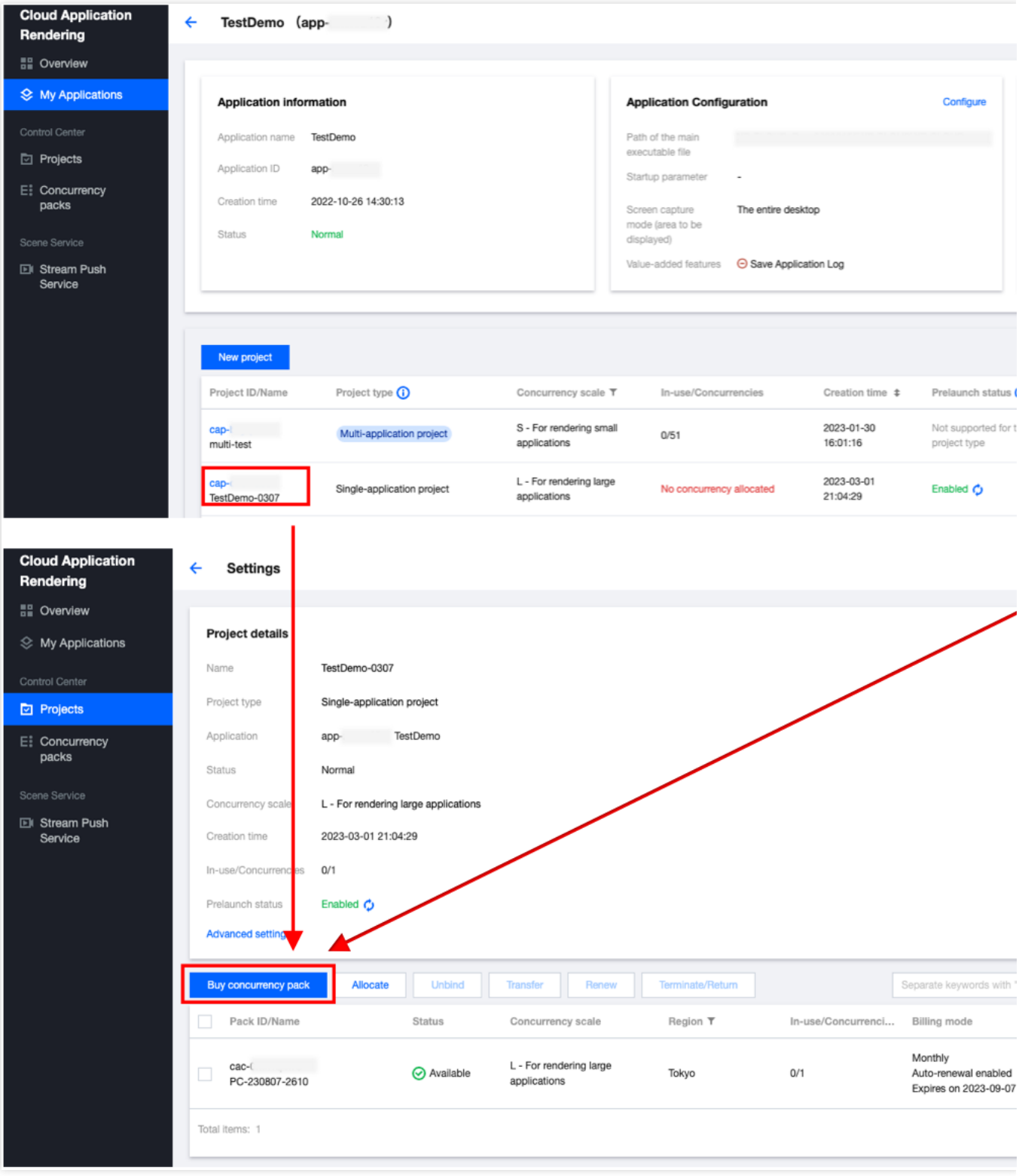
Project type: Supports two types of projects: single-application and multi-application. Single-application type ensures that the concurrency package under the project is only used by a single application, while the concurrency packs under a multi-application project can be shared by multiple applications as a "resource pool". Click to view how to achieve concurrency resource sharing through multi-application projects.

Concurrency scale: The concurrency scale under a project must be consistent. You can create multiple projects for an application and specify different concurrency scales to test the rendering effects of different concurrency scales. It is recommended to start testing from L scales to avoid problems such as stuttering caused by insufficient concurrency performance. For specific configurations of different concurrency scales, please refer to Billing.

Advanced project settings: Projects allow customization of parameters such as frame rate, bit rate range, default cloud desktop resolution, application startup parameters, prelaunch feature, and so on. You can adjust them

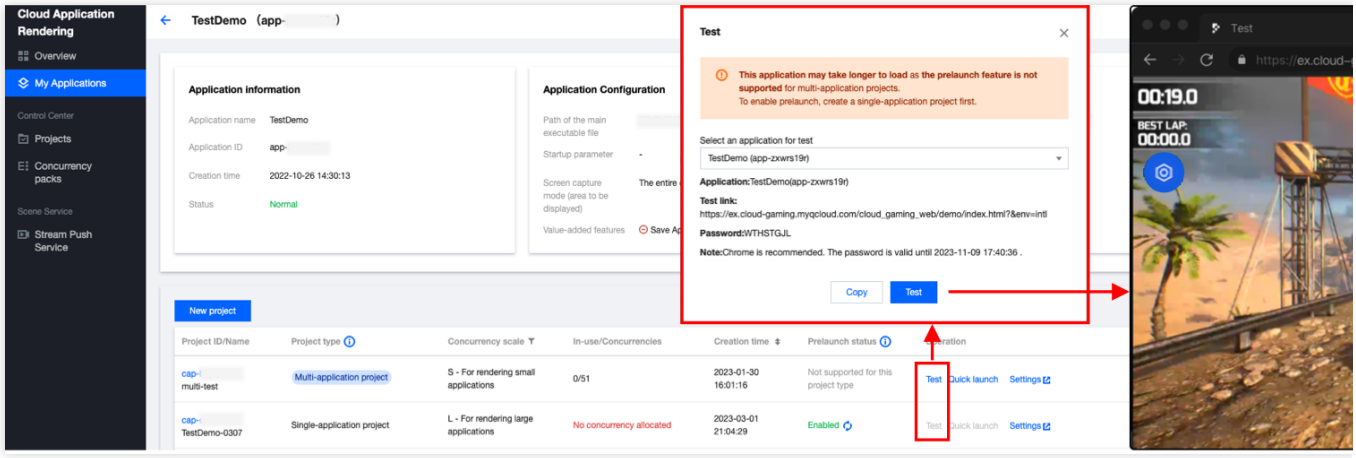
according to your needs. For details, please refer to the advanced project settings instructions.

2. Purchase the specified concurrency packs for the project and proceed to the next step.

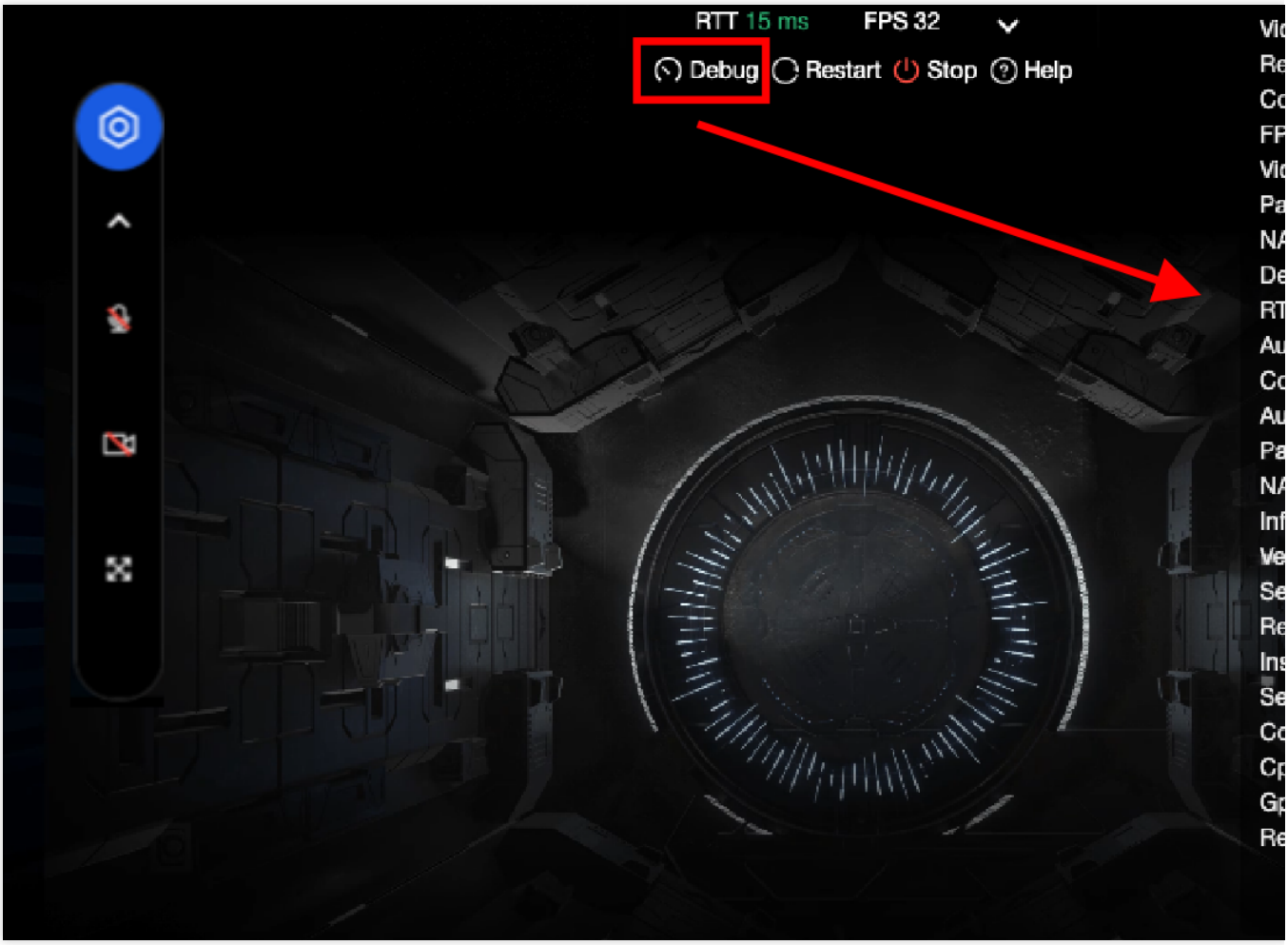


Step 3. Start a test:

Click **Test** to generate a test link and password. Open the test link to test the basic effect of operating your cloud application.



You can open the "Debug" tool (shortcut key `Ctrl+~`) in the toolbar of the test page and pay attention to the following data to choose the most suitable concurrency scale:



FPS (frames per second): The frame rate should be maintained at 30 or above under normal circumstances. If the FPS suddenly drops when entering certain specific scenes or performing specific operations, it may be due to a sudden increase in GPU computing power consumption that the current concurrency scale cannot handle, resulting in dropped frames. It is recommended that you try a higher concurrency scale.

RTT (round-trip time): If the RTT is higher than 100ms, there may be noticeable delays. It is recommended that you first check if there is any network jitter and try accessing it using a 4G/5G network. If the concurrency region is too far from your physical location, it will also cause high RTT.

Region (concurrency region): The area where the concurrency is located. If you purchase concurrency in multiple regions under a project, we will automatically schedule the nearest idle concurrency based on the end user's IP address. For example, if there are concurrencies in both Singapore and North America regions under a project, users in North America will connect to the concurrency in North America, while users in Southeast Asia will connect to the concurrency in Singapore.

InstanceType (concurrency scale): There are three concurrency scales: S, M, and L which are suitable for small, medium, and large applications, respectively. For detailed configurations, please refer to Billing. If your application requires high computing power but uses a lower concurrency scale, it may cause high CPU/GPU usage and result in stuttering, crashing, and other issues.

CpuUsage (CPU usage): If you find that the FPS data is dropping, check if there is 90-100% CPU usage. If so, it means that the concurrency scale is insufficient to handle the workload. Please try a higher concurrency scale.
GpuUsage (GPU usage): L=load, M=memory, E=encoder, D=decoder. Pay attention to the value of L (load). If you find that the FPS data is dropping, check if L appears with a value of 90-100%. If so, it means that the concurrency scale is insufficient to handle the workload. Please try a higher concurrency scale.
RequestId: If you encounter any problems/questions, please try to keep the connection and contact us, providing the RequestId.
Common issues encountered during testing:
A message is displayed indicating that there are no idle concurrencies:
Check whether there are any idle concurrencies under the project. When a user exits the application, it takes about one minute for the concurrency to be automatically cleared. Only then will the concurrency become idle so that a new user can connect to it.
The application cannot start / black screen:
If the cloud rendering service is stuck on loading and cannot be accessed, you can try testing it on a 4G network to see if the Wi-Fi or company intranet is restricting UDP. The cloud rendering service relies on UDP data transmission and reception, and it is necessary to open UDP port 60000-60100 to all source IPs on the client. If there are no special security concerns, we recommend you open all UDP ports.
If the message "Startup failed, please check the startup path configuration" is displayed, check if the main executable path of the application is correctly filled in.
If the application uses the Capture a window mode, check whether the window title and class name are correctly entered. If they are configured incorrectly, a black screen will occur.
The application starts slowly:
Generally, the prelaunch feature is used to load an application in advance so the application will already be running when a user connects to it. Because multi-application projects do not support prelaunch, the applications under those projects take a longer time to start.
For a single-application project, you can enable prelaunch so that the application will be loaded in seconds when the user connects to it.
When a user exits the application, the concurrency they were connected to will be repossessed, cleared, and reset, and when the concurrency becomes idle, the application will be prelaunched again. If the next user enters the application just when the concurrency becomes idle, the application may have not been completely prelaunched.
The RTT data is abnormal: We recommend that you first check for local network jitter. You can try accessing over a 4G/5G network. If the concurrency region is too far from your physical location, it will also cause high RTT. In this case, you may want to consider purchasing concurrency in a region closer to your physical location.
The application is slow or stuttering: This may be because your application has high requirements for computing power but the concurrency scale is low (such as S). We recommend you try a higher concurrency scale.
Prompt "Cloud rendering has been disconnected": If you are using your company's Wi-Fi network, it is recommended to check whether the network meets the CAR service environment, that is, the UDP ports have been

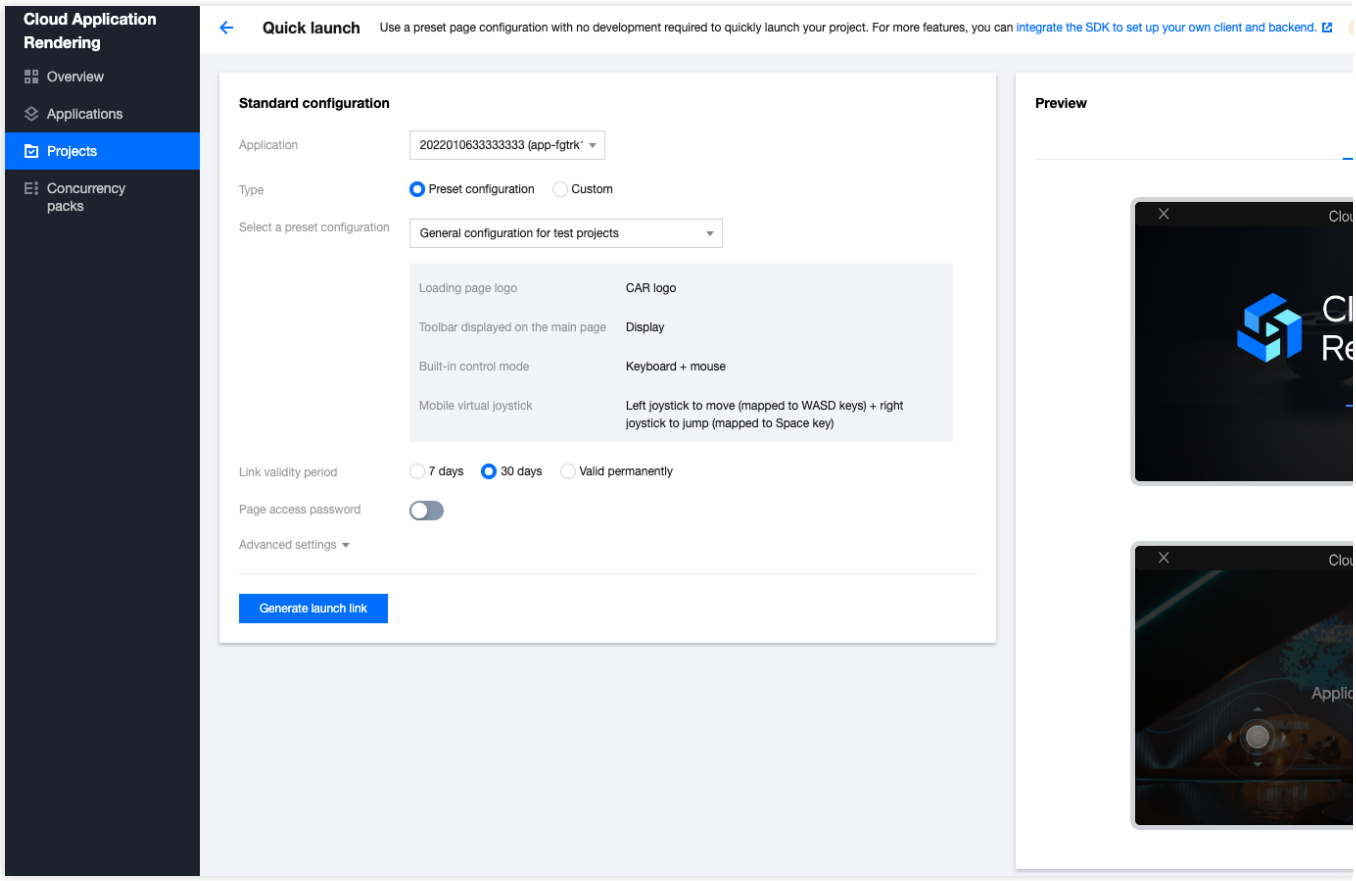
opened. At the same time, you can also check if there are any jitter issues with your local network and try accessing the link again via 4G/5G mobile networks.
If your problem still persists, contact us for assistance. For more FAQs, see Cloud Application FAQs.

Step 4. Launch project

After completing the test, you have determined the most suitable concurrency specification for your application and would like to create a client that can be officially provided to the end users. At this point, you can choose from the following two methods:

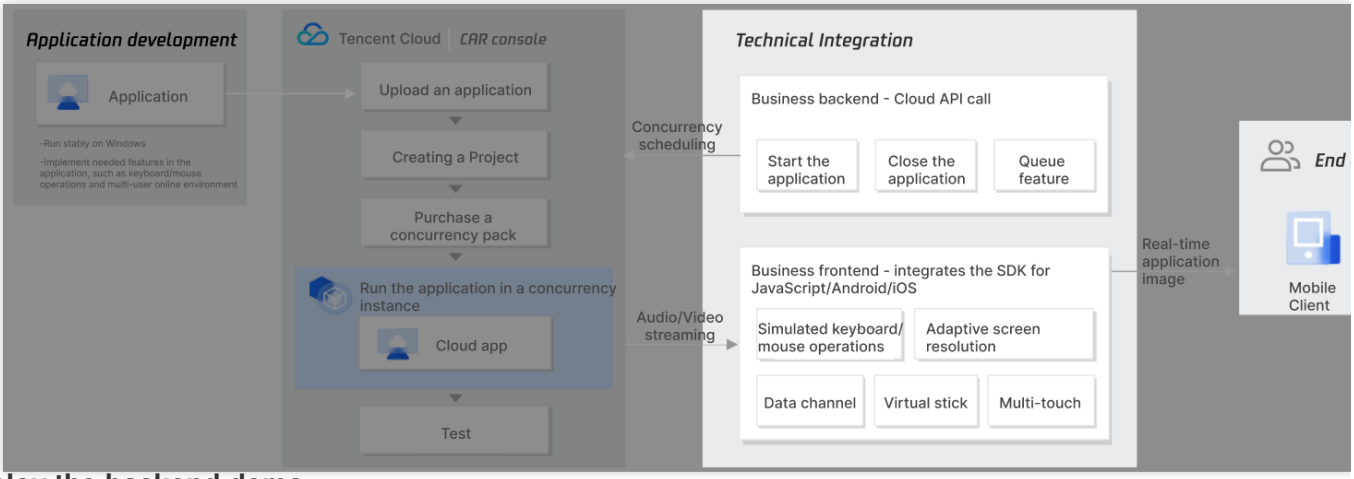
Method one. Quick Launch (no development required)

CAR supports standardized configuration of pages, allowing zero development to generate links for your project and application. For detailed operation steps, please refer to the quick launch guide.



Method two. Integrate CAR PaaS

To implement specific requirements and guarantee the optimal user experience, you might want to develop and set up your own backend service and client program and connect to CAR PaaS's APIs and SDKs for JavaScript, Android, and iOS. We provide demos for reference to help you quickly set up and deploy your own backend service and client program.



1. **Deploy the backend demo:**

Deploy the backend demo locally or on any server as instructed here. The demo is lightweight and has no special requirements for server configuration.

Deploying your own backend is a necessary step for using CAR services:

For the security of your assets and services, the `SecretId` and `SecretKey` of your Tencent Cloud account (which can be obtained on the Manage API Key page in the console) that are needed in order to access the CAR TencentCloud API service must be processed on your backend service. In addition, management of user sessions and other features such as user queue must be implemented on your backend service. For more information, see Basic logic of the business backend and client.

2. **Deploy the client demo:**

Taking the demo for JavaScript as an example,

2.1 Download TCGSDK to C or D drive on the local PC or the `Downloads` folder on Mac and keep the path of the folder such as `c:/cloudgame-js-sdk` .

2.2 Download the demo HTML file.

2.3 Replace the following paths/parameters in the demo. For more information, see Tutorial: Cloud Application Rendering web demo:

Paths/parameters that need to be replaced	Note
`src="path/to/tcg-sdk"`	TCGSDK path, for example: `src="./cloudgame-js-sdk/dist/tcg-sdk/index.js"`
`url = 'http://xxxx/StartProject'`	Replace `xxxx` with the URL of the business backend demo such as `url = 'http://192.168.0.1:3000/StartProject'`
`ProjectId: 'cap-12345'` `ApplicationId: 'app-12345'` `UserId: 'user-id'`	`ProjectId` : The ID of the project created in the console, prefixed with "cap-". `ApplicationId` : The ID of the application created in the console, `UserId` : The unique ID of the end user currently connected to CAR, which is customized by you and is not understood by CAR. It can be

	randomly generated with the timestamp and should be kept unchanged during user reconnection.

2.4 For mobile operations, CAR provides the Joystick plugin to map WASD/up, down, left, and right arrow keys.
2.5 We recommend that you integrate the user queue system to improve the user experience when the number of users requesting your application becomes greater than the maximum number of concurrencies. The queue logic involves the frontend and backend. For the frontend code, see Queue. The signature logic can be added based on your business needs. For the backend implementation, see Queue Feature.
3. **Test, debug, and launch your application:**
After correctly performing the operations in the console and successfully deploying your backend and client services, you can open the demo HTML file in Chrome (recommended). The rendered application image will be displayed after being loaded successfully, and you can then interact with the application with the mouse/WASD.
You can perform more custom development based on the demo. If you encounter any problems during the testing process, contact us and provide the RequestId.
Relevant guides:
Technical Integration: Contains detailed directions on implementing the queue feature, data channel, heartbeat connection, adaptive resolution, enabling the mic, etc.
JavaScript SDK Documentation: Contains the demos for JavaScript and detailed documentation of the rich TCGSDK APIs, making it easier for you to perform custom development.
Android SDK Documentation: Contains detailed documentation of the SDK for Android as well as three demo projects and an APK for trial purposes.
iOS SDK Documentation: Contains detailed documentation of the SDK for iOS and a simple demo for integrating the SDK.

Appendix

Basic logic of the business backend and client

1. **The client initializes the CAR SDK**:
No matter which type of client you use, you can get `ClientSession` after the SDK is initialized successfully. `ClientSession` will be used by the business server to get `ServerSession` subsequently. The specific initialization and acquisition methods for each client type are as detailed below:
SDK for JavaScript: The business client calls the TCGSDK.init(params) API to perform initialization. After initialization, the client calls the TCGSDK.getClientSession() function to get its `ClientSession` .
SDK for Android: Call the TcrSdk.getInstance().init(context, null, callback) API to perform initialization. After initialization, the client gets its `ClientSession` in the TcrSession.Observer#onEvent callback.
2. **The backend service reserves a concurrency**:
Your backend service calls the CAR API ApplyConcurrent() to reserve a concurrency and proceeds to the next step

after receiving the success callback.

3. **The backend service gets `ServerSession`** :

Your backend service calls the CAR API CreateSession(ClientSession) to get `ServerSession` in the success callback and return it to the client.

4. **Start CAR**:

The call method to start CAR after `ServerSession` is received varies slightly for SDKs for different client types. See the following guides based on your specific client type:

SDK for JavaScript: The client calls the TCGSDK.start(ServerSession) function to start CAR.

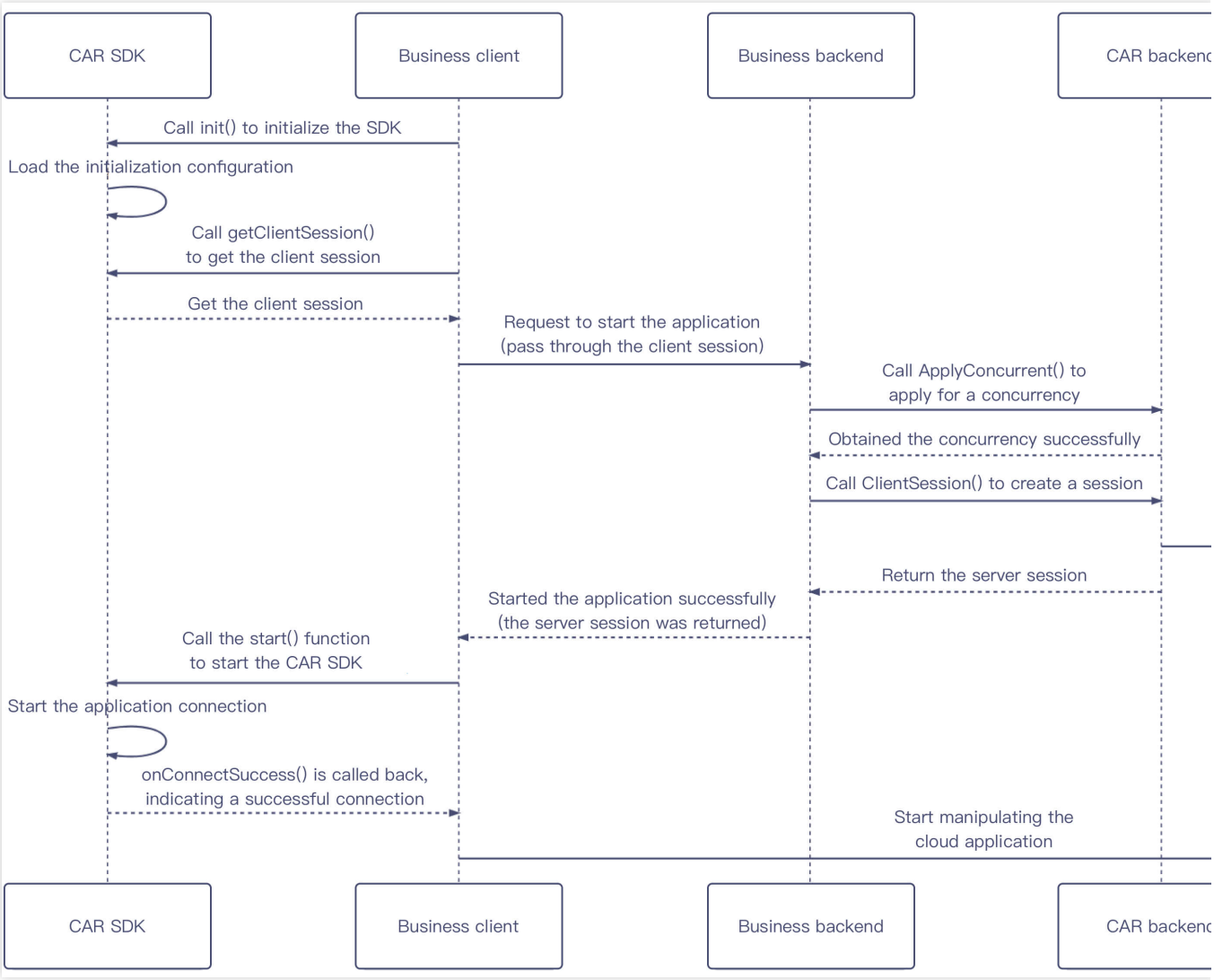
SDK for Android: The client calls the TcrSession.start(serverSession,callback) function to start CAR.

Technical Integration
Starting an Application

Last updated：2024-01-26 11:54:09

This section describes how to start a cloud application.

Sequence diagram



Parameter description

Role	Description
CAR SDK	CAR client SDKs include SDK for JavaScript, SDK for Android, and SDK for iOS. This section takes the JavaScript SDK as an example.

Business client	The platform you provide for your users. You can see Step 3 in Getting Started and for reference when setting up your business client.
Business backend	Your backend service that can be deployed on any server. You can see Step 3 in Getting Started for reference when setting up your backend.
CAR backend	Tencent Cloud APIs provided by CAR.
Cloud application	The application you uploaded.

Directions

1. The business client calls the TCGSDK.init() API to perform initialization. After initialization, the client calls the TCGSDK.getClientSession() API to get its `ClientSession` .

2. The business client requests the business backend to start the application based on parameters passed in such as `UserId` and `ClientSession` . Here, `UserId` is the unique user identifier customized by the business and should remain unchanged during user queuing and application reconnection.

Note:

A concurrency can be used by only one `UserId` at a time. If you use the same `UserId` to send requests, concurrency contention may occur. The CAR backend sends the currently connected concurrency to the device sending the latest request, and the previous device will be disconnected and display a black screen.

3. The business backend calls the `[ApplyConcurrent()](https://cloud.tencent.com/document/product/1547/72827)` API through the Real-Time Cloud Rendering API to apply to reserve a concurrency. If there are no available concurrencies or an error is returned, go back to Step 2 to send a request again.

4. The business backend calls the CreateSession() API to create a session and returns `ServerSession` to the business client.

5. The business client calls the TCGSDK.start() API to start the cloud application. After the application connection, the SDK will trigger the `onConnectSuccess()` callback. We recommend you call the data channel creation API and APIs for other features after this callback.

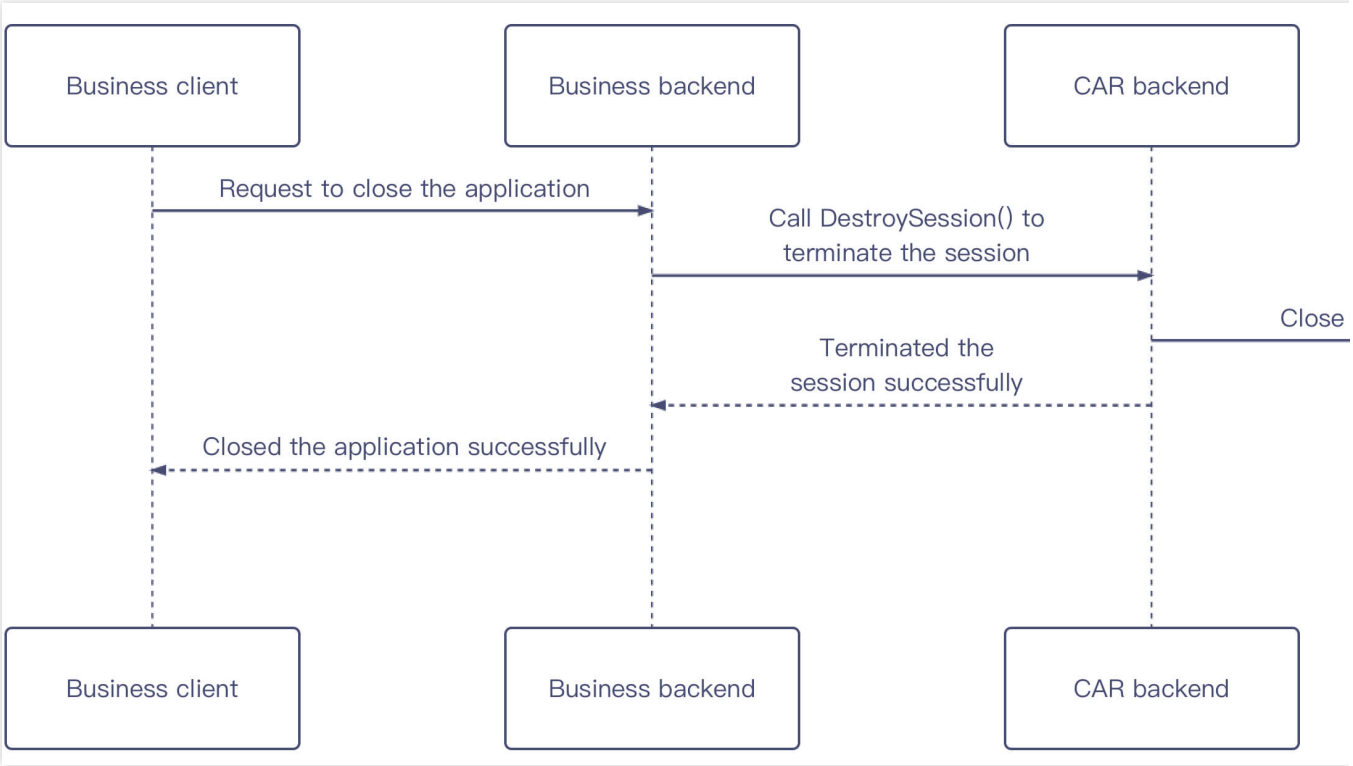
Closing the Application

Last updated：2024-01-26 11:54:09

Closing the Application

This section describes how to close the application. We recommend you release concurrencies promptly when the application is closed, so that the concurrencies can be made available to other users more quickly.

Sequence diagram



Directions

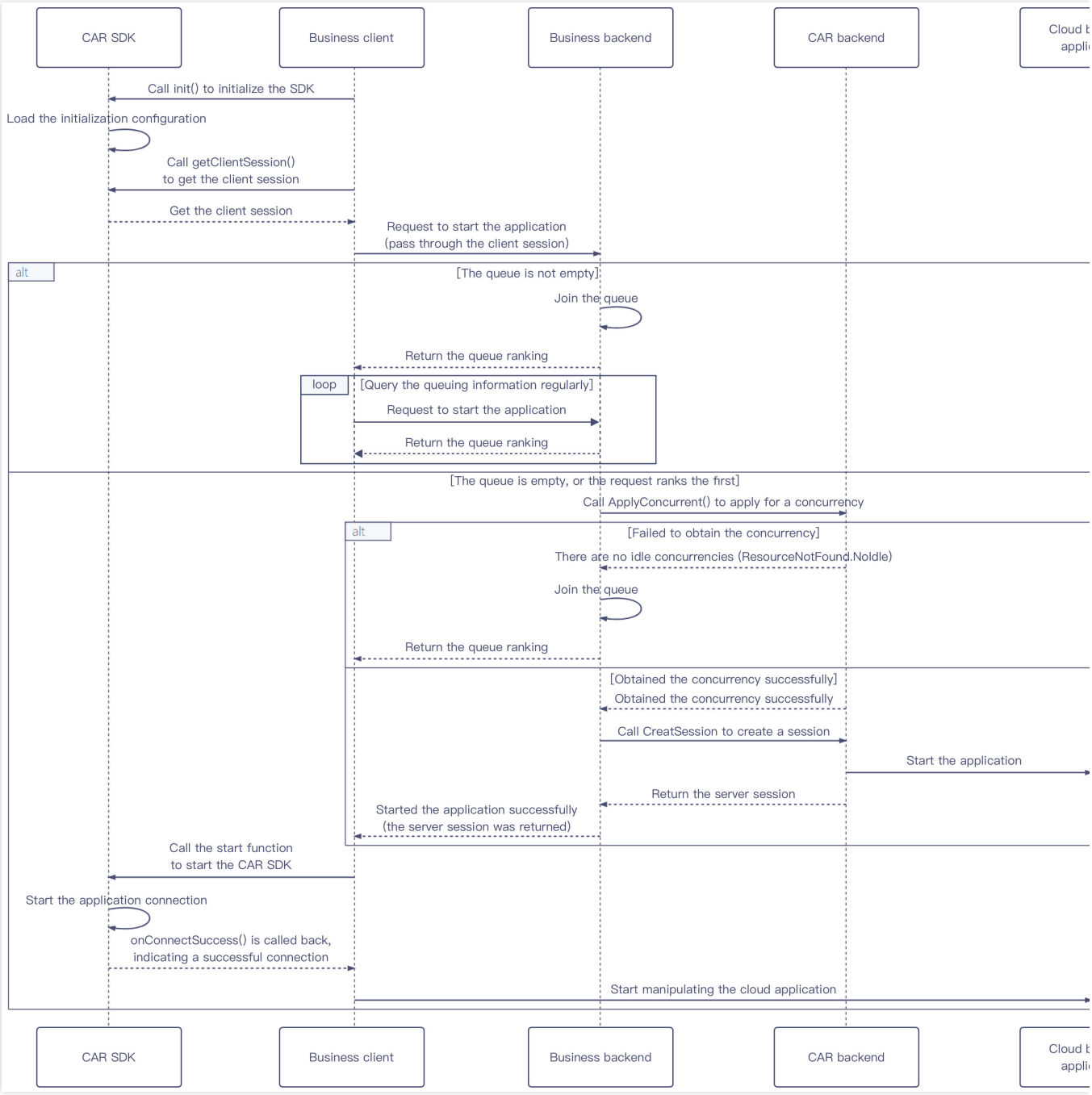
1. The business client actively requests the business backend to close the application.
2. The business backend calls the DestroySession API to terminate the session and close the cloud application actively. Then, the business client disconnects from the cloud application.

Queue Feature

Last updated：2024-01-26 11:54:09

This section describes how to integrate the user queue system. We recommend you integrate the user queue system to improve the user experience when the number of users requesting your application becomes greater than the max number of concurrencies.

Sequence diagram



Directions

1. The business client calls the TCGSDK.init() API to perform initialization. After initialization, the client calls the TCGSDK.getClientSession() API to get its `ClientSession`.
2. The business client requests the business backend to start the application based on parameters passed in such as `UserId` and `ClientSession`. Here, `UserId` is the unique user identifier customized by the business and should remain unchanged during user queuing and application reconnection.
The business backend needs to perform the following operations based on the current queuing status:

If the queue isn't empty, add the user to the queue and return the current queue ranking information. The business client regularly initiates requests after receiving the queue information and repeats step 2 until the application is started successfully.
If the queue is empty or the user ranks the first, proceed to step 3.
3.
The business backend calls the ApplyConcurrent() API to reserve a CAR concurrency and performs the following steps based on the returned result:

If there are no idle concurrencies or another error is returned, add the user to the queue, return the current queue ranking information, and go back to step 2 to initiate a request again.
If the concurrency is obtained successfully, proceed to step 4.
4.
The business backend calls the CreateSession() API to create a session and returns `ServerSession` to the business client.

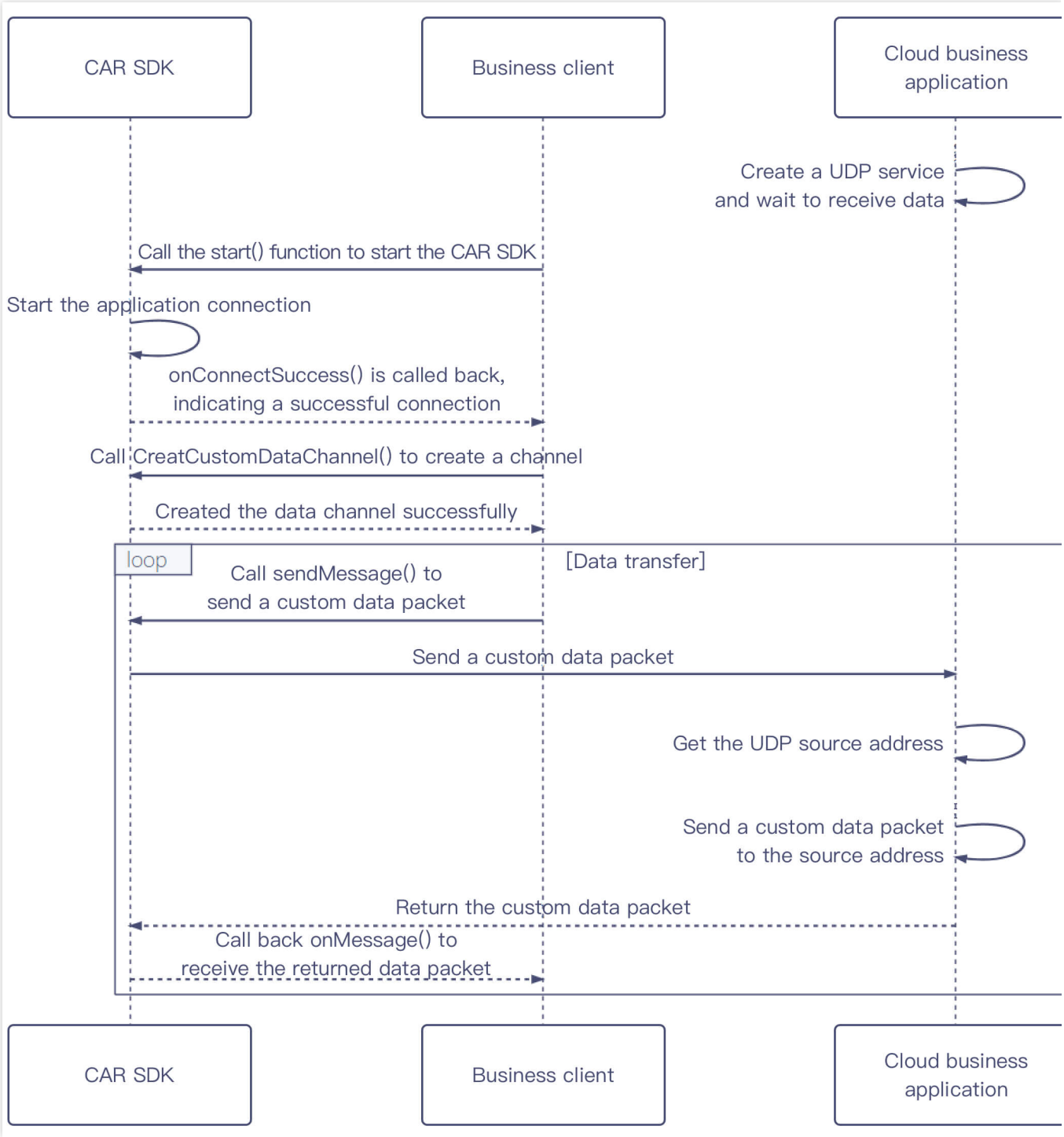
5. The business client calls the TCGSDK.start() API to start the cloud application. After the application connection, the SDK will trigger the `onConnectSuccess()` callback. We recommend you call the data channel creation API and APIs for other features after this callback.

Data Channel

Last updated：2024-01-26 11:54:09

This document describes how to use a data channel to establish communication between the business client and a cloud application and transfer startup parameters, commands, messages, or other data.

Sequence diagram



Directions

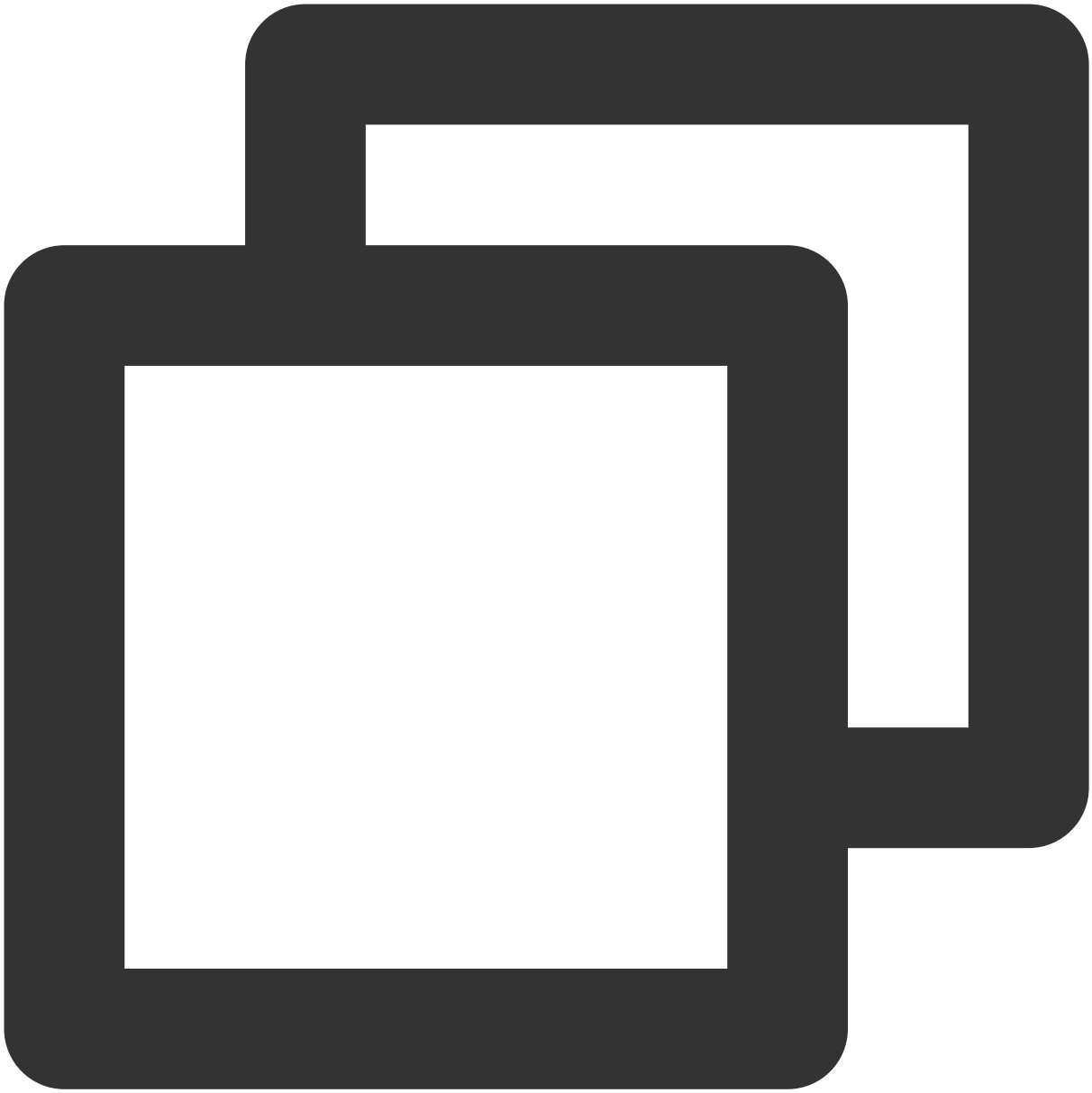
1. Create a UDP service for the cloud business application to listen for a local UDP port (recommended port range: 10000–20000 for `localhost 127.0.0.1`) and wait to receive UDP packets.
2. The business client calls the TCGSDK.start() API to start the cloud application. After the application connection, the `onConnectSuccess()` callback is triggered. We recommend you create a data channel for the business client after this callback.

3. The business client calls the TCGSDK.createCustomDataChannel() API to create a passthrough channel. The target port parameter in the API must be the port listened for by the cloud application in step 1. If the data channel fails to be created, repeat this step until it is created successfully.
4. After the data channel is created successfully, the business client can call `sendMessage()` to send a data packet customized by the business. The UDP
service of the cloud application receives the request and parses the UDP source address.

5. The cloud application sends a custom data packet to the UDP source address obtained in step 4. The data packet will then be returned through the created data channel. The business client can process the returned packet in the `onMessage()` callback API.

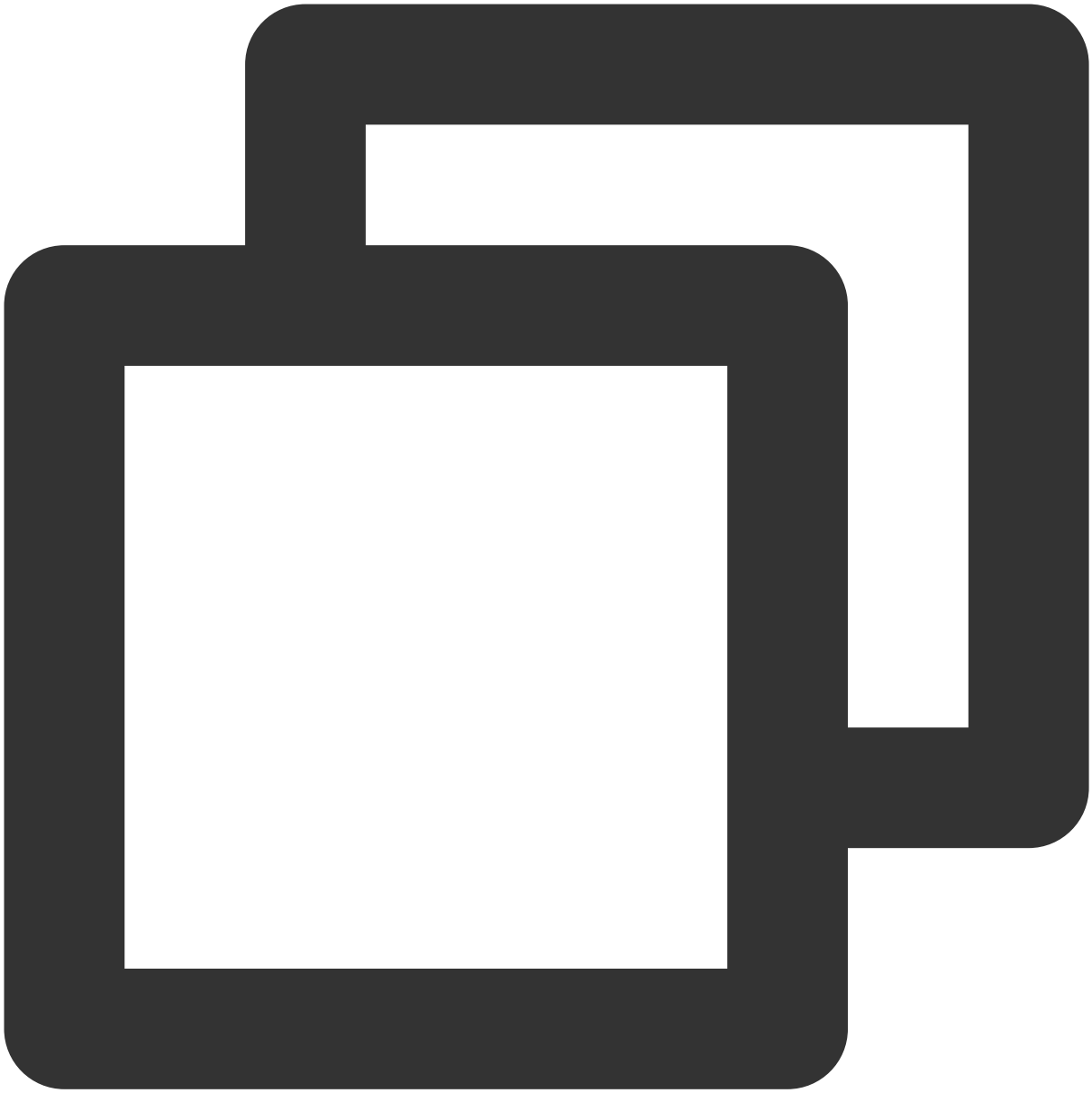
Sample code

Business client (JavaScript SDK example)
Cloud application (C/C++ example)



```
let timer = null;

const { sendMessage, code } = await TCGSDK.createCustomDataChannel({
  destPort: xxxx, // The recommended port range of destPort is 10000-20000.
  onMessage: (res) => {
    console.log('CustomDataChannel onMessage', res);

    // The receipt of this callback indicates that the cloud application has been s
    // clearInterval(timer);
  },
});
```

```
// The code value 0 indicates successful establishment of a data channel between th
if (code === 0) {
  // Send custom data.
  sendMessage('test');
}

if (code === 1 || code === -1) {
  // Retry upon failed creation.

  // timer = setInterval(() => {
  //   sendMessage('test');
  // }, 5000);
} else {
  // Contemplate re-establishing the data channel.
}
```



```
int main() {
    int udp_socket_fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (udp_socket_fd == -1) {
        printf("socket failed!\\n");
        return -1;
    }

    // Set the destination IP address.
    struct sockaddr_in bind_addr = { 0 };
    bind_addr.sin_family = AF_INET;
    bind_addr.sin_port = htons(xxxx); // The recommended port range in htons(xxxx)
```

```
    bind_addr.sin_addr.s_addr = inet_addr("0.0.0.0"); // Bind the IP

    // Bind the port.
    int ret = bind(udp_socket_fd, (struct sockaddr *)&bind_addr, sizeof(bind_addr))
    if (ret < 0) {
        perror("bind fail:");
        close(udp_socket_fd);
        return -1;
    }

    // Start to wait for a client message.
    struct sockaddr_in  upstream_addr = { 0 };  // Address used to store the CAR pr
    int len = sizeof(upstream_addr);
    char buf[1024] = { 0 }; // Receive buffer.
    while (true) {
        ret = recvfrom(udp_socket_fd, buf, sizeof(buf), 0, (struct sockaddr *)&upst
        if (ret == -1) {
            break;
        }
        // buf is the message "test" sent by the frontend.
        // upstream_addr can be subsequently used to send messages to the frontend.
        const char* response = "response";
        sendto(udp_socket_fd, response, strlen(response), 0, (struct sockaddr *)&up
    }
    return 0;
}
```

FAQs

The data channel is successfully created, the business frontend successfully sends data, but the response from the cloud application is not received. What should I do?

A successful call to the TCGSDK.createCustomDataChannel() API indicates that a data channel has been successfully established between the business frontend and the cloud. However, it is possible that the cloud application may not be fully initiated at this time. The business frontend can send custom data via timeout, interval, polling, etc., to ensure that the cloud application normally receives the custom data sent by the business frontend once it is successfully initiated. As long as the data channel is successfully created, data can be sent successfully by default.

If you confirm that the cloud application data is received in the `onMessage` callback, you can cancel the polling and then send the data normally.

What types of data can be sent and received?

The `sendMessage()` API supports data types such as string and ArrayBuffer.

Is there any limit on the size of a data packet to be transferred?

There is no limit on the size of a data packet to be transferred. However, it must be noted that the UDP packet can be of up to 64 KB and it is suggested that the packet size be kept less than MTU 1500. If the packet size is too large, it is recommended to use packet splitting for the transmission, as too much upstream data could potentially cause congestion and affect the user experience.

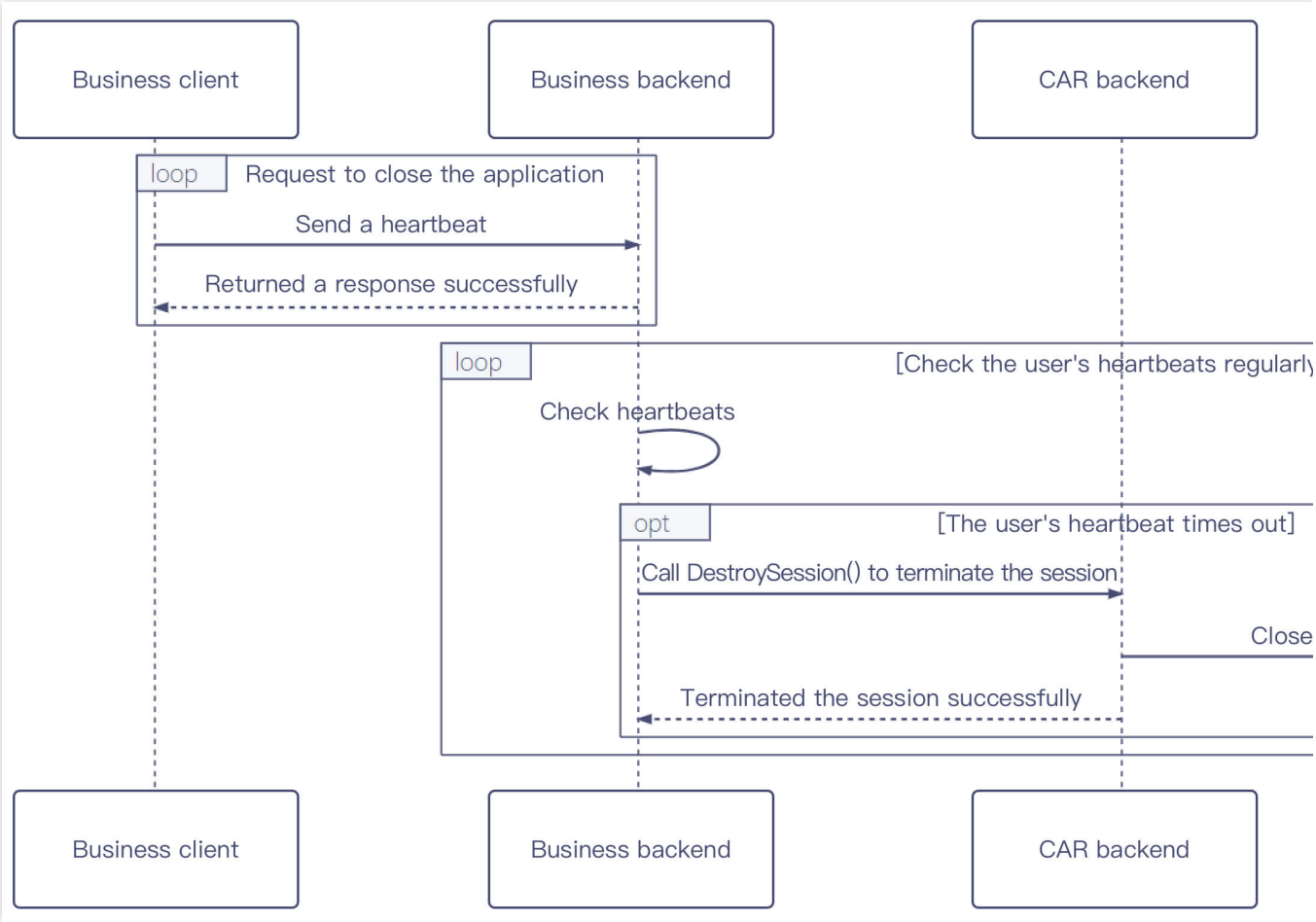
Heartbeat Connection

Last updated：2024-01-26 11:54:09

Heartbeat Connection

This section describes how to maintain a heartbeat connection between the client and the backend. The heartbeat can be used to detect whether the user is still connected and used to collect and manage data such as user connection duration. In this way, if the user exits due to an error, the backend can more quickly repossess the concurrency the user was occupying, and make it available to other users.

Sequence diagram



Directions

1. The client regularly sends heartbeat messages to the backend. The backend maintains the user connectivity based on the heartbeat messages.
2. The backend regularly checks whether the user is still connected. If the user heartbeat times out, the backend will call the DestroySession() API to terminate the session and close the application, and the client will disconnect from the application.

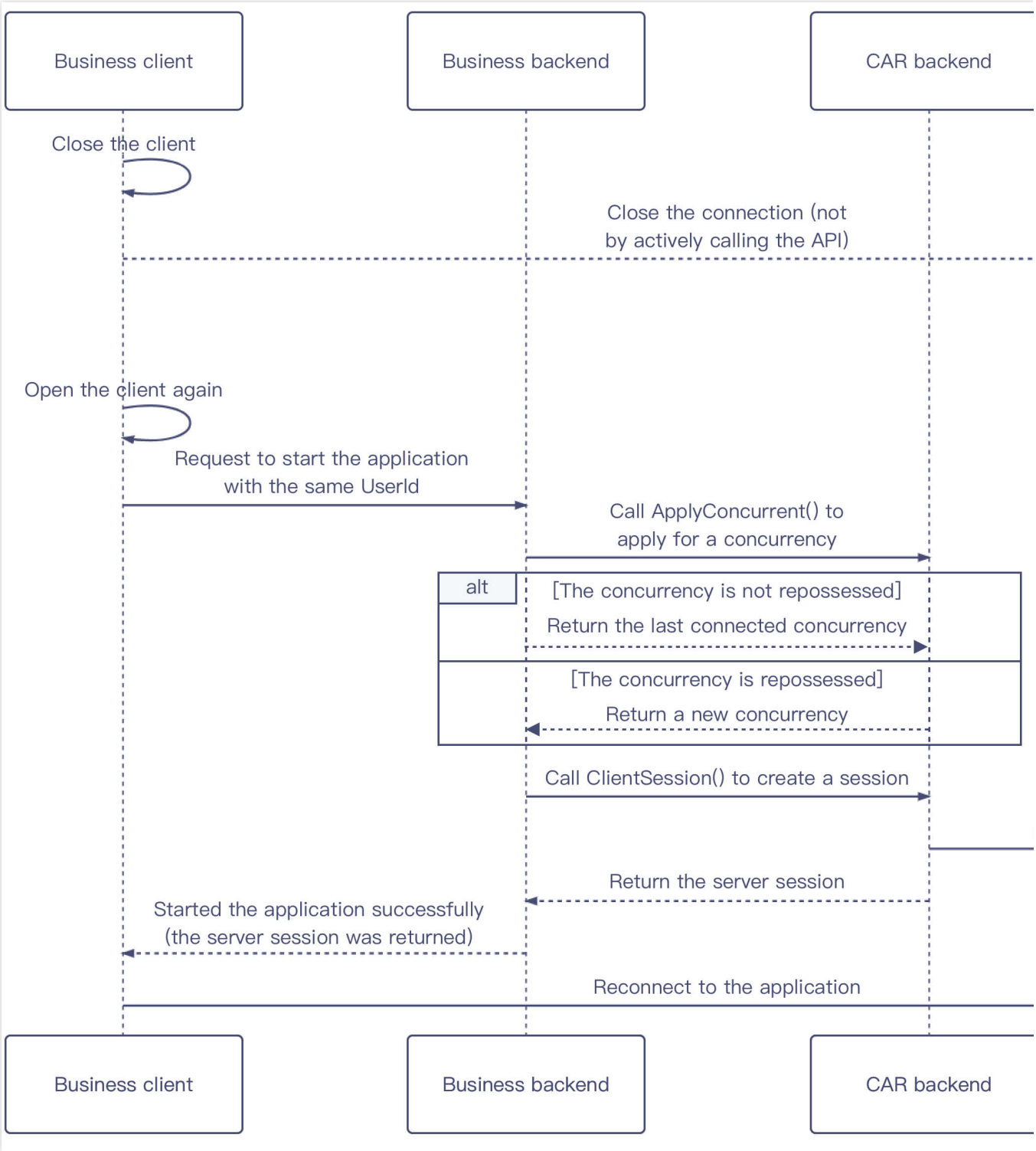
Application Reconnection

Last updated：2024-01-26 11:54:09

Application Reconnection

This section describes how to reconnect to an application. If a user directly closes the client but doesn't actively call the application close feature, APIs for applying for a concurrency and creating a session can be called again to reconnect to the application.

Sequence diagram



Directions

1. After the client is closed, it will disconnect from the cloud application. If the backend doesn't actively call the API to close the application, the concurrency will wait for 90 seconds by default, so that there is time to reconnect.
2. After the client is opened again, it will pass in the same `UserId` to request the backend to start the application. If the `UserId` values are different, it cannot reconnect the user to the application.

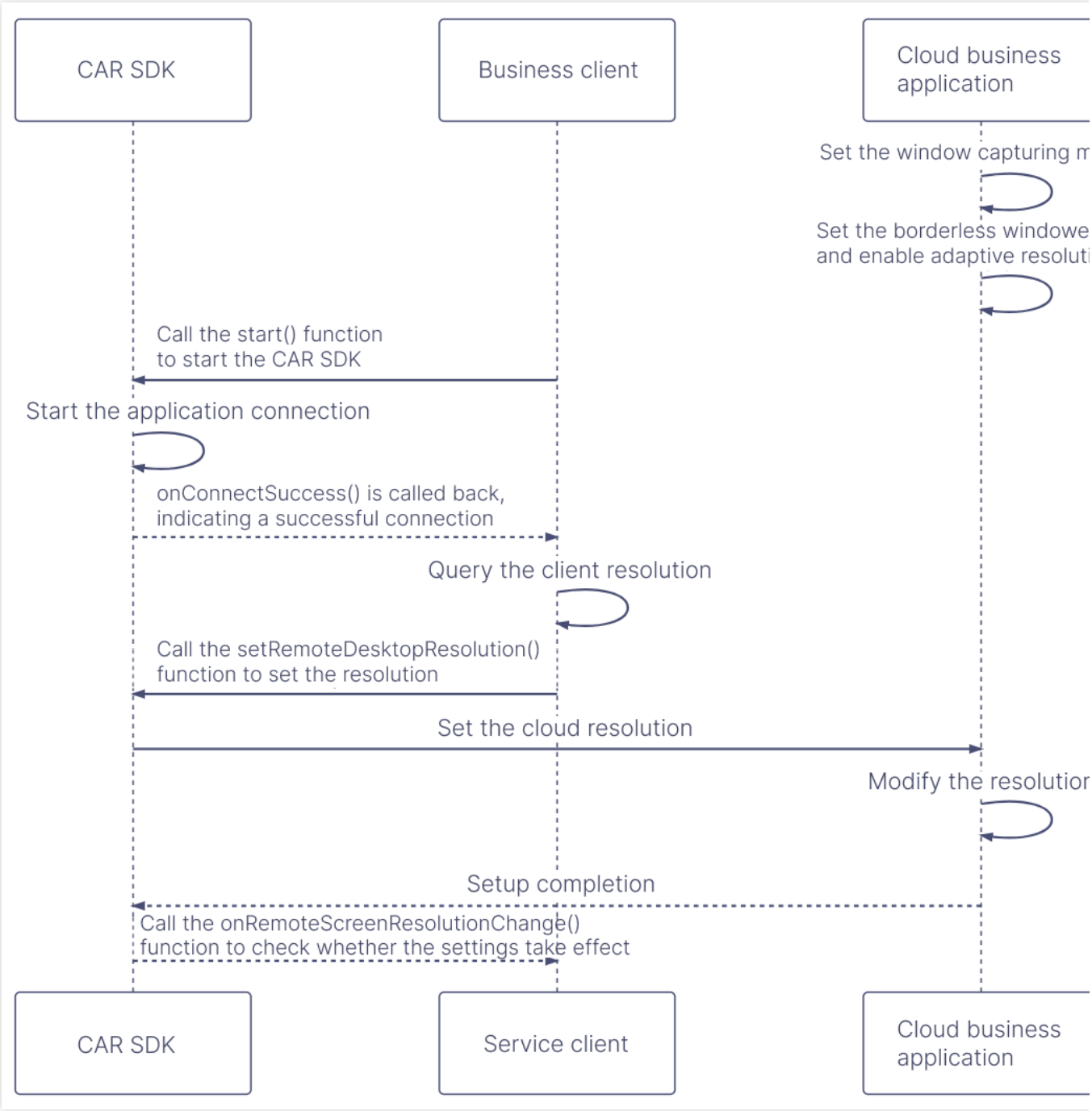
3. The backend calls the ApplyConcurrent() API to apply to reserve the concurrent and perform the following steps based on the concurrency's status:
If the user's last connected concurrency hasn't been repossessed, the last concurrency will be returned.
If the user's last connected concurrency has already been repossessed, a new concurrency will be returned.
4. The backend calls the CreateSession() API to create a session and returns `ServerSession` to the client to restart the application.

Adaptive Resolution

Last updated：2024-01-26 11:54:09

This document describes how to support adaptive resolution for cloud applications. This feature can be used to make an application adapt to different client resolutions and achieve a fullscreen display without black bars.

Sequence diagram



Directions

Recommended method

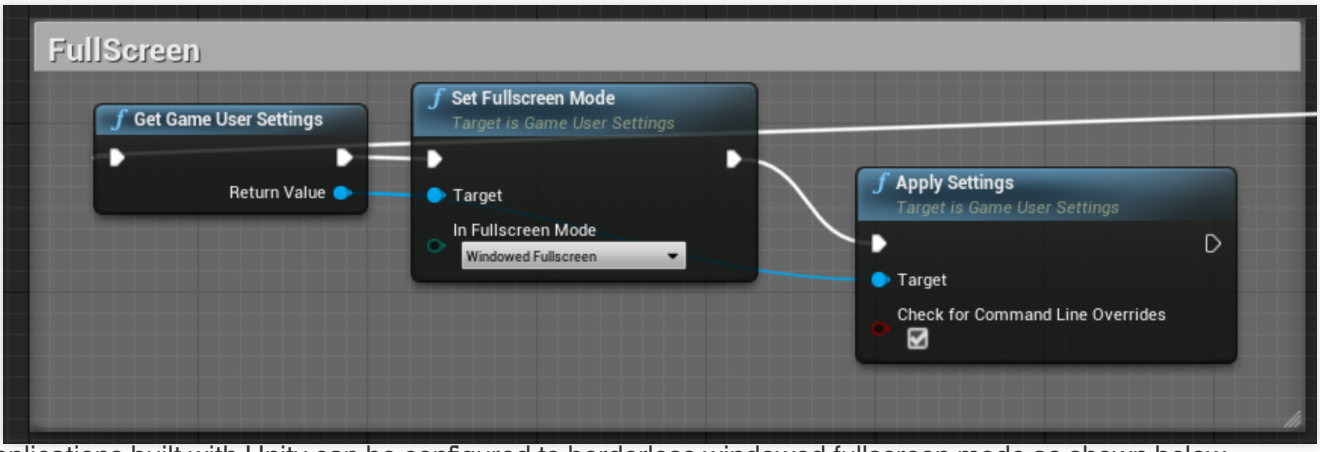
1. Prerequisites:

1.1 Before you upload an application to CAR, **we recommend you first set the application to borderless windowed fullscreen mode and enable adaptive resolution**.

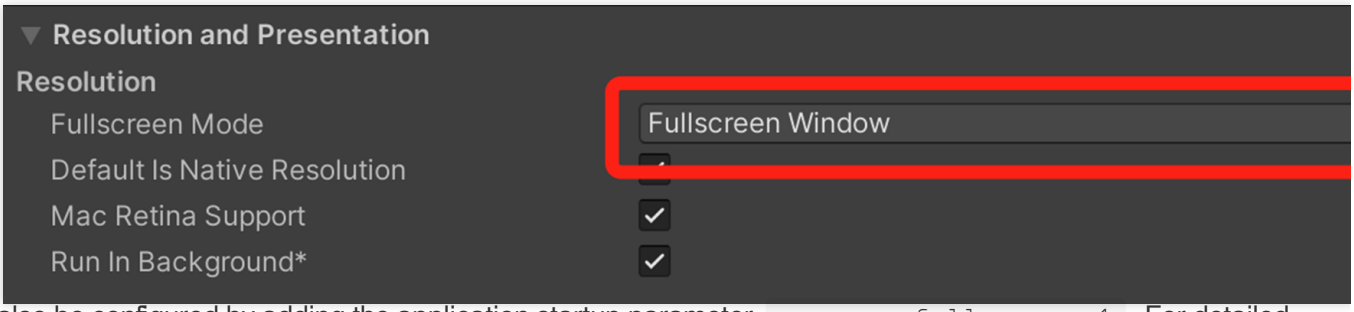
Note:

This operation must be done by adjusting settings within the application.

For applications built with Unreal Engine, you can add the blueprint below to your Map's BeginPlay to enable adaptive desktop resolution. Make sure `In Fullscreen Mode` is configured as `Windowed Fullscreen`.



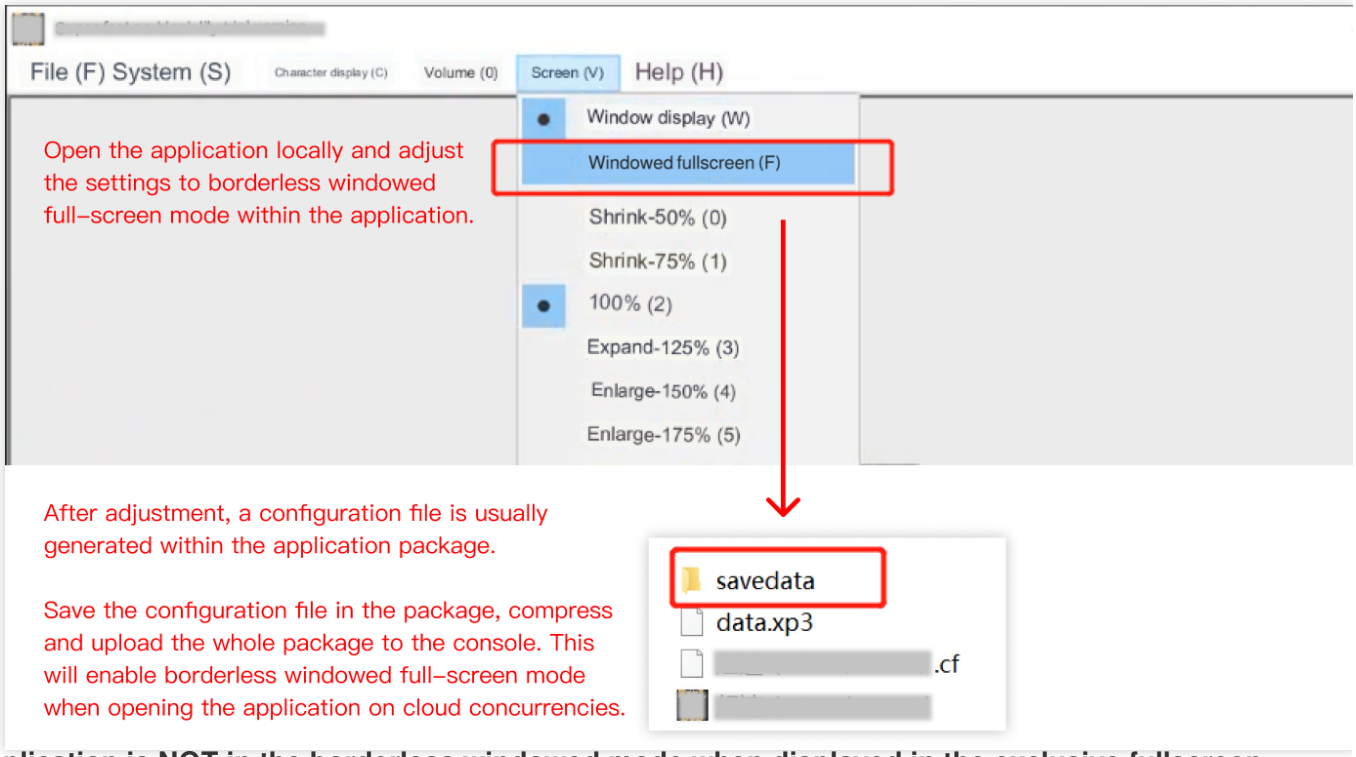
The applications built with Unity can be configured to borderless windowed fullscreen mode as shown below.



It can also be configured by adding the application startup parameter `-screen-fullscreen 1`. For detailed parameters of Unity applications, see here.

`-screen-fullscreen`	Override the default full-screen state. This n

For some applications, the display mode can be adjusted in the application settings. After adjusting the settings, you can update the application version in the console.



An application is NOT in the borderless windowed mode when displayed in the exclusive fullscreen mode. In CAR, adaptive resolution is achieved by modifying the **cloud desktop resolution**. An application in exclusive fullscreen mode has full control over the display resolution. In this case, a crash may occur when the display resolution is modified.

How to differentiate between borderless windowed mode and exclusive fullscreen mode? Press **Alt+Tab** to switch to a different window. Borderless windowed applications do not cause the display to flicker when the window is switched, while exclusive fullscreen applications do cause the display to flicker.

1.2 In the CAR console, go to the application configuration and set the screen capture mode to **Capture the entire desktop**.

2. The business client calls the TCGSDK.start() API to start the cloud application. After it connects to the application, the `onConnectSuccess()` callback is triggered. We recommend you set the resolution for the client after this callback.

3. The business client calls the TCGSDK.getPageSize() API or other methods to get the client resolution.

4. After getting the resolution, the client calls the TCGSDK.setRemoteDesktopResolution() API to set the cloud resolution.

5. After the request to set the resolution is received in the cloud, **the cloud desktop resolution is modified, and the cloud application adapts to the resolution change**.

6. The business client calls the `onRemoteScreenResolutionChange()` API to check whether the resolution settings have taken effect.

Adapting to different client resolutions if the application is only displayed in a non-fullscreen window (not recommended)

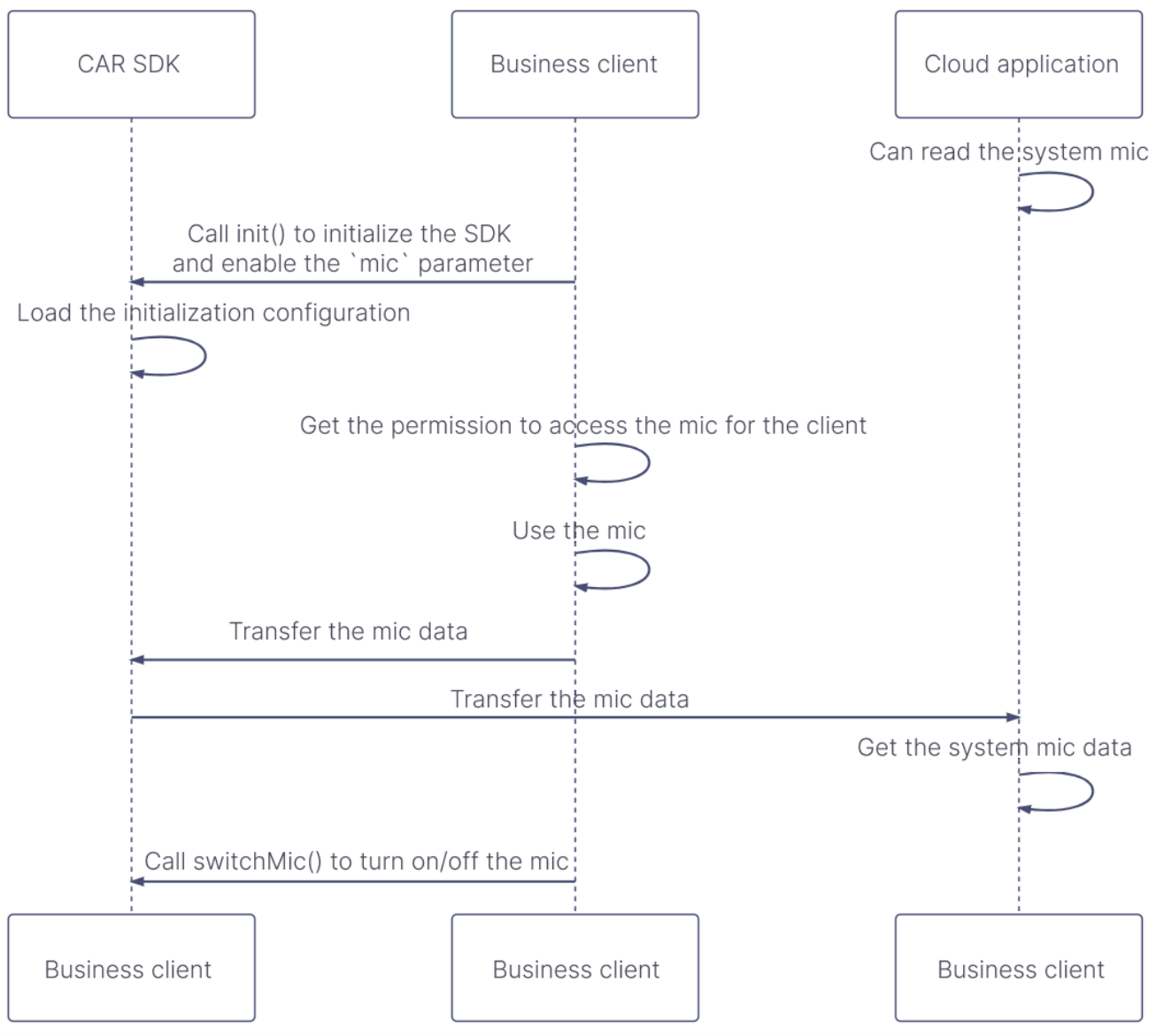
If your application is only displayed in a non-fullscreen window and cannot be set to borderless windowed fullscreen mode, you can go to the application configuration in the console to set the screen capture mode to **Capture a window**. You need to fill in the correct application window title and class name as described in Using Window Capturing Mode. If the window title is not specified during development, the window title after starting Demo.exe is usually “Demo”; if your application is developed with Unreal Engine, the class name is usually “UnrealWindow”.
Next steps:
1. The business client calls the TCGSDK.start() API to start the cloud application. After it connects to the application, the `onConnectSuccess()` callback is triggered. We recommend you set the resolution for the client after this callback.
2. The business client calls the TCGSDK.getPageSize() API or other methods to get the client resolution.
3. The business client notifies the cloud application to modify the resolution through a data channel or by using a custom method.
4. After the request to set the resolution is received in the cloud, **the cloud desktop resolution is then modified**. (This requires support by the application)
5. Since the display resolution of the application changes, to avoid problems such as incomplete display, partial screen unresponsiveness, and other issues caused by the application resolution exceeding the cloud desktop resolution, make sure the cloud desktop resolution is always greater than the application resolution by using one of the following methods:
Method 1: In the console, go to **Project settings** > **Advanced settings** and change the default cloud desktop resolution to a sufficiently large size, such as 2500 x 2500.
Method 2: Synchronously call the TCGSDK.setRemoteDesktopResolution() API to modify the cloud desktop resolution. The business client calls the `onRemoteScreenResolutionChange()` API to check whether the resolution settings have taken effect.
Note:
The methods above are relatively complicated, and improper configuration may cause issues such as incomplete screen display, partial screen unresponsiveness, and abnormal client screen resolution. Therefore, we recommend you use the recommended method of setting the application to borderless windowed mode and enabling adaptive resolution.

Enabling Mic

Last updated：2024-01-26 11:54:09

This document describes how to enable the mic. If your business needs to play back audio through a mic, you can use the SDK to enable the mic feature.

Sequence diagram



Directions

1. The cloud application can capture the audio of the system mic.
2. The business client calls the TCGSDK.init() API and sets the `mic` parameter to `true` to complete initialization.
3. The business client must be granted the permission to access the mic.

Note:

Taking a browser as an example, the system first needs to enable the mic permission of the browser. The browser then needs to enable the mic permission of the accessed page, so that the business client can enable the mic permission.

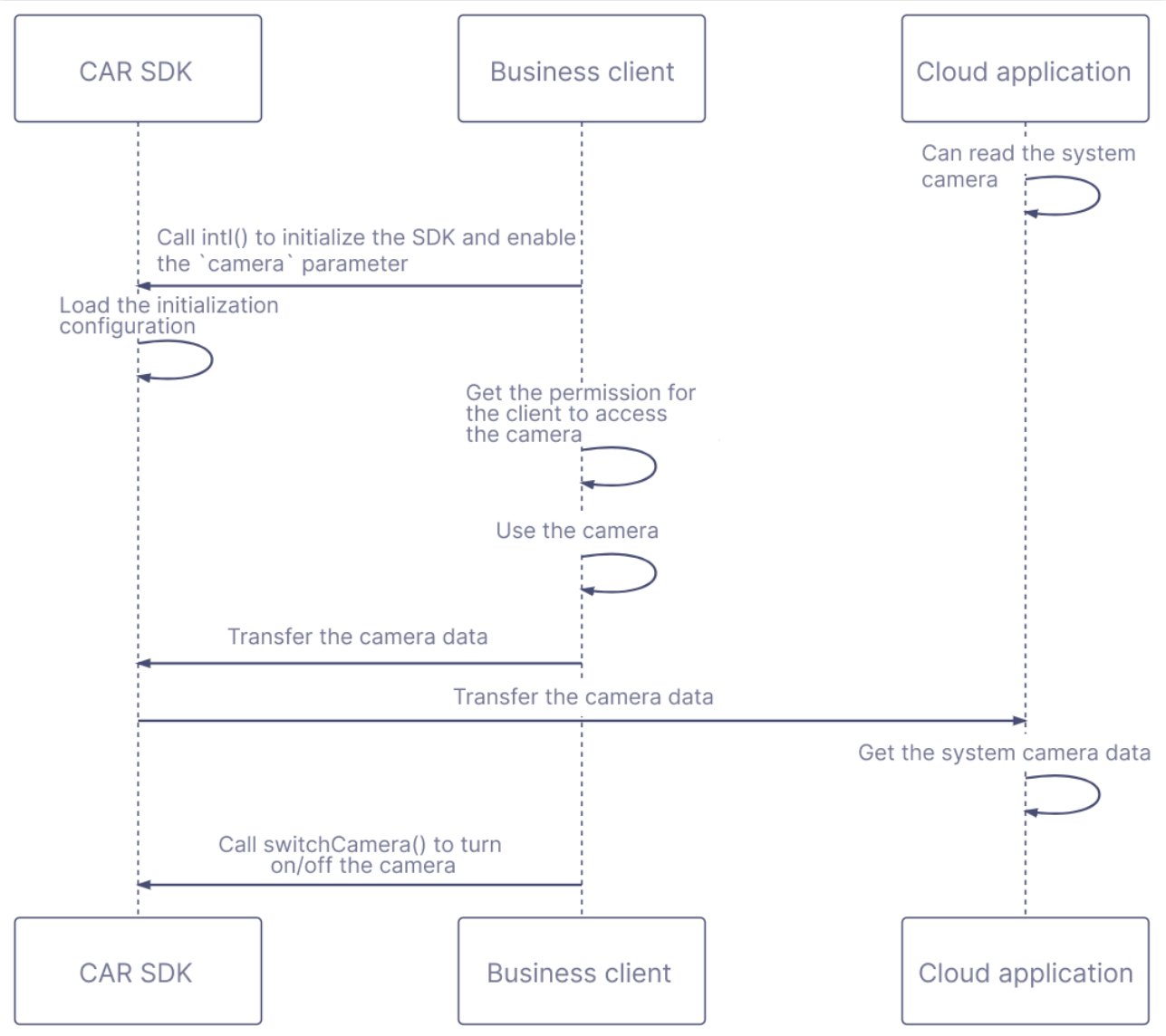
4. When the business client uses the mic, the data will be transferred to the cloud application through the SDK.
5. If the business needs to turn on/off the mic during connection, the business client can call TCGSDK.switchMic() to do this.

Enabling Camera

Last updated：2024-01-26 11:54:09

This document describes how to enable the camera. If your business needs to play back video from a camera, you can use the SDK to enable the camera feature.

Sequence diagram



Directions

1. The cloud application can capture the video image of the system camera.
2. The business client calls the TCGSDK.init() API and sets the `camera` parameter to `true` to complete initialization.
3. The business client must be granted the permission to access the camera.

Note:

Taking a browser as an example, the system first needs to enable the camera permission of the browser. The browser then needs to enable the camera permission of the accessed page, so that the business client can enable the camera permission.

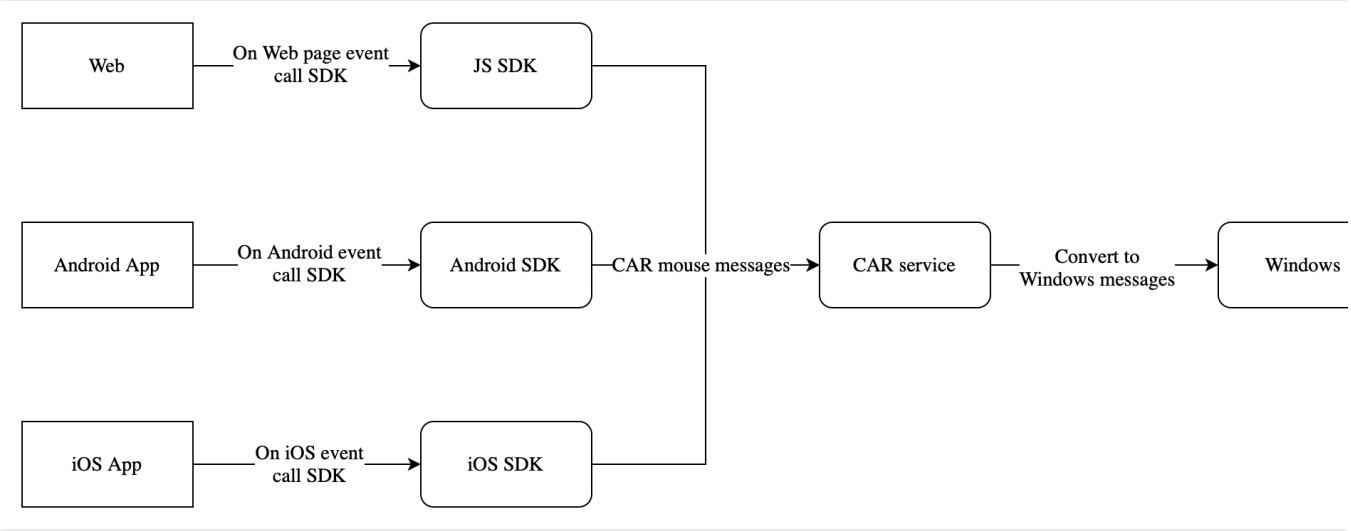
4. When the business client uses the camera, the data will be transferred to the cloud application through the SDK.
5. If the camera needs to be turned on/off during the connection, the business client can call TCGSDK.switchCamera() to do this.

Mouse/Keyboard/Touch Interactive Processing

Last updated：2024-01-26 11:54:09

This section mainly introduces the methods for handling interactions with mouse, keyboard, and touch screen.

Mouse



The above figure shows the conversion flow of mouse messages in CAR interaction. From the above, the **Application** is a self-service application uploaded by the business side in the console, which runs on the Windows system and responds to mouse window messages from Windows. When the business side accesses the SDK to send mouse messages, it needs to convert the messages from the platform where the SDK is running into mouse messages to operate the application.

In general, the situations that each platform deals with are as follows:

On Android and iOS platforms, the touch messages from the system are converted to CAR mouse messages.

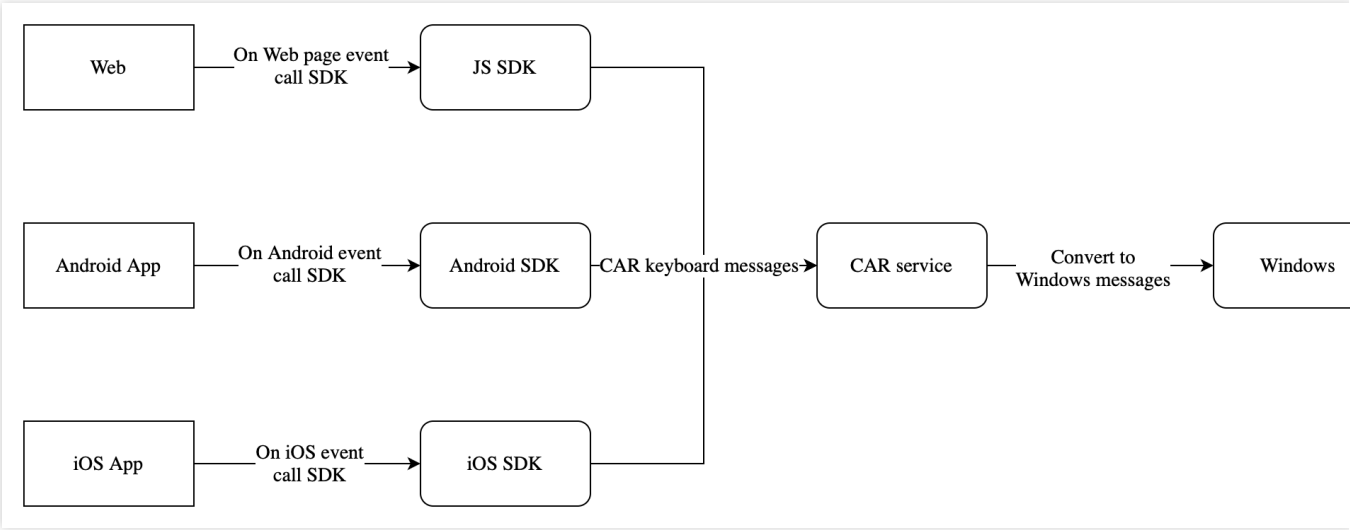
On Web pages, depending on the system on which the page is running, mouse messages are converted to CAR messages when the Web page is running in a Windows/macOS browser, and touch messages are converted to CAR messages when the Web page is running in an Android/iOS browser.

Among them, the conversion of touch messages to CAR mouse messages usually involves converting a single finger click to a CAR mouse click, a single finger swipe to a CAR mouse move, and so on.

Therefore, the business side needs to be adapted according to the SDK used and the platform running. For more details, please refer to the following:

Web page reference document: onTouchEvent, mouseMove, sendMouseEvent
Android App reference document: Android peripheral interaction processing
iOS App reference document: iOS peripheral interaction processing

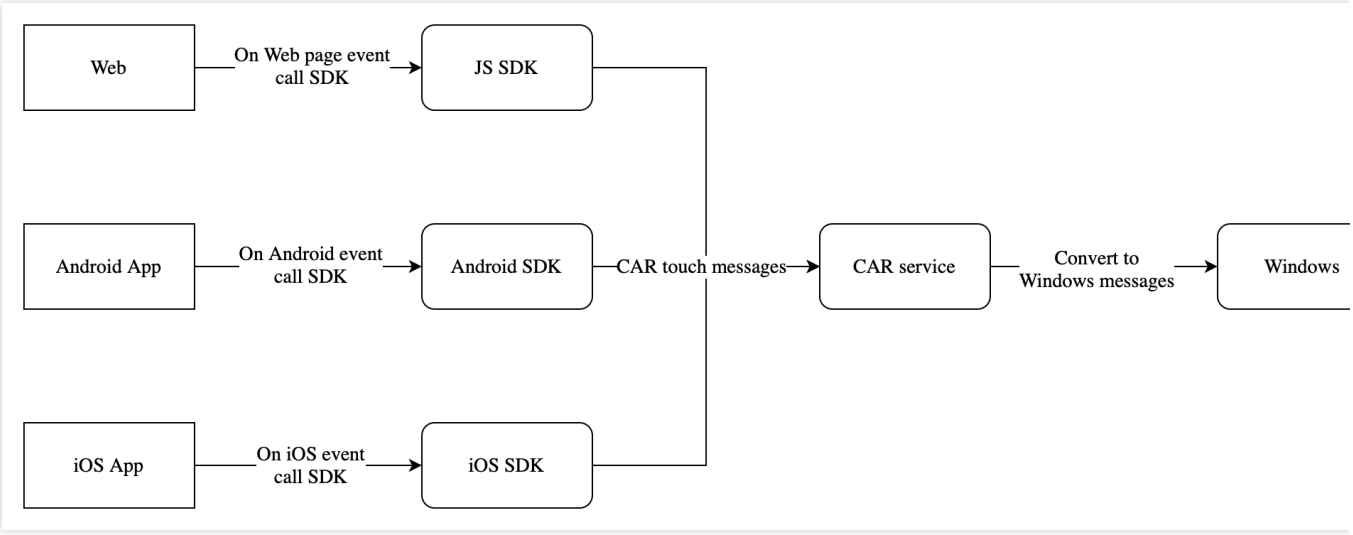
Keyboard



The above figure shows the conversion flow of keyboard messages in CAR interaction. From the above, the **Application** is a self-service application uploaded by the business side in the console, which runs on a Windows system and responds to keyboard window messages from Windows. When the business side accesses the SDK to send keyboard messages, it needs to convert the messages from the platform where the SDK is running into keyboard messages to operate the application.
In general, the situations that each platform deals with are as follows:
On Android and iOS platforms, the system's touch messages or soft keyboard messages are converted to CAR keyboard messages.
On Web pages, depending on the system on which the page is running, keyboard messages are converted to CAR keyboard messages when the Web page is running in a Windows/macOS browser, and touch messages or soft keyboard messages are converted to CAR keyboard messages when the Web page is running in an Android/iOS browser.
Among them, the conversion of touch messages or soft keyboard messages to CAR messages usually involves converting touch steering wheel operations to WASD messages of the CAR keyboard, converting soft keyboard messages directly to CAR keystroke messages, and so on.
Therefore, the business side needs to be adapted according to the SDK used and the platform running. For more details, please refer to the following:

Web page reference document: sendKeyboardEvent, Joystick
Android App reference document: Android peripheral interaction processing
iOS App reference document: iOS peripheral interaction processing

Touch



The above figure shows the conversion flow of touch messages in the CAR interaction. From the above, the **Application** is a self-service application uploaded by the business side in the console, which runs on the Windows system and responds to touch window messages from Windows. When the business side accesses the SDK to send touch messages, it needs to convert the messages from the platform where the SDK is running into touch messages to operate the application.
In general, the situations that each platform deals with are as follows:
On Android and iOS platforms, the touch messages from the system are converted to CAR touch messages.
Converting touch messages to CAR touch messages when the web page is running in an Android/iOS browser only.
It is recommended that the business side can use the SDK to convert touch messages from their platform to CAR touch messages only if the application supports Windows touch window message processing and the application tests well on a physical Windows touch screen device.

Note：

It is not recommended to use the mouse-to-touch of UE or Unity engine to test whether the application software handles the touch messages normaly, because the touch behavior will be abnormal on the touch screen device. Therefore, the business side needs to be adapted according to the SDK used and the platform running. For more details, please refer to the following:
Web page reference document: init.clientInteractMode, setClientInteractMode

Android App reference document: Android peripheral interaction processing

iOS App reference document: iOS peripheral interaction processing

Integration Demo

Last updated：2024-01-26 11:54:09

CAR provides demos for JavaScript, Android, and iOS, as well as a backend demo. This document describes how to quickly configure a demo to run a simple cloud application.

Step 1: **Deploy the b**ackend demo

Deploy the backend demo locally or on any server as instructed here. The demo is lightweight and has no special requirements for server configuration.
The CAR backend demo demonstrates various features, including application startup and exits as well as user queue. For more information on features and integration, see Technical Integration.
Deploying your own backend is a necessary step for using CAR services:
For the security of your assets and services, the `SecretId` and `SecretKey` of your Tencent Cloud account (which can be obtained on the Manage API Key page in the console) that are needed in order to access the CAR TencentCloud API service must be processed on your backend service. In addition, management of user sessions and other features such as user queue must be implemented on your backend service. For more information, see Basic logic of the business backend and client.

Step 2: Deploy the client demo

Demo for JavaScript

The CAR SDK for JavaScript supports various implementation scenarios, including PC and mobile web pages. This document describes how to quickly set up, deploy, and run the demo for the web.
1. Download TCGSDK to C or D drive on the local PC or the `Downloads` folder on Mac and keep the path of the folder such as `c:/cloudgame-js-sdk` .
2. Download the demo HTML file.
3. Replace the following paths/parameters in the demo. For more information, see CAR Demo:

Paths/parameters that need to be replaced	Note
`src="path/to/tcg-sdk"`	TCGSDK path, for example: `src="./cloudgame-js-sdk/dist/tcg-sdk/index.js"`
`url = 'http://xxxx/StartProject'`	Replace `xxxx` with the URL of the business backend demo such as `url = 'http://192.168.0.1:3000/StartProject'`

`ProjectId: 'project-id'` `UserId: 'user-id'`	`ProjectId` : The ID of the project created in the console. `UserId` : The unique ID of the end user currently connected to CAR, which is customized by you and is not understood by CAR. It can be randomly generated with the timestamp and should be kept unchanged during user reconnection.

4. For mobile operations, CAR provides the Joystick plugin to map WASD/up, down, left, and right arrow keys.
5. We recommend that you integrate the user queue system to improve the user experience when the number of users requesting your application becomes greater than the maximum number of concurrencies. The queue logic involves the frontend and backend. For the frontend code, see Queue. The signature logic can be added based on your business needs. For the backend implementation, see Queue Feature.
6. **Test, debug, and launch your application:**
After correctly performing the operations in the console and successfully deploying your backend and client services, you can open the demo HTML file in Chrome (recommended). The rendered application image will be displayed after being loaded successfully, and you can then interact with the application with the mouse/WASD.
You can perform more custom development based on the demo. If you encounter any problems during the testing process, contact us and provide the RequestId.

Demo for Android

The demo for Android provides basic CAR capabilities, including simple samples, virtual keyboard, and usage of some common APIs. Your own cloud application can be operated based on the features of the demo. You can quickly configure the demo and run a cloud application as follows:
1. Download the demo project for Android here and import the project to the AndroidStudio tool.
2. Integrate the application as instructed here.

Demo for iOS

The demo for iOS provides samples of a virtual keyboard and some common APIs to help you quickly integrate the SDK and start the CAR service. You can implement more features based on the demo to meet your specific needs. You can quickly configure the demo and run a cloud application as follows:
1. Download the demo project for iOS here.
2. Integrate the application as instructed here.