Cloud Phone Host

Best Practices

Issue	01
Date	2026-08-27

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: https://www.huawei.com

Email: support@huawei.com

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process.* For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Best Practices of Connecting to General-Purpose Cloud Phones.....1
1.1 Buying a Server for General-purpose Cloud Phones.....1
1.2 Connecting to a Cloud Phone to Obtain Its Screen.....8
1.3 Keeping Management Programs in the Cloud Phone Alive.....11
1.4 Deploying Applications on General-purpose Cloud Phones.....15
1.4.1 Querying Cloud Phones.....15
1.4.2 Installing Applications on a Cloud Phone.....16
1.4.3 Generating a TAR Package for an Application and Pushing It to an OBS Bucket.....17
1.4.4 Deploying Applications.....18
1.4.5 Updating the Version of an Application.....20
2 Best Practices of Cloud Phone Application Sharing.....22
2.1 Overview of Application Sharing.....22
2.2 Managing Application Packages in the Shared Space.....24
2.2.1 Configuring the Shared Space.....24
2.2.2 Preparing the TAR Package of an Application to Be Shared.....24
2.2.2.1 Querying Cloud Phones.....24
2.2.2.2 Installing Applications on a Cloud Phone.....25
2.2.2.3 Generating a TAR Package for an Application and Pushing It to an OBS Bucket.....27
2.2.3 Pushing the TAR Package of an Application to the Shared Space.....28
2.2.4 Deleting an Application from the Shared Space.....29
2.3 Application Scenarios of Shared Applications.....29
2.3.1 Application Installation on Cloud Phones.....29
2.3.1.1 Installing an Application.....29
2.3.1.2 Updating an Application.....30
2.3.1.3 Uninstalling an Application.....31
2.3.1.4 Uninstalling Applications in a Batch.....31
2.3.2 Pre-installation of Applications on Cloud Phones.....31
2.3.2.1 Installing Pre-installed Applications.....32
2.3.2.2 Updating Pre-installed Applications.....32
2.3.2.3 Uninstalling Pre-installed Applications.....33
2.3.3 File Deployment on Cloud Phones.....33
2.3.3.1 Presetting Files.....33
2.3.3.2 Deploying Files.....35

2.3.3.3 Updating Files 35
2.3.3.4 Removing Preset Files 36
2.3.3.5 Deploying Files Using Multiple Configuration File Packages 36
2.4 appctrl Commands 37
2.4.1 Running the appctrl Commands 37
2.4.2 Running appctrl install to Install Applications 37
2.4.3 Running appctrl start to Install and Start an Application 38
2.4.4 Running appctrl uninstall to Uninstall an Application 38
2.4.5 Running appctrl clear to Uninstall Applications in a Batch 38
2.4.6 Running appctrl applyConfig to Deploy Files 39

3 Installing an Application on Cloud Phones in a Batch 40

4 Simulating a Cloud Phone Location 45

5 Using the Cloud Phone Camera 48

6 Allowing a Cloud Phone Server to Access a Public Network Outside the Chinese Mainland 51

7 Delegating CPH to Operate OBS Buckets 57

8 Changing the AOSP Version of a Cloud Phone 61

1 Best Practices of Connecting to General-Purpose Cloud Phones

1.1 Buying a Server for General-purpose Cloud Phones

Procedure

1. Log in to the **CPH console** and go to the **Buy Cloud Phone Server** page.
2. Configure basic settings.

Table 1-1 Basic configuration parameters

Parameter	Description	Example Value
Billing Mode	CPH is billed only yearly/monthly.	Yearly/Monthly
Region	Cloud phone servers in different regions cannot communicate with each other over a private network. For lower latency and quicker access, select the nearest region. After a server is purchased, its region cannot be changed.	CN-Hong Kong

Parameter	Description	Example Value
AZ	An AZ is a part of a region and has its own independent power supplies and networks. AZs can communicate with each other over an internal network and are physically isolated. • If you require high availability, buy servers in different AZs. • If you require low network latency, buy servers in the same AZ.	AZ1
Server Type	Select a server type based on service requirements. For details, see **Servers for General-Purpose Cloud Phones**	Cloud phone server physical.rx1.xlarge
Instance Specifications	Select the required cloud phone specifications.	rc1.se
Phone Image	Only the Android is supported. **NOTE** To view private images, you must have the **ims:images:list** permissions.	AOSP7.1.1
Quantity	• A maximum of 10 servers can be purchased at a time. • The required duration ranges from 1 month to 3 years.	**Quantity**: **1** Required duration: **6 months**

3. Click **Next: Configure Network** to configure the network for the cloud phone server.

   You are advised to use the custom network described in **Table 1-2**.

Table 1-2 Configuring a custom network

Parameter	Description	Example Value
Network	Select an available VPC and subnet from the drop-down list and specify the method for assigning a private IP address. Ensure that you have the **VPC ReadOnlyAccess** or **CPH AgencyDependencyAccess** permissions at least. This VPC, including its subnets and security groups will be used to isolate the cloud phone server from the internet. You can also create a VPC. • **Automatically-assigned IPv6 address**: This parameter is available only for cloud phone servers with specific flavors and deployed in a VPC with IPv6 enabled in given regions. For details about how to enable IPv6 on a subnet, see **IPv4/IPv6 Dual-Stack Network**. By default, the system assigns IPv4 addresses. If you select **Automatically-assigned IPv6 address**, the system assigns both an IPv4 address and an IPv6 address. • **Do not configure**/Select the required shared bandwidth: In the same VPC, the cloud phone server can use the IPv6 address to access dual-stack servers. To access the Internet, select a shared bandwidth from the drop-down list and add the IPv6 address to the shared bandwidth. Then, the cloud phone server can access the IPv6 network on the Internet through the IPv6 address.	None

Parameter	Description	Example Value
	NOTE • If you want to use IPv6 addresses, select **Automatically-assigned IPv6 address** here. This option cannot be modified after your server is purchased. If you select **IPv6 not required** and want to use IPv6 addresses later, you can only buy a new server. • Due to VPC restrictions, IPv4/IPv6 dual stack is not supported in the CN East-Shanghai 2 region. • IPv6 addresses cannot be added to a dedicated bandwidth. • By default, a maximum of 20 IPv6 addresses can be added to a shared bandwidth. A dual-stack cloud phone server has the same number of IPv6 addresses and virtual IP addresses. If you want to purchase a cloud phone server with multiple virtual IP addresses, such as e0v100, apply for a higher shared bandwidth quota in advance. • RX1 cloud phone servers do not support IPv6 addresses.	

Parameter	Description	Example Value
Agency	Authorize CPH to create a **cph_admin_trust** agency that can assign the **CPH AgencyDependencyAccess** permissions. For more information, see **Permissions Management**. CPH will use the agency to perform the following operations: • Create an elastic NIC, EIP, and virtual IP address for a cloud phone. • Apply the default security group for the cloud phone server. The default security group defines the port range allowed to access the server. The port range will be mapped to that of each cloud phone. Then the cloud phones can access applications through the mapped ports. **NOTE** By default, if an ECS and a cloud phone are in the same VPC, the ECS cannot access the cloud phone through ports 1 to 9999. If you want to allow such access, add a security group rule with a higher priority by following instructions provided in **What Are the Security Group Authorization Rules for Cloud Phones Using Custom Networks?**	None
EIP	• **Auto assign**: Buy a new EIP for the server. • **Using existing**: An existing EIP will be assigned to the server.	Auto assign
EIP Type	• Static BGP offers routing control and protects against route flapping, but cannot choose an optimal path in real time when a network connection fails. • Dynamic BGP enables automatic failovers and chooses the optimal path when a network connection fails.	Dynamic BGP

Parameter	Description	Example Value
Billed By	The following options are available only when a new EIP is purchased: • Traffic: You will be charged based on the total traffic your applications generate. • Shared bandwidth: You will be charged by the bandwidth shared by multiple EIPs.	Shared bandwidth
Bandwidth Size	Value range: 1 Mbit/s to 300 Mbit/s	300 Mbit/s
Bandwidth Name	If you set **Bandwidth** to **Shared bandwidth**, select an existing shared bandwidth name from the drop-down list.	bandwidth-001

4. Click **Next: Configure Advanced Settings**.

Table 1-3 Parameters for advanced settings

Parameter	Description	Example Value
Name	Specifies a unique name for the server and its cloud phones. Naming rule: The system automatically adds a hyphen followed by a one-digit incremental number to the end of each server name. For the names of the cloud phones that are virtualized from the server, the system automatically adds a 5-digit number suffix in ascending order. For example, if you purchased a server that can virtualize 60 cloud phones and entered **CPH** for **Name**, the server name is **CPH-1**, and the cloud phone names vary from **CPH-1-00001** to **CPH-1-00060**.	CPH

Parameter	Description	Example Value
Key Pair	A key pair is used for remote login authentication. • If you have created a key pair and stored the private key file (in .pem format) locally, you can select it from the drop-down list. • If no key pair is available, click **Create Key Pair** to create one. Then go back to the **Configure Advanced Settings** page, refresh the drop-down list, and select the created key pair. The private key file is used for identity authentication during remote login. For security purposes, the private key file (in .pem format) can be downloaded only once. Keep it secure. For more information about key pairs, see **(Recommended) Creating a Key Pair on the Management Console**. **NOTE** • Ensure that your account has the **ecs:serverKeypairs:list** permissions to query key pairs. • If you need to create a key pair, ensure that your account has the **ecs:serverKeypairs:create** permissions.	KeyPair-test

Parameter	Description	Example Value
Application Port	Enable this parameter when your cloud phones need to provide services for external systems. • **Application name**: The value can contain only letters and cannot be **ADB** (case-insensitive). • **Port number**: Ports from **0** to **65535** are supported. • **Internet access** – If this option is selected, the cloud phone application port can be accessed over the Internet without authentication. The cloud phone port and the server port are accessible from the Internet. – If this option is not selected, cloud phones can be accessed only over a private network. **CAUTION** • Ensure that security control has been performed before you select **Internet access**. • CPH does not perform security check for ports you configured to be accessible from the Internet.	key 10001 Do not select it.

5. Click **Next: Confirm** to check the configuration.
   – If the configuration is correct, click **Buy Now**.
   – To modify the configuration, click **Previous**.
6. Complete the payment as prompted.
   After the payment, it takes the system about 20 to 30 minutes to automatically create cloud phones.
   The cloud phones are available when their statuses change to **Running**.

1.2 Connecting to a Cloud Phone to Obtain Its Screen

a cross-platform UI automation compiler, to obtain the cloud phone's screen.

Prerequisites

- You have purchased a cloud phone server and connected to a cloud phone using ADB. For details, see **Purchasing a Cloud Phone Server**.
- Airtest has been installed on your local PC.

NOTE

Airtest is available at the **Airtest official website**. Log in to the website, download the required version, and install it.

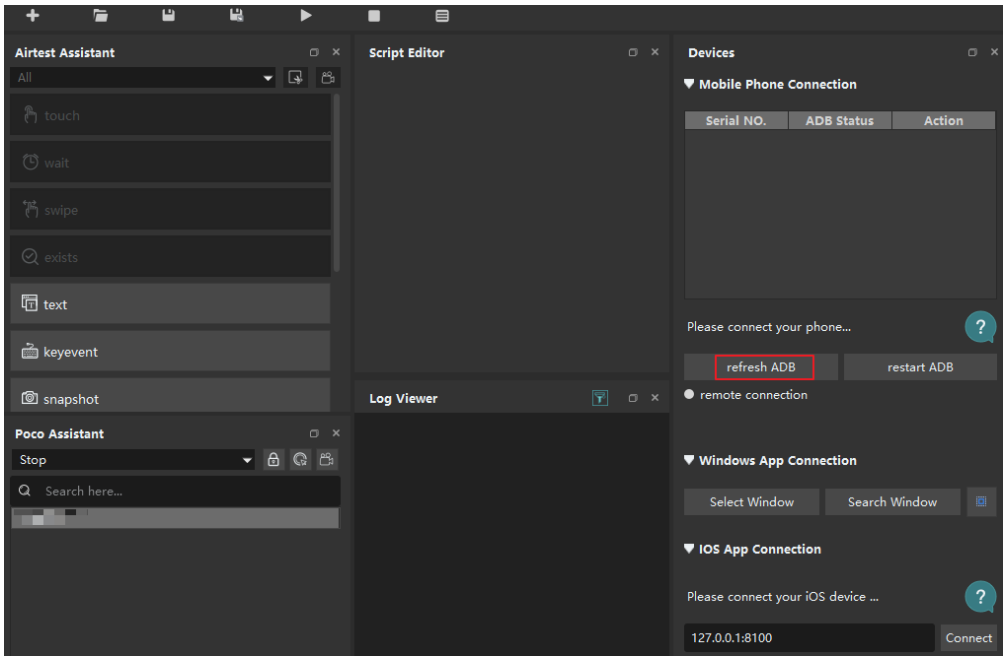
- The command-line interface (CLI) for the ADB connection has been closed, and an SSH tunnel has been successfully established.

Procedure

1. On the Airtest homepage, click **refresh ADB**.

   Connected mobile phones are displayed.

   Figure 1-1 Airtest homepage

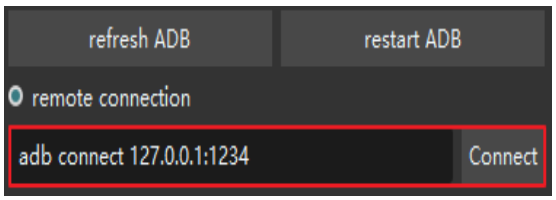


2. If the cloud phone to which you want to connect is not displayed, select **remote connection** and enter the ADB command for connecting to the target cloud phone, as shown in **Figure 1-2**.

   adb connect 127.0.0.1:*1234*

   1234 is the local idle port used for establishing the SSH tunnel.

   Figure 1-2 Establishing a remote connection to a cloud phone



   Click **Connect** on the right. The cloud phone to be connected will be displayed in the **Mobile Phone Connection** list.

CAUTION

Ensure that the CLI for the ADB connection has been closed. Otherwise, the connection will fail in this step. Ensure that the SSH tunnel has been successfully established. Otherwise, **ADB Status** will be **offline** and the cloud phone screen cannot be obtained even if the cloud phone has been identified.

3. In the list of identified mobile phones, click **connect** on the right of the target cloud phone to obtain its screen.

Figure 1-3 Mobile Phone Connection

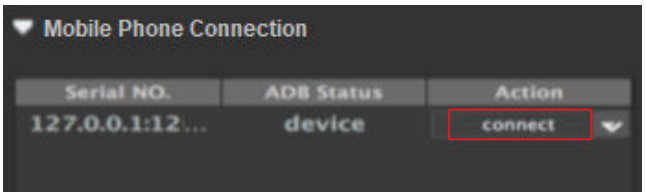
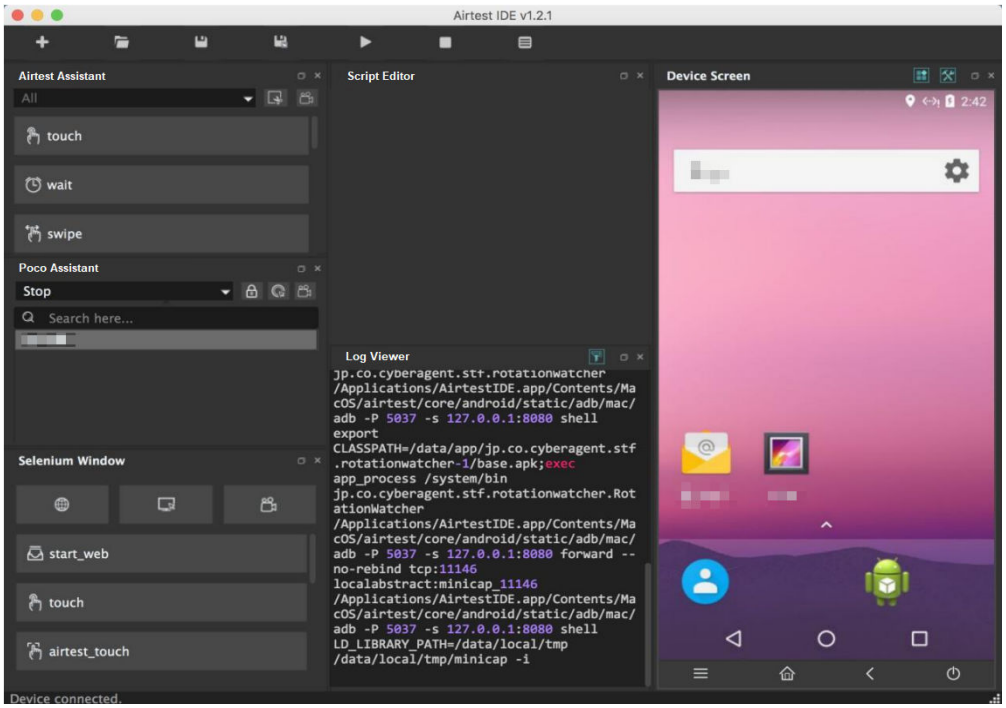
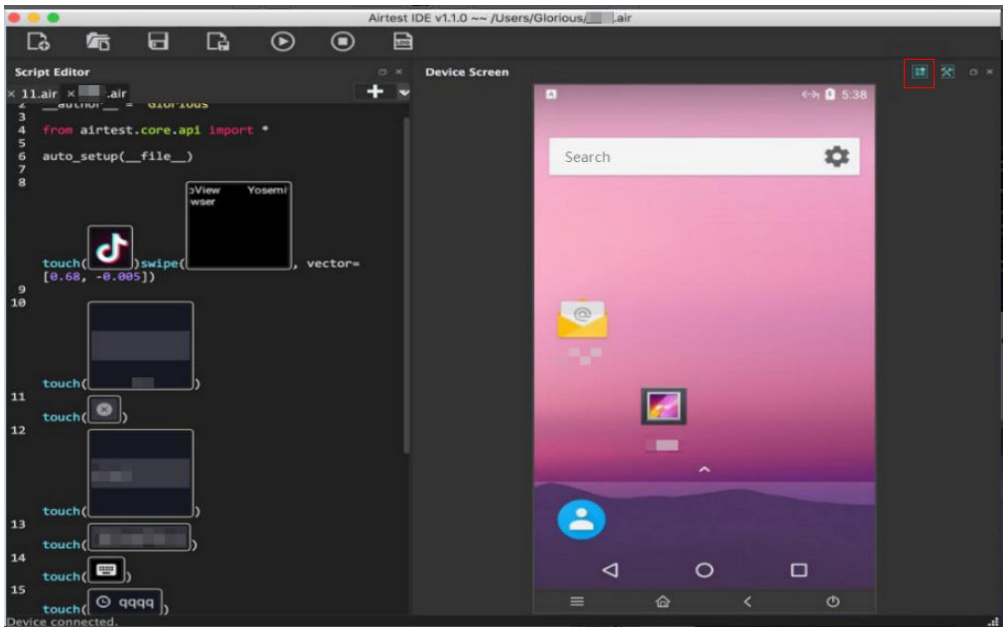


Figure 1-4 Device Screen



4. If you have connected to multiple cloud phones through ADB, click the switch icon in the upper right corner to switch between screens.

Figure 1-5 Switching between cloud phone screens



1.3 Keeping Management Programs in the Cloud Phone Alive

Management Programs

The management programs in a cloud phone are classified into the following types:

Android APK Service programs: Android services running in the background of the cloud phone without a UI

Android JNI Native programs: executable binary program developed using Android Java Native Interface (JNI)

Methods for Keeping Management Programs Alive

The **extend_custom.sh** hook script needs to be built into the cloud phone to keep its management problems alive. For details, see **Startup Script**.

When the cloud phone changes to the **boot_complete** state, it checks whether the **extend_custom.sh** script exists in the **/data/local/tmp** directory. If it does, the cloud phone executes the script. You can use this script to operate and move your own files, and start and manage your own programs. The script execution timeout interval is 10s.

- **Keeping Android APK Service programs alive**

Due to system mechanism restrictions, management programs of the Android APK Service type are in the stopped state if they are not started after the first installation. The broadcast indicating that the startup is complete uses **FLAG_EXCLUDE_STOPPED_PACKAGES**. As a result, the stopped applications

cannot receive the broadcast. The installation and initial startup of the Android APK Service management programs depend on the built-in **extend_custom.sh** script in the **/data/local/tmp** directory of the cloud phone. After the cloud phone is restarted, the programs can be started by receiving the startup completion broadcast.

You can configure the **AndroidManifest.xml** file to receive the startup broadcast and start the Android APK Service programs in the corresponding broadcast processing.

```
<!-- example code-->

<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<!--Applicable to 8.0 and later versions-->
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>

<receiver android:name=".DemoServiceReceiver">
    <intent-filter android:priority="1000">
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
 </receiver>
```

Process the startup broadcast and start an Android APK Service management program.

```
// example code
public class DemoServiceReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (!intent.getAction().equals("android.intent.action.BOOT_COMPLETED")) {
            return;
        }
        Intent serviceIntent = new Intent(context, DemoService.class);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            context.startForegroundService(serviceIntent);
        } else {
            context.startService(serviceIntent);
        }
    }
}
```

The keepalive of the Android APK Service management programs can return **START_STICKY** in the onStartCommand function of the corresponding Service type. In this way, the Android APK Service management programs can be restarted after being killed by the system.

```
// example code
public class DemoService extends Service {
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //other todo...

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) { //8.0 or later
            String channel_id = "MyTestService-id";
            String channel_name = "MyTestService-name";
            NotificationManager manager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
            NotificationChannel Channel = new NotificationChannel(channel_id, channel_name,
NotificationManager.IMPORTANCE_HIGH);
            if (manager != null) {
                manager.createNotificationChannel(Channel);
            }
            Notification notification = new
Notification.Builder(this).setChannelId(channel_id).setSmallIcon(R.mipmap.ic_launcher).build();
            startForeground(100, notification);
        }
        return START_STICKY;
```

```
    }
}
```

The following is an example of the installation and initial startup of Android APK Service management programs that depend on the **extend_custom.sh** script. (The default APK file is also stored in the **/data/local/tmp** directory.)

```
#  example code

# !/system/bin/sh

# APK name
ApkName=service.apk
# APK package name
PackageName=com.huawei.myapplication
# Service
ServiceName=com.huawei.myapplication/.MyService

InstallService() {
        count=`pm list packages | grep $PackageName |wc -l`
        if [ $count -le 0 ]; then
                echo "$ApkName is not installed, now install."
                ret=`pm install $ApkName`
                Succ="Success"
                if [ "$ret" == "$Succ" ]; then
                        echo "install succeed."
                else
                        echo "install failed."
                        exit 1
                fi
        else
                echo "$ApkName is already installed."
                echo "Service start by boot_complete broadcast."
                exit 1
        fi
        return 0
}

StartService() {
    ret=`am startservice -n $ServiceName`
    Err="Error: Not found; no service started."
        contain=$(echo $ret | grep "${Err}")
        if [[ "$contain" != "" ]]; then
                echo "Service start failed."
                exit 1
        else
                echo "Service start succeed."
        fi
        return 0
}

main() {
        # Installation
        InstallService

        echo "To start when first installed."
        # Start
        StartService
}
```

- **Keepalive scheme of management programs of the Android JNI Native type**

The Android JNI Native management programs must be stored in the **/system/bin** directory of the cloud phone and granted the execute permissions. The binary .so libraries on which the Android JNI Native management programs depend must be stored in the **system/lib** and **system/lib64** directories.

The Android JNI Native management programs and binary .so libraries can be pushed to the **data** directory of cloud phones by shared storage or application. You can use the **extend_custom.sh** script to place your files in the corresponding directory. The following methods are available to meet different keepalive requirements:

- System-level keepalive

System-level keepalive can edit and move the **init_custom.rc** file to the **/data/local/** directory. You need to restart the cloud phone, then the system will scan the **init_custom.rc** file. After the system is in the **boot_completed** state, it starts NativeDemo (the name of the Android JNI Native management program in the example). If the NativeDemo process exits abnormally or is killed, the system restarts it again.

```
on property:sys.boot_completed=1
    start NativeDemo

service NativeDemo /system/bin/NativeDemo
    user root
    group root
    disabled
writepid /dev/cpuset/system-background/tasks
```

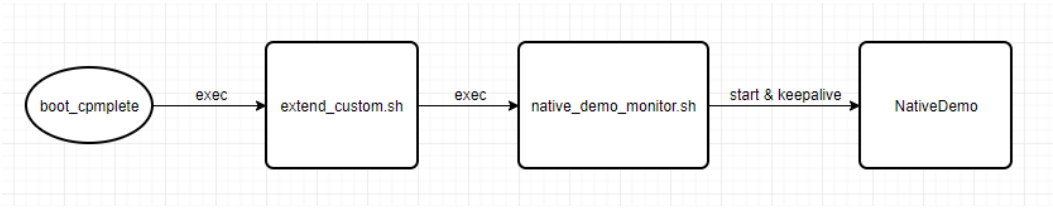
The system-level keepalive has the following advantages and disadvantage:

Advantages: System-level automatic startup and keepalive ensure high reliability and high real-time performance.

Disadvantage: The setting takes effect only after the cloud phone is restarted.

- User-level keepalive

If you need both automatic startup and management program keepalive, and you do not want to restart the cloud phones for the settings to take effect, you can run the **native_demo_monitor.sh** script in **extend_custom.sh** to check whether the management program is running. The **native_demo_monitor.sh** script implements the startup and keepalive of the NativeDemo management program.



Example of the **extend_custom.sh** hook script

```
#  example code

# !/system/bin/sh

[[ -f /data/local/tmp/native_demo_monitor.sh ]] && sh /data/local/tmp/native_demo_monitor.sh &
```

Example of the **native_demo_monitor.sh** script

```
#  example code
# !/system/bin/sh

file_dir="/data/demo"

CopyFile() {
```



```
	cp -rf $file_dir/NativeDemo /system/bin/
	chmod 755 /system/bin/NativeDemo

	cp $file_dir/lib*.so /system/lib64/
}

CheckIfCopyFile() {
	if [ -s $file_dir ]; then
		echo "file exist"
		CopyFile
	else
		echo "file not exist"
	fi
}

Start() {
	nohup NativeDemo &
}

KeepAlive() {
	while do
		echo "check NativeDemo proc"
		proc_count=`ps | grep NativeDemo | wc -l`
		if [ $proc_count -le 0 ]; then
			echo "start NativeDemo"
			Start
		else
			echo "NativeDemo already started"
		fi
		sleep 5
	done
}

main() {
	CheckIfCopyFile
	KeepAlive
}

main
```

The user-level keepalive has the following advantage and disadvantage:

Advantage: You do not need to restart your cloud phone for the settings to take effect.

Disadvantage: The real-time performance of the startup is not as good as that at the system level. The startup depends on the check interval.

1.4 Deploying Applications on General-purpose Cloud Phones

1.4.1 Querying Cloud Phones

Obtain the cloud phone list by referring to **Querying Cloud Phones** in the *Cloud Phone Host API Reference*.

Example API

```
GET https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones?
phone_name={phone_name}&server_id={server_id}&status={status}&offset={offset}&limit={limit}&type={type}
Header:
```

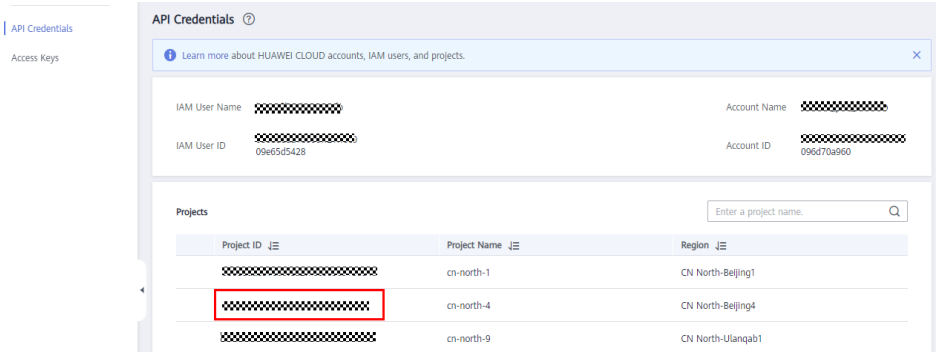


```
Content-Type: application/json
X-Auth-Token: ${token}
```

Parameter descriptions:

- **CPH Endpoint** indicates the CPH endpoint in each region in the endpoint list. For example, the CPH endpoint in the CN-Hong Kong region is **cph.ap-southeast-1.myhuaweicloud.com**.
- **project_id** indicates the project ID of the region where the gaming cloud phone server is deployed, for example, **083e9f825e80f50c2f96c0045edc70e8**. The project ID can be obtained by performing the following operations:
  a. Log in to the management console.
  b. Click the username in the upper right corner of the page, and choose **My Credentials** from the drop-down list.
  c. On the **API Credentials** page, obtain the project ID in the project list.

  Figure 1-6 Obtaining the project ID



- The part after the question mark (**?**) in the URL is optional.
- **$token** indicates the response of the API for **Obtaining a User Token Through Password Authentication**.

API Calling Example

```
GET https://cph.cn-north-4.myhuaweicloud.com/v1/083e9f825e80f50c2f96c0045edc70e8/cloud-phone/phones
Header:
Content-Type: application/json
X-Auth-Token: ${token}
```

NOTE

Replace **${token}** with the actual token.

1.4.2 Installing Applications on a Cloud Phone

Install applications on a cloud phone by referring to **Installing the APK** in the *Cloud Phone Host API Reference*.

Prerequisites

- The required Android Package (APK) has been stored in the Object Storage Service (OBS) bucket in the region where the cloud phone server is deployed. For details about how to upload the installation package, see **Scenario 2: Uploading and Downloading Files Through OBS Browser+**.

- The OBS bucket policies have been configured based on **Delegating CPH to Operate OBS Buckets**.

Example API

```
POST https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones/commands
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
   "command": "install",
   "content": "-t -r obs://{bucket_name}/{object_path}",
   "phone_ids": [
      "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
   ]
}
```

Parameter descriptions:

- Obtain the values of parameters such as **CPH Endpoint**, **project_id**, and **${token}** by referring to **Querying Cloud Phones**.
- **bucket_name** indicates the OBS bucket name. **object_path** indicates the path for storing the application installation package.
- **phone_ids** indicates the ID of the cloud phone on which the application is to be installed. (Obtain the cloud phone ID by referring to **Querying Cloud Phones**. You can enter multiple cloud phone IDs, and the application will be installed on all these cloud phones.)

API Calling Example

```
POST https://cph.cn-east-3.myhuaweicloud.com/v1/081ceeb7fb800f0c2f4cc004bb39c2f7/cloud-phone/phones/commands
Content-Type: application/json
X-Auth-Token: ${token}
{
   "command": "install",
   "content": "-t -r obs://yzw-apk-install/apk/com.hermes.bgame.apk",
   "phone_ids": [
      "bdc2f2e960164dd9a2765374afeea300"
   ]
}
```

- **yzw-apk-install** is the OBS bucket name.

  apk/com.hermes.bgame.apk is the path of the application installation package.

  obs://yzw-apk-install/apk/com.hermes.bgame.apk is the full path of the application installation package.
- Replace **${token}** with the actual token.

1.4.3 Generating a TAR Package for an Application and Pushing It to an OBS Bucket

Prerequisites

- The required application has been installed on the cloud phone.
- The OBS bucket policies have been configured based on **Delegating CPH to Operate OBS Buckets**.

Example API

```
POST https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones/batch-storage
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
    "storage_infos": [{
        "phone_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "include_files": [
            "/data/app/${package_name}-*",
            "/data/data/${package_name}",
            "/data/media/0/Android/data/${package_name}"
        ],
        "bucket_name": "${bucket_name}",
        "object_path": "apk/${package_name}_${version_name}.tar"
    }]
}
```

Parameter descriptions:

- Obtain the values of parameters such as **CPH Endpoint**, **project_id**, **${token}**, **bucket_name**, and **object_path** by referring to **Installing Applications on a Cloud Phone**.
- **phone_id** indicates the ID of the cloud phone on which the application is installed.
- The three elements in **include_files** must be set to the absolute paths on the cloud phone.
- If the installation package is an .xapk package, add **/data/media/obb/***${package_name}* to **include_files.**
- **object_path** indicates the path to which the .tar package is uploaded in the OBS bucket.

NOTICE

1. In **object_path**, **apk** indicates an existing folder in the OBS bucket, *${package_name}* indicates the package name of the current application, and *${version_name}* indicates the version of the current application. The version number can be customized.
2. Some shared applications download resource files online when they are started. You can start such applications before generating TAR packages to make them download resource files and patches. This way, shared applications do not need to download the files and patches again when they are installed and started on cloud phones.

1.4.4 Deploying Applications

Servers of Storage 2.0 (Recommended)

Push the .tar package to servers. That is, push **apk/${package_name}_${version_name}.tar** to the shared application of **${server_id1}** and **${server_id2}**.

- Example API

```
POST https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones/share-apps
Header:
```



```
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
     "package_name": "${package_name}"
   "bucket_name": "${bucket_name}",
   "object_path": "apk/${package_name}_${version_name}.tar",
   "server_ids": [
      "${server_id1}",
      "${server_id2}"
   ]
}
```

Parameter descriptions:

- Obtain the values of parameters such as **CPH Endpoint**, **project_id**, **${token}**, **bucket_name**, and **object_path** by referring to **Installing Applications on a Cloud Phone**.
- **package_name** indicates the package name of the application in Android, for example, **com.miniteck.miniworld**.
- **object_path** indicates the path to which the .tar package is uploaded.
- **package_name** and **version_name** indicate the package name and version number of the current application.

  NOTE

  apk is any existing folder. In **${package_name}_${version_name}.tar**, **package_name** and **version_name** need to be modified as required.
- **server_ids** indicates IDs of servers where the application is to be deployed. You can enter multiple server IDs. To obtain the server IDs, call the API for **Querying Cloud Phone Servers**.

- Example

  For details, see **Pushing a Shared Application** in the *Cloud Phone Host API Reference*.

Servers of Storage 1.0

Push the .tar package to the server. That is, push **apk/${package_name}_${version_name}.tar** to the shared storage of **${server_id1}** and **${server_id2}**.

- Example API

```
POST https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones/share-files
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
   "bucket_name": "${bucket_name}",
   "object_path": "apk/${package_name}_${version_name}.tar",
   "server_ids": [
      "${server_id1}",
      "${server_id2}"
   ]
}
```

Parameter descriptions:

- Obtain the values of parameters such as **CPH Endpoint**, **project_id**, **${token}**, **bucket_name**, and **object_path** by referring to **Installing Applications on a Cloud Phone**.
- **object_path** indicates the path to which the .tar package is uploaded.

  - **package_name** and **version_name** indicate the package name and version number of the current application.

    NOTE

    apk is any existing folder. In **${package_name}_${version_name}.tar**, **package_name** and **version_name** need to be modified as required.
  - **server_ids** indicates the ID list of servers where the application is deployed. You can enter multiple server IDs. To obtain the server IDs, you can call the API for **Querying Cloud Phone Servers**.
- Example

  For details, see **Pushing Shared Storage Files** in the *Cloud Phone Host API Reference*.
- Follow-up procedure

  Reset all cloud phones in batches by referring to **Resetting Cloud Phones** in the *Cloud Phone Host API Reference*.

1.4.5 Updating the Version of an Application

Servers of Storage 2.0 (Recommended)

Uninstall an application from your cloud phone.

Push the latest application.

Run the **appctrl start** command to start the application of the latest version.

Servers of Storage 1.0

To update the version of an application, delete the application of the earlier version from the server and deploy the application of the new version.

Deleting the Application of an Earlier Version

- Example API

```
POST https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones/share-files
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
   "file_paths": [
      "/data/app/${package_name}-1",
      "/data/app/${package_name}-2",
      "/data/data/${package_name}",
      "/data/media/0/Android/data/${package_name}"
   ],
   "server_ids": [
      "${server_id1}",
      "${server_id2}"
   ]
}
```

  Delete the following files from the shared storage on servers **${server_id1}** and **${server_id2}**:

  /data/app/${package_name}-1

  /data/app/${package_name}-2

/data/data/${package_name}

/data/media/0/Android/data/${package_name}

Parameter descriptions:

- Obtain the values of parameters such as **CPH Endpoint**, **project_id**, and **${token}** by referring to **Installing Applications on a Cloud Phone**.
- The content of **file_paths** is the same as that of **include_files** in **Generating a TAR Package for an Application and Pushing It to an OBS Bucket**. **package_name** indicates the package name of the current application.
- **server_ids** indicates IDs of servers where the application is deployed. You can enter multiple server IDs. To obtain the server IDs, call the API for **Querying Cloud Phone Servers**.

- Example

  For details, see **Deleting a Shared Storage File** in the *Cloud Phone Host API Reference*.

Deploying the Application of the New Version

See **Deploying Applications**.

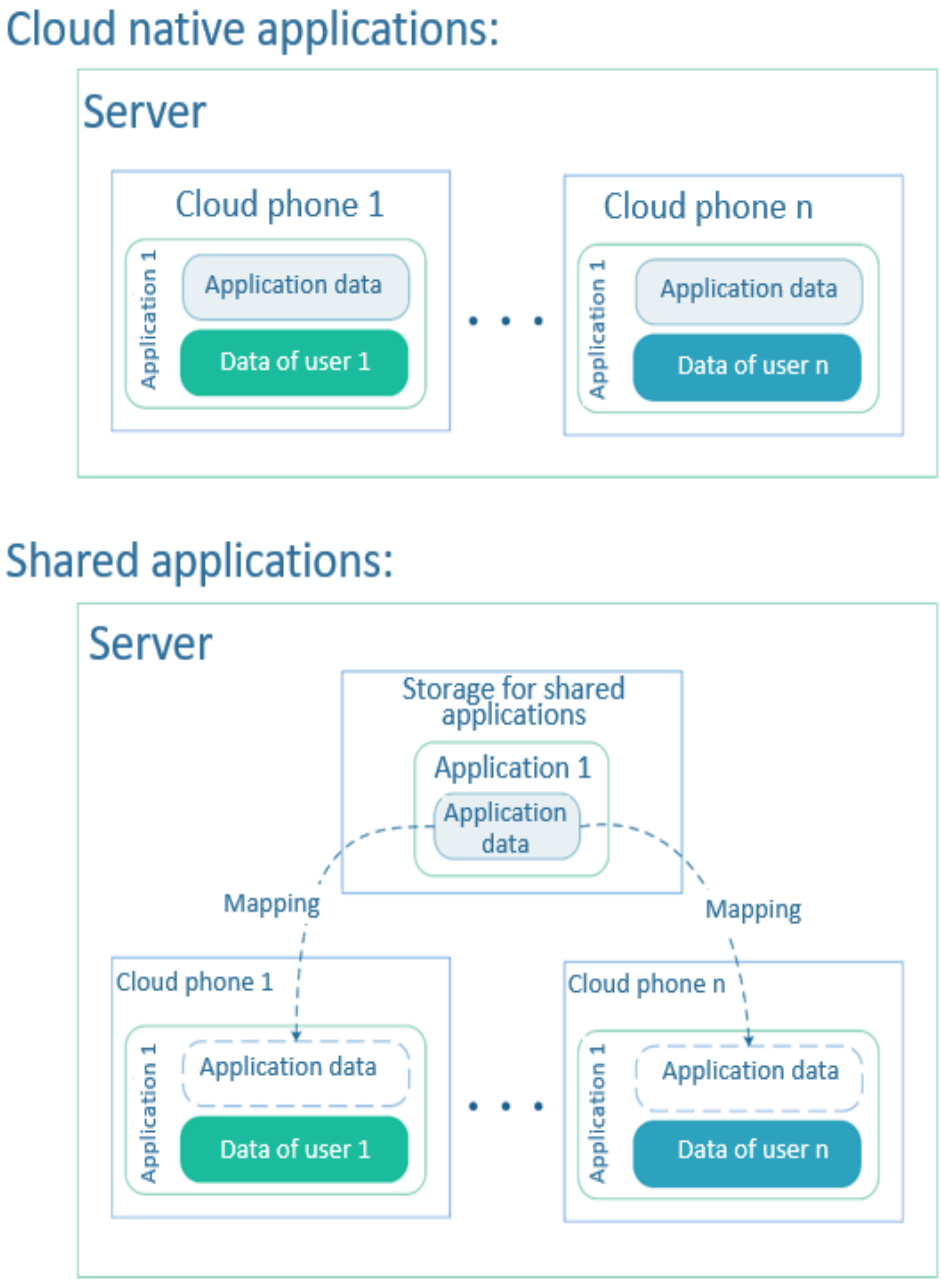
2 Best Practices of Cloud Phone Application Sharing

2.1 Overview of Application Sharing

What is application sharing?

A cloud phone server provides a shared application storage area (shared space for short). The user of the cloud phone server only needs to download or update an application once, and then package and upload the application data to the shared space of a specified cloud phone server. In this way, all cloud phones on the cloud phone server can share the application.

Figure 2-1 Application sharing



For cloud phone server users, application sharing prevents the same application data from being repeatedly downloaded, stored, and updated on multiple cloud phones, reducing storage and network costs. In addition, application sharing has the following advantages:

- Flexible and controllable deployment: Cloud phone applications can be deployed in batches on multiple cloud phones or sequentially on individual cloud phones.
- Convenient and ultimate experience: Updated applications can be shared, simplifying installation and eliminating the need for update operations. In this way, you can use applications immediately after accessing them.

Helpful Links

1. For details about how to install and update applications for cloud phones on a cloud phone server, see **Application Installation on Cloud Phones**.
2. For details about how to pre-install and update the same applications for all cloud phones on a cloud phone server, see **Pre-installation of Applications on Cloud Phones**.
3. Application sharing can be used to deploy non-application files in batches. For details, see **File Deployment on Cloud Phones**.

2.2 Managing Application Packages in the Shared Space

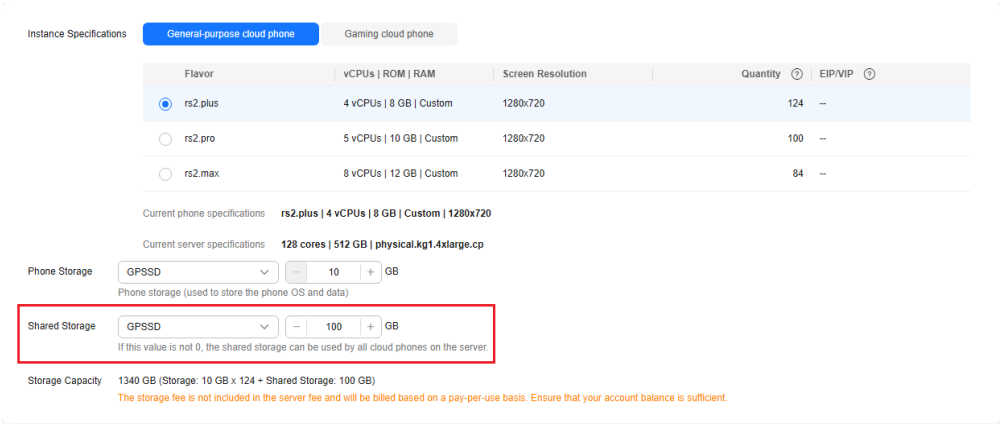
Before using application sharing on a cloud phone server, configure a shared space with an appropriate size for the server when you purchase it. Then, package and push the application files to the shared space. If an application is no longer used, delete it from the shared space.

2.2.1 Configuring the Shared Space

On the **Buy Cloud Phone Server** page, after you select a region, server type, and instance specifications, configure the shared space.

You are advised to increase the size of the required shared space by 30% to meet the requirements of subsequent application updates.

Figure 2-2 Configuring **Shared Space**



2.2.2 Preparing the TAR Package of an Application to Be Shared

2.2.2.1 Querying Cloud Phones

Obtain the cloud phone list by referring to **Querying Cloud Phones** in the *Cloud Phone Host API Reference*.

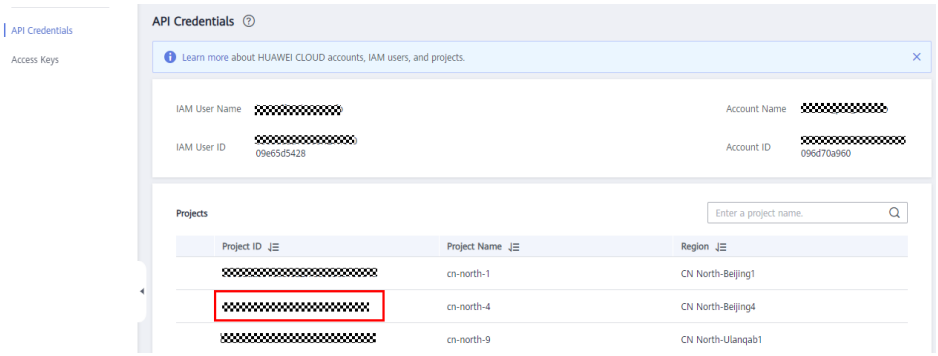
Example API

```
GET https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones?
phone_name={phone_name}&server_id={server_id}&status={status}&offset={offset}&limit={limit}&type={type}
Header:
Content-Type: application/json
X-Auth-Token: ${token}
```

Parameter descriptions:

- **CPH Endpoint** indicates the CPH endpoint in each region in the endpoint list. For example, the CPH endpoint in the CN-Hong Kong region is **cph.ap-southeast-1.myhuaweicloud.com**.
- **project_id** indicates the project ID of the region where the gaming cloud phone server is deployed, for example, **083e9f825e80f50c2f96c0045edc70e8**. The project ID can be obtained by performing the following operations:
  a. Log in to the management console.
  b. Click the username in the upper right corner of the page, and choose **My Credentials** from the drop-down list.
  c. On the **API Credentials** page, obtain the project ID in the project list.

  Figure 2-3 Obtaining the project ID



- The part after the question mark (**?**) in the URL is optional.
- **$token** indicates the response of the API for **Obtaining a User Token Through Password Authentication**.

API Calling Example

```
GET https://cph.cn-north-4.myhuaweicloud.com/v1/083e9f825e80f50c2f96c0045edc70e8/cloud-phone/phones
Header:
Content-Type: application/json
X-Auth-Token: ${token}
```

NOTE

Replace **${token}** with the actual token.

2.2.2.2 Installing Applications on a Cloud Phone

Install applications on a cloud phone by referring to **Installing the APK** in the *Cloud Phone Host API Reference*.

Prerequisites

- The required Android Package (APK) has been stored in the Object Storage Service (OBS) bucket in the region where the cloud phone server is deployed. For details about how to upload the installation package, see **Scenario 2: Uploading and Downloading Files Through OBS Browser+**.
- The OBS bucket policies have been configured based on **Delegating CPH to Operate OBS Buckets**.

Example API

```
POST https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones/commands
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
   "command": "install",
   "content": "-t -r obs://{bucket_name}/{object_path}",
   "phone_ids": [
      "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
   ]
}
```

Parameter descriptions:

- Obtain the values of parameters such as **CPH Endpoint**, **project_id**, and **${token}** by referring to **Querying Cloud Phones**.
- **bucket_name** indicates the OBS bucket name. **object_path** indicates the path for storing the application installation package.
- **phone_ids** indicates the ID of the cloud phone on which the application is to be installed. (Obtain the cloud phone ID by referring to **Querying Cloud Phones**. You can enter multiple cloud phone IDs, and the application will be installed on all these cloud phones.)

API Calling Example

```
POST https://cph.cn-east-3.myhuaweicloud.com/v1/081ceeb7fb800f0c2f4cc004bb39c2f7/cloud-phone/phones/commands
Content-Type: application/json
X-Auth-Token: ${token}
{
   "command": "install",
   "content": "-t -r obs://yzw-apk-install/apk/com.hermes.bgame.apk",
   "phone_ids": [
      "bdc2f2e960164dd9a2765374afeea300"
   ]
}
```

- **yzw-apk-install** is the OBS bucket name.

  apk/com.hermes.bgame.apk is the path of the application installation package.

  obs://yzw-apk-install/apk/com.hermes.bgame.apk is the full path of the application installation package.
- Replace **${token}** with the actual token.

2.2.2.3 Generating a TAR Package for an Application and Pushing It to an OBS Bucket

Prerequisites

- The required application has been installed on the cloud phone.
- The OBS bucket policies have been configured based on **Delegating CPH to Operate OBS Buckets**.

Example API

```
POST https://{CPH Endpoint}/v1/{project_id}/cloud-phone/phones/batch-storage
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
    "storage_infos": [{
        "phone_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "include_files": [
            "/data/app/${package_name}-*",
            "/data/data/${package_name}",
            "/data/media/0/Android/data/${package_name}"
        ],
        "bucket_name": "${bucket_name}",
        "object_path": "apk/${package_name}_${version_name}.tar"
    }]
}
```

Parameter descriptions:

- Obtain the values of parameters such as **CPH Endpoint**, **project_id**, **${token}**, **bucket_name**, and **object_path** by referring to **Installing Applications on a Cloud Phone**.
- **phone_id** indicates the ID of the cloud phone on which the application is installed.
- The three elements in **include_files** must be set to the absolute paths on the cloud phone.
- If the installation package is an .xapk package, add **/data/media/obb/*${package_name}*** to **include_files.**
- **object_path** indicates the path to which the .tar package is uploaded in the OBS bucket.

NOTICE

1. In **object_path**, **apk** indicates an existing folder in the OBS bucket, *${package_name}* indicates the package name of the current application, and *${version_name}* indicates the version of the current application. The version number can be customized.
2. Some shared applications download resource files online when they are started. You can start such applications before generating TAR packages to make them download resource files and patches. This way, shared applications do not need to download the files and patches again when they are installed and started on cloud phones.

2.2.3 Pushing the TAR Package of an Application to the Shared Space

Call the API to push the TAR package of an application in the OBS bucket to the shared space of the cloud phone server.

Example API

```
POST https://${CPH Endpoint}/v1/${projectId}/cloud-phone/phones/share-apps
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
   "package_name": "${package_name}",
   "bucket_name": "${bucket_name}",
   "object_path": "apk/${package_name}_${version_name}.tar",
   "pre_install_app": 0,
   "server_ids": [
      "${server_id1}",
      "${server_id2}"
   ]
}
```

Parameter descriptions:

- *${bucket_name}* indicates the OBS bucket name, and **object_path** indicates the path for storing the TAR package of the application in the OBS bucket.
- If **pre_install_app** is set to **1**, the application to be pushed is set as a pre-installed application. If **pre_install_app** is set to **0**, the application is not pre-installed. (For details about the pre-installed applications, see **Pre-installation of Applications on Cloud Phones**.) If an application is set as a pre-installed application, the application will be automatically installed on a cloud phone after the cloud phone is reset.
- **server_ids** indicates the ID list of servers to which the TAR package of the application is to be pushed. You can specify multiple server IDs at a time.
- For details about this API, see **Pushing a Shared Application**.

NOTICE

1. An application can be pushed for multiple times. The version pushed later is the latest version of the application. Multiple versions can coexist in the shared space.
2. **package_name** indicates the actual package name of the application and cannot be modified. If the package is downloaded from non-official channels, the package may be suffixed with the channel name. You can view the actual package name on the cloud phone where the application has been installed in **Querying Cloud Phones**.

   Example:

   Package name downloaded from the official website: **com.huawei.***xxxx*

   Package name downloaded from other channels: **com.huawei.***xxxx***.huawei**

2.2.4 Deleting an Application from the Shared Space

When all cloud phones on a cloud phone server no longer use a shared application, you can delete the shared application from the shared space to release storage space.

Constraints and Limitations

The shared application has been uninstalled from all cloud phones on the cloud phone server.

Example API

```
DELETE "https://${CPH Endpoint}/v1/${projectId}/cloud-phone/phones/share-apps"
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
   "package_name": "com.huawei.xxxx",
   "server_ids": ["1678567b8bab40f937112xxxxx","1234567b8bab40ffb711xxxxx"]
}
```

package_name indicates the actual package name of the application. **server_ids** indicates the server ID list. You can specify multiple server IDs.

For details, see **Deleting a Shared Application**.

NOTICE

1. Before deleting a shared application from a cloud phone server, the application must have been uninstalled from all cloud phones of the server. Otherwise, the application will fail to be deleted. The API will fail to be invoked and a list of cloud phones where the application is not uninstalled will be returned.
2. If there are multiple versions of a shared application in the shared space and the application of the earlier version is not installed on any cloud phone, the application of the earlier version will be automatically cleared periodically.

2.3 Application Scenarios of Shared Applications

After the TAR package of an application is pushed to the shared space, you can run commands to install, update, and uninstall the application.

2.3.1 Application Installation on Cloud Phones

2.3.1.1 Installing an Application

Scenarios

- Cloud gaming: You access a game after accessing a cloud phone.
- Personal cloud phones: After accessing a cloud phone, you can view the cloud phone screen and install existing applications in the shared space as required.

Procedure

Run the **appctrl** commands on the cloud phone. For details, see **appctrl Commands**.

1. Run the **appctrl install** command to install an application.
   Command: **appctrl install** *{package name}*
   Example command: **appctrl install com.huawei.***xxxx*
2. Run the **appctrl start** command to install and start the application.
   Command: **appctrl start** *{package name}*
   Example command: **appctrl start com.huawei.***xxxx*

NOTICE

If an application in the shared space has multiple versions, **appctrl install** and **appctrl start** automatically select the latest version of the application for installation.

2.3.1.2 Updating an Application

Scenarios

- You can update a shared application after its new version is released.
- When online resources for a game are updated, the resources in the shared application need to be updated to prevent users from waiting for the download and installation of the new online resources in cloud gaming.

Procedure

1. Install the latest version of an application on a single cloud phone by referring to **Querying Cloud Phones** and **Installing Applications on a Cloud Phone**.
2. If resources of this application are updated, start the application and download and update its resource files.
3. Generate a TAR package for the application and push the package to the shared space of the cloud phone server by referring to **Generating a TAR Package for an Application and Pushing It to an OBS Bucket** and **Pushing the TAR Package of an Application to the Shared Space**.
4. Run **appctrl** commands to perform the update. For details, see **appctrl Commands**.
   - Run **appctrl start**. The latest version of the application in the shared space will be installed and started on cloud phones.
   - Run **appctrl install**. The latest version of the application in the shared space will be installed on cloud phones.

NOTE

1. If an application in the shared space is not updated, **appctrl start** and **appctrl install** will not install the application.

2. For shared applications installed on a cloud phone, if their resources are updated online after the applications are started, the updated resources occupy the storage space of the cloud phone. To avoid storage occupation on cloud phones, you are advised to update applications and their online resources in the shared space of the cloud phone server in time.

2.3.1.3 Uninstalling an Application

Procedure

Run the **appctrl** command on a cloud phone to uninstall a shared application on it. For details, see **appctrl Commands**.

Usage guide: Run the **appctrl uninstall** *{package name}* command on the cloud phone.

Example:

appctrl uninstall com.huawei.xxxx

NOTICE

appctrl uninstall only applies to shared applications. Applications installed through native APIs cannot be uninstalled using this command.

2.3.1.4 Uninstalling Applications in a Batch

Scenarios

When a cloud phone allocated to you expires and needs to be reclaimed, all shared applications need to be uninstalled from it.

Procedure

Run **appctrl clear** on the cloud phone to uninstall all shared applications from it.

For details, see **appctrl Commands**.

NOTICE

appctrl clear only applies to shared applications installed using **appctrl install** or **appctrl start**. Applications installed using native APIs and pre-installed shared applications cannot be uninstalled using this command.

2.3.2 Pre-installation of Applications on Cloud Phones

2.3.2.1 Installing Pre-installed Applications

Scenarios

Some applications must be installed on all cloud phones on a cloud phone server, and by default, they will be installed after the cloud phones are reset.

Procedure

1. Push the TAR packages of the applications to the shared space of the cloud phone server and set **pre_install_app** to **1**. For details, see **Pushing the TAR Package of an Application to the Shared Space**.
2. Reset the cloud phones. All of the applications will be installed on the cloud phones automatically.

NOTICE

- Installing pre-installed applications will increase the time it takes to reset your cloud phones.
- If a cloud phone is in use and cannot be reset, run **appctrl install** to install the pre-installed applications one by one.
- To cancel the application pre-installation, push the TAR packages of the applications again and set **pre_install_app** to **0**. For details, see **Pushing the TAR Package of an Application to the Shared Space**. When you reset the cloud phones, the applications will not be automatically installed.

2.3.2.2 Updating Pre-installed Applications

Procedure

1. Install applications of new versions on a cloud phone by referring to **Installing Applications on a Cloud Phone**.
2. Start the applications to update resource files.
3. Generate TAR packages for the applications and push them to an OBS bucket. For details, see **Generating a TAR Package for an Application and Pushing It to an OBS Bucket**.
4. Push the TAR packages to the shared space of the cloud phone server and set **pre_install_app** to **1**. For details, see **Pushing the TAR Package of an Application to the Shared Space**.
5. Restart or reset the cloud phones to allow automatic update of the pre-installed application.

NOTICE

Updating pre-installed applications will increase the time it takes to reset or restart your cloud phones.

2.3.2.3 Uninstalling Pre-installed Applications

Scenario 1

Pre-installed applications are no longer required by one or some cloud phones.

Procedure

1. Uninstall pre-installed applications from cloud phones by referring to **Uninstalling an Application**.

   NOTICE

   If you reset the cloud phones, the uninstalled applications will be automatically reinstalled.

Scenario 2

Pre-installed applications are no longer required by all cloud phones on a cloud phone server.

Procedure

1. Uninstall the pre-installed applications from all cloud phones on the cloud phone server. For details, see **Uninstalling an Application**.
2. Delete the pre-installed applications from the shared space of the server. For details, see **Deleting an Application from the Shared Space**.

2.3.3 File Deployment on Cloud Phones

Some user services require resource files to be deployed on cloud phones. Before the deployment, these files need to be compressed into TAR packages, which are called configuration file packages in the following sections.

2.3.3.1 Presetting Files

Scenarios

You need to preset files on all cloud phones of a cloud phone server, such as keepalive scripts and application behavior monitoring files.

Procedure

1. Select a cloud phone from the list and save the files to a directory of the cloud phone. For details, see **Querying Cloud Phones**.

   For example, save the **test.txt** and **test.sh** files in the **/data/local/huawei/** and **/data/local/tmp/** directories respectively on the cloud phone.
2. Pack the files into a TAR package and upload it to an OBS bucket.

   Example API:


```
POST https://${CPH Endpoint}/v1/${project_id}/cloud-phone/phones/batch-storage
Header:
Content-Type: application/json
X-Auth-Token: ${token}
Body:
{
    "storage_infos": [{
        "phone_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        "include_files": [
            "/data/local/huawei/test.txt",
            "/data/local/tmp/test.sh"
        ],
        "bucket_name": "${bucket_name}",
        "object_path": "apk/${package_name}_${version_name}.tar"
    }]
}
```

Parameter descriptions:

- The elements in **include_files** is the absolute paths of the files to be packed on the cloud phone.
- *${package_name}* in **object_path** supports only one of **com.cph.config**, **com.cph.config.level1**, and **com.cph.config.level2**. *${version_name}* is used to identify the configuration package version and can be customized.

3. Push the TAR package of the configuration file to the target cloud phone server. For details, see **Pushing the TAR Package of an Application to the Shared Space**.

   Example API

   ```
   POST https://${CPH Endpoint}/v1/${projectId}/cloud-phone/phones/share-apps
   Header:
   Content-Type: application/json
   X-Auth-Token: ${token}
   Body:
   {
       "package_name": "${package_name}",
       "bucket_name": "${bucket_name}",
       "object_path": "apk/${package_name}_${version_name}.tar",
       "pre_install_app": 1,
       "server_ids": [
           "${server_id1}",
           "${server_id2}"
       ]
   }
   ```

   Set **pre_install_app** to **1** during push.

4. Reset or restart cloud phones to deploy all the files in the package to the corresponding directories of the phones.

NOTICE

1. Deploying a configuration file package on a cloud phone will occupy the storage space of the cloud phone. Ensure that the configuration file package is not too large.
2. You are advised to store the required files in a fixed directory on the cloud phone for easier file management. Then, pack these files into the **com.cph.config** package.
3. If all the three configuration packages **com.cph.config**, **com.cph.config.level1**, and **com.cph.config.level2** need to be preset, they will be deployed in the sequence of **com.cph.config**, **com.cph.config.level1**, and **com.cph.config.level2** when cloud phones are reset or restarted. The files with the same name will be overwritten in the same sequence.

2.3.3.2 Deploying Files

Scenarios

You can deploy files on some running cloud phones on a cloud phone server.

Procedure

1. Pack the files into a TAR package and upload it to an OBS bucket. For details, see **1** and **2** in **Presetting Files**.
2. Push the TAR package to the target cloud phone server. For details, see **Pushing the TAR Package of an Application to the Shared Space**.

   During the push, set **package_name** to one of the three supported configuration file package names and set **pre_install_app** to **0**.
3. Run the **appctrl applyConfig** *{configuration file package name}* command on the cloud phone. For details, see **appctrl Commands**.

2.3.3.3 Updating Files

Scenarios

Files deployed on cloud phones need to be updated.

For example, if the **/data/local/tmp/test.sh** script has a new version, you need to update the new version to cloud phones.

Procedure

1. Pack the new file into a TAR package and push it to the shared space of the cloud phone server. For details, see **1**, **2**, and **3** in **Presetting Files**.
2. Update the file to cloud servers.

   Method 1: Restart cloud phones to deploy the file on them automatically.

   Method 2: Run **appctrl applyConfig** to deploy the file. For details, see **appctrl Commands**.

NOTICE

1. If a file is associated with your services, for example, it is closely related to your application whose version matters, the file needs to be updated when the application is updated.
2. When resetting a cloud phone, only the files in the last pushed configuration file package will be deployed. So, pack all required files instead of just the updated ones every time you update a configuration file package.

   For example, the configuration file package deployed contains **1.txt** and **2.txt**. Now, **2.txt** needs to be updated. You need to pack **1.txt** and the updated **2.txt** and push them. If you only pack and push the updated **2.txt**, only **2.txt** will be deployed after the cloud phone reset.

2.3.3.4 Removing Preset Files

Scenarios

Files do not need to be deployed on any cloud phone of a cloud server when the phones are reset.

Procedure

1. Push the TAR configuration file package to the target cloud phone server by referring to 1, 2, and 3 in **Presetting Files**.

   Set the **pre_install_app** parameter in 3 to **0**.
2. Reset all cloud phones to remove all files in the configuration file package.

2.3.3.5 Deploying Files Using Multiple Configuration File Packages

Scenario 1

You need to preset public files for all cloud phone servers and special configuration files for some cloud phone servers to process different services.

For example, **1.txt** and **2.txt** are the public configuration files that you need to preset on cloud phones of all cloud phone servers. **a.txt** and **b.txt** are two special configuration files that you need to preset as they only take effect for specific server groups (server group A and server group B). Specifically, **a.txt** only takes effect for server group A, and **b.txt** only takes effect for server group B.

In this case, you need to compress **1.txt** and **2.txt** and push them as **com.cph.config** to all cloud phone servers.

Pack **a.txt** separately and push it as **com.cph.config.level1** to the cloud phone servers in server group A.

Pack **b.txt** separately and push it as **com.cph.config.level1** to the cloud phone servers in server group B.

If **1.txt** and **2.txt** need to be updated, pack and push the updated files as **com.cph.config** to all the cloud phone servers.

If **a.txt** or **b.txt** needs to be updated, pack and push the updated file as **com.cph.config.level1** to the corresponding cloud phone servers.

Procedure

1. Push the TAR configuration file package to the target cloud phone servers by referring to **1**, **2**, and **3** in **Presetting Files**.

   Set the **pre_install_app** parameter in **3** to **1**.
2. Deploy the files on cloud phones by referring to **4** in **Presetting Files**.

Scenario 2

Public files, such as **1.txt** and **2.txt**, have been preset for all cloud phones on a cloud phone server. Some cloud phones require additional configuration files, such as **3.txt** and **4.txt**.

In this case, you need to pack and push **1.txt** and **2.txt** as **com.cph.config** to the cloud phone servers. You also need to pack and push **3.txt** and **4.txt** as **com.cph.config.level1** to the cloud phone servers.

Procedure

1. Push the TAR configuration file package to the target cloud phone servers by referring to **1** and **2** in **Deploying Files**.

When pushing **com.cpm.config.level1**, set **pre_install_app** in **2** to **0**.

2. Deploy the files on cloud phones by referring to **3** in **Deploying Files**.

2.4 appctrl Commands

2.4.1 Running the appctrl Commands

To connect to a cloud phone and execute the **appctrl** commands, you have the option of using ADB or calling the API.

1. ADB: See **ADB (Recommended)**.
2. API: See **Running adb Commands Asynchronously**.
3. The service programs or scripts on the cloud phone invoke the **appctrl** commands.

   NOTE

   You must have **root** privilege for the service programs or scripts.

2.4.2 Running appctrl install to Install Applications

Scenarios

Install an application on a cloud phone.

Prerequisites

The TAR package of the application has been pushed to the cloud phone server.

Command

appctrl install *{package name}*

Example command: **appctrl install com.huawei.***xxxx*

2.4.3 Running appctrl start to Install and Start an Application

Scenarios

Install an application on a cloud phone and start the application.

Prerequisites

The TAR package of the application has been pushed to the cloud phone server.

Command

appctrl start *{package name}*

Example command: **appctrl start com.huawei.***xxxx*

NOTICE

For service applications, there is no entry activity. You need to run **appctrl install** to install the applications and then run **am** to start their frontend and backend services.

2.4.4 Running appctrl uninstall to Uninstall an Application

Scenarios

If a shared application is no longer used on a cloud phone, you can uninstall it.

Command

appctrl uninstall *{package name}*

Example command: **appctrl uninstall com.huawei.***xxxx*

2.4.5 Running appctrl clear to Uninstall Applications in a Batch

Scenarios

Uninstall all shared applications that are not preinstalled ones.

Command

appctrl clear

For example, if there are two non-preinstalled shared applications on a cloud phone, running the **appctrl clear** command will batch uninstall them.

2.4.6 Running appctrl applyConfig to Deploy Files

Scenarios

Deploy files on a cloud phone.

Prerequisites

- The TAR package of the configuration files has been pushed to the cloud phone server. The package name must meet the requirements of this command.
- Only the following package names are supported: **com.cph.config**, **com.cph.config.level1**, and **com.cph.config.level2**.

Command

appctrl applyConfig *{configuration file package name}*

Example command: **appctrl applyConfig com.cph.config.level2**

3 Installing an Application on Cloud Phones in a Batch

After you install an application on a cloud phone, you can share and install the application on other cloud phones by calling an API.

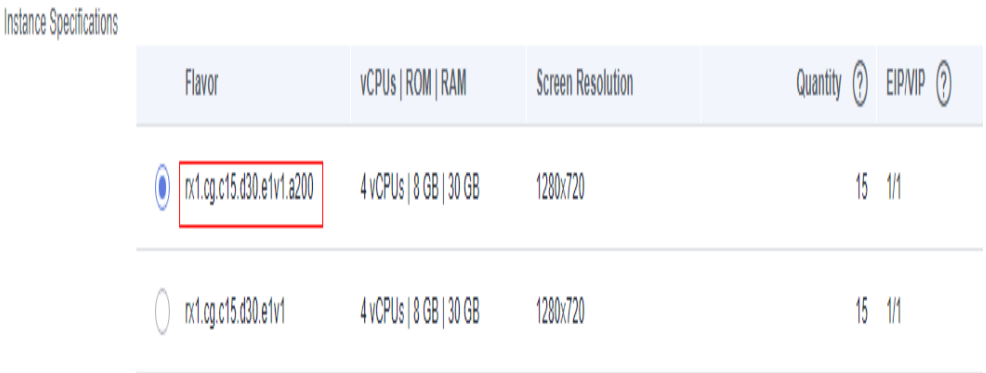
NOTE

The cloud phone that has the application installed is regarded as a seed cloud phone.

Constraints and Limitations

- This solution applies only to cloud phones whose specifications name does not contain **qemu**.

  Figure 3-1 Specifications

- You can identify whether a server is using storage 1.0 or 2.0 on its details page.

Figure 3-2 Server details

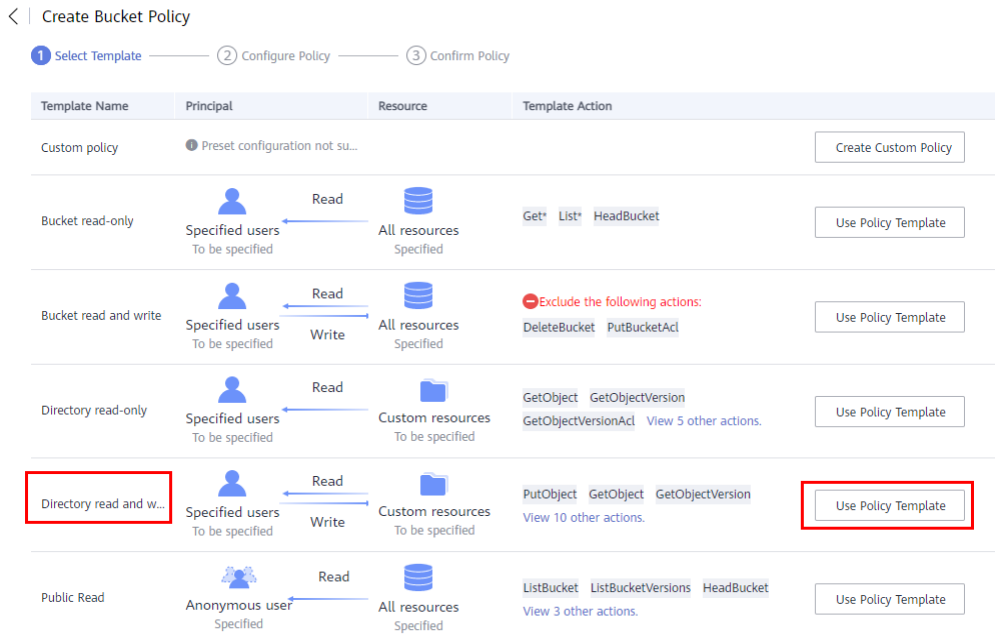
- The seed cloud phone must be a cloud phone that has not been operated on. Otherwise, **reset it**.

Procedure

1. Log in to the **OBS console**, create an OBS bucket by referring to **Managing Cloud Phones in Batches**, and set access permissions for the bucket.

   To successfully upload files to the bucket, select **Directory read and write** when configuring the bucket policy.

   Figure 3-3 Creating a bucket policy

2. Connect to the seed cloud phone in ADB mode and install the required application on it.

   For details, see **How Do I Install Applications on a Cloud Phone?**

3. Call an API to export data of the seed cloud phone, pack the data, and upload the data package to the OBS bucket created in **1**.

   curl command

   ```
   curl -i -k -X POST "https://${CPH Endpoint}/v1/${projectId}/cloud-phone/phones/batch-storage" -H "Content-Type: application/json" -H "X-Auth-Token: $token" -d '
   {
      "storage_infos": [
         {
            "phone_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
            "include_files": [
   ```

```
            "/data/app/${package-name}-1",
            "/data/data/${package-name}",
            "/data/media/0/Android/data/${package-name}"
        ],
        "bucket_name": "${bucket_name}",
        "object_path": "${your_dir}/${package-name}.tar"
    }
  ]
}'
```

Parameters:

- **phone_id**: ID of the seed cloud phone
- **bucket_name**: name of the OBS bucket for storing the data exported
- **object_path**: OBS path for storing the data exported

CAUTION

If you want to pack all data of applications on the seed cloud phone, the following three paths must be included:

- **/data/app/${package-name}-1**

  ${package-name} may not be followed by **-1**. Configure this parameter as required.
- **/data/data/${package-name}**
- **/data/media/0/Android/data/${package-name}**

Example

```
curl -i -k -X POST "https://${CPH Endpoint}/v1/${projectId}/cloud-phone/phones/batch-storage" -H
"Content-Type: application/json" -H "X-Auth-Token: $token" -d '
{
   "storage_infos": [
      {
         "phone_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
         "include_files": [
            "/data/app/com.taptap-1",
            "/data/data/com.taptap",
            "/data/media/0/Android/data/com.taptap"
         ],
         "bucket_name": "your-bucket-name",
         "object_path": "your/dir/taptap.tar"
      }
   ]
}'
```

4. View the OBS bucket created in **1** and check whether the file packed and uploaded in **3** is successfully uploaded. If yes, go to the next step.
5. Call an API to push files from an OBS bucket to the server.
   - Servers with storage 2.0 (recommended)

     Example:

     ```
     curl -i -k -X POST "https://${CPH Endpoint}/v1/${projectId}/cloud-phone/phones/share-apps" -H
     "Content-Type: application/json" -H "X-Auth-Token: $token" -d '
     {
        "package_name": "com.miniteck.miniworld",
        "pre_install_app": 0,
        "bucket_name": "your-bucket-name",
        "object_path": "your/dir/miniworld.tar",
        "server_ids": ["1678567b8bab40f93711234cb8","1234567b8bab40ffb711234cb6"]
     }'
     ```

Parameter:

- **bucket_name** and **object_path** are the same as those in 3.
- **pre_install_app** (optional): whether to pre-install an application. **0** indicates it will not be pre-installed and you need to manually install it. **1** indicates the application will be pre-installed after the cloud phones are reset.
- **server_ids**: IDs of servers that receive the file pushed. If multiple server IDs are specified, the application can be installed on cloud phones on multiple servers.

NOTE

For details about this API, see **Pushing Shared Applications**.

– Servers with storage 1.0

Example:

```
curl -i -k -X POST "https://${CPH Endpoint}/v1/${projectId}/cloud-phone/phones/share-files" -H "Content-Type: application/json" -H "X-Auth-Token: $token" -d '
{
   "bucket_name": "your-bucket-name",
   "object_path": "your/dir/taptap.tar",
   "server_ids": ["1678567b8bab40f93711234cb8","1234567b8bab40ffb711234cb"]
}'
```

Parameters:

- **bucket_name** and **object_path** are the same as those in 3.
- **server_ids**: IDs of servers that receive the file pushed. If multiple server IDs are specified, the application can be installed on cloud phones on multiple servers.

NOTE

For details about this API, see **Pushing Shared Storage Files**.

6. Install the application.
   – Servers with storage 2.0 (recommended)

   If you set **pre_install_app** to **1** in the previous step, the application will be installed automatically after you reset the cloud phones. If **pre_install_app** is set to **0**, run **appctrl install** *package-name* on the cloud phones to install the application. *package-name* is the value of **package_name** in 5, for example, **com.miniteck.miniworld**.

   CAUTION

   If an application is pre-installed, after it is uninstalled, it will be installed automatically again after the cloud phones are reset.

   – Servers with storage 1.0

   Log in to the CPH console, click the name of the server that has received the files pushed. In the **Cloud Phones** area, select all cloud phones on which the application needs to be installed, and click **Reset**.

> **CAUTION**
>
> This step is a reset operation, not a restart operation.

Execution Results

For servers that use storage 1.0, the applications have been installed after the cloud phones are reset.

For servers that use storage 2.0, the applications can be started by running **appctrl start** *package_name*.

4 Simulating a Cloud Phone Location

Introduction

There are four methods of determining the location of a phone: GPS longitude and latitude, base station, Wi-Fi, and IP address. CPH can simulate GPS and Wi-Fi location data.

Method	Precision	Scenario
GPS	High	Apps that require continuous positioning, such as map and navigation apps
Wi-Fi	Medium	Auxiliary positioning for GPS and base stations

Prerequisites

You have purchased a cloud phone server and connected to a cloud phone using ADB. For details, see **Buying a Cloud Phone Server**.

GPS

1. Obtain the GPS longitude and latitude.

   The GPS data simulation of CPH is based on the World Geodetic System 1984 (WGS84). If other geodetic systems such as GCJ02 and BD09 are used, deviations may occur during data injection.

   Example:

   ```
   mLocationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
   mLocationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 0,
   locationListener);
   private final LocationListener locationListener = new LocationListener() {
         public void onLocationChanged(Location location) {
   // location contains GPS data such as the longitude and latitude.
         }
   };
   ```

2. Inject GPS data.
   - Inject the GPS data using a shell command.

On the cloud phone, run **adb shell "echo '***parameters***' > /data/gps/fifo"**. Separate parameters with colons.

Table 4-1 describes the parameters.

Table 4-1 GPS data injection

Parameter	Description	Mandatory	Default Value	Remarks
latitude	Latitude. The north latitude is a positive value and the south latitude is a negative value.	Yes	22.657501	Value range: –90.000000 to 90.000000 Unit: degree (°)
longitude	Longitude. The east longitude is a positive value and the west longitude is a negative value.	Yes	114.055939	Value range: –180.000000 to 180.000000 Unit: degree (°)
altitude	Altitude.	No	51.0	Unit: meter
speed	Speed.	No	0.0	Unit: meter
bearing	Azimuth. 0° indicates the true north, 90° indicates the true east, 180° indicates the true south, and 270° indicates the true west.	No	30.0	Value range: 0.0 to 360.0 Unit: degree (°)
accuracy	Positioning accuracy.	No	90.0	Unit: meter

Example: Change the cloud phone location to 114.055939 degrees East longitude and 22.657501 degrees North latitude.

```
adb -s 127.0.0.1:local-idle-port shell "echo 'longitude=114.055939:latitude=22.657501' > /data/gps/fifo"
```

- Continuously inject GPS data (non-blocking injection recommended).

  Use self-developed or third-party SDK application code to continuously inject GPS data in **O_NONBLOCK** (non-blocking) mode.

Example:

```
#define GPSFifoName "/data/gps/fifo"
if((fifo_fd = open(GPSFifoName, O_WRONLY | O_NONBLOCK)) < 0) {
ALOGE("open fifo "%s" (write) fail, error = %s\n", GPSFifoName, strerror(errno));
return;
}
// Use colons (:) to separate cmd parameters.
char* cmd = "longitude=113.370592:latitude=23.123642";
len = strlen(cmd);
if(write(fifo_fd, cmd, len) != len) {
ALOGE("%s: write "%s" to "%s" fail: %s", FUNCTION, cmd, GPSFifoName, strerror(errno));
}
```

Wi-Fi

1. Obtain Wi-Fi data.

   Example:

   ```
   WifiManager wifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
   WifiInfo wifiConnection = wifiManager.getConnectionInfo();
   if (wifiConnection != null) {
      String bssid =  wifiConnection.getBSSID();
   }
   ```

2. Inject the Wi-Fi data.

   Configure **com.cph.wifi.bssid** with the Wi-Fi BSSID to inject the Wi-Fi data.

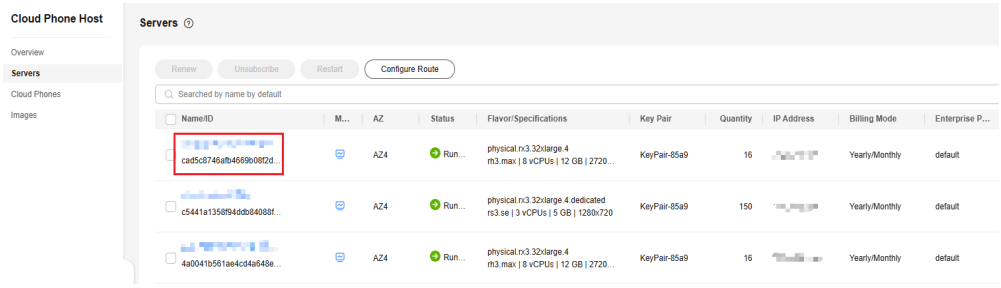
   Example:

   ```
   setprop com.cph.wifi.bssid 02:00:00:00:00:00
   ```

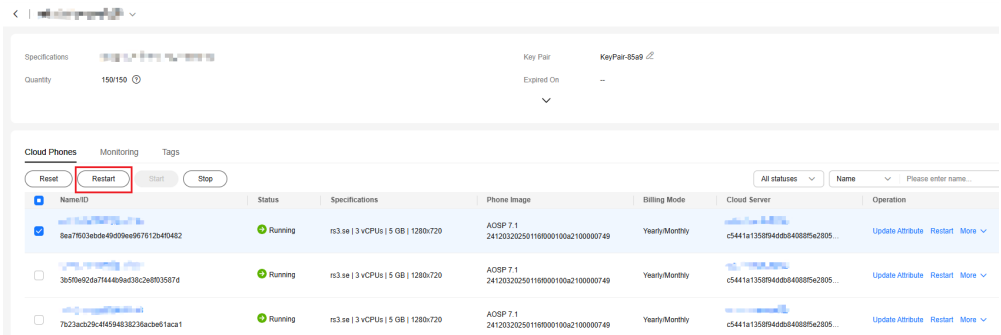

5 Using the Cloud Phone Camera

Step 1: Replace the Image of a Cloud Phone

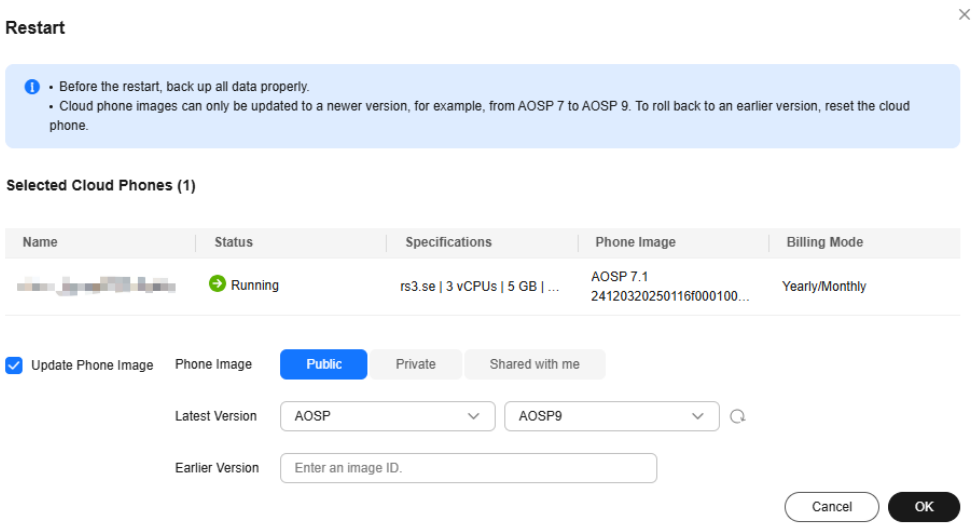
1. View **What's New**, select an image released on or after October 9, 2020, and copy the image ID.
2. Log in to the **CPH console** and switch to the region where your resources are located.
3. Choose **Servers** to view servers.



4. Click the name of a server to go to the server details page, select a cloud phone, and click **Restart**.



5. Select the **Update Phone Image** checkbox and enter the image ID selected in **1**.



6. Click **OK**.

Step 2: Upload a Picture to the Cloud Phone

Upload a picture to the **/data/local/tmp/** directory of the cloud phone in either of the following ways. The following uses **pic.jpeg** in the **/path/to/local** directory as an example.

- Method 1: Run the **adb push** command to push the picture.

  Connect to the cloud phone through ADB, and run the following commands:

  adb push </path/to/local/pic.jpeg> /data/local/tmp/pic.jpeg

  adb shell chmod 644 /data/local/tmp/pic.jpeg
- Method 2: Call the CPH API to push the picture.

  See **Pushing Files Using ADB Commands**.

NOTICE

- The size of the picture to be uploaded must be 480 (width) x 640 (height). If the size is not 480 x 640, the picture may be zoomed in or out on the camera.
- Only JPEG and PNG pictures stored in the **/data/local/tmp/** path are supported.
- Picture permissions must be at least 644 (rw-r--r--).

Step 3: Configure Cloud Phone Attributes

You can configure cloud phone attributes in either of the following ways:

Method 1: Use ADB to connect to the cloud phone and run the **adb** command.

adb shell setprop com.cph.cam_local_pic_path /data/local/tmp/pic.jpeg

If you use this method, the cloud phone attributes become invalid after the cloud phone is restarted.

Method 2: Call the CPH API to configure attributes.

Set **"com.cph.cam_local_pic_path":"/data/local/tmp/pic.jpeg"** for the cloud phone. For details, see **Updating Properties of Cloud Phones**. The attributes will always be valid even after the cloud phone is restarted.

Step 4: Test the Camera

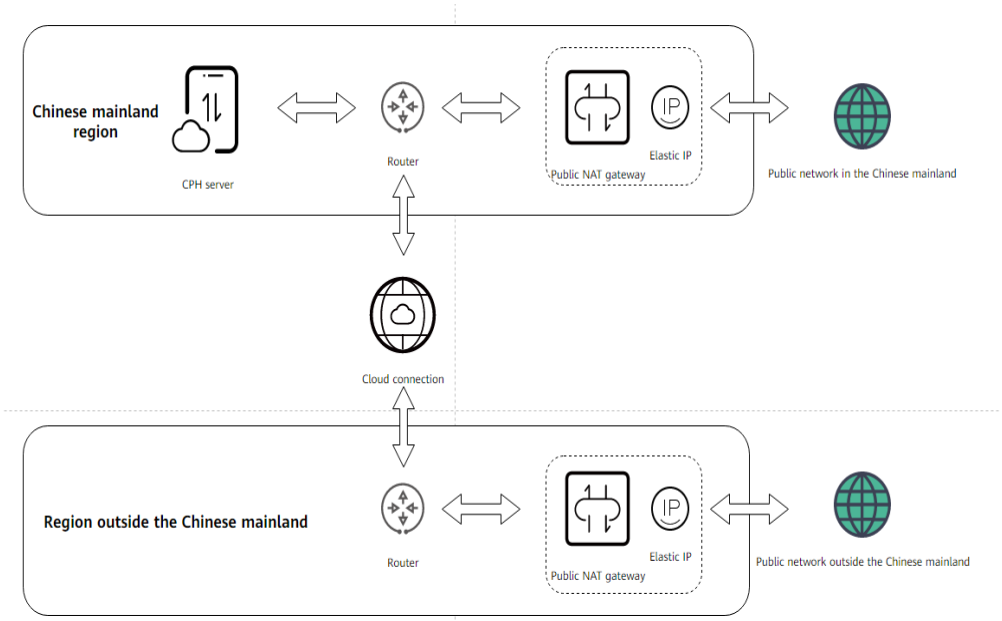
Install an application that needs to use the camera, open the application, and check whether the picture you uploaded is displayed in the viewfinder.

NOTE

CPH supports only the rear camera.

6 Allowing a Cloud Phone Server to Access a Public Network Outside the Chinese Mainland

The following figure shows how to allow a cloud phone server to access a public network outside the Chinese mainland.



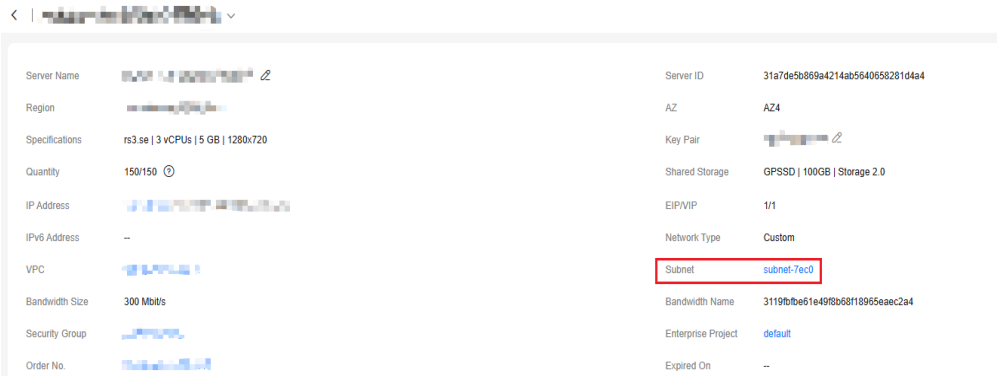
Constraints and Limitations

- This practice applies only to cloud phone servers without EIPs. That is, the number of EIPs in the cloud phone specifications must be 0. The number of virtual IP addresses is not limited.

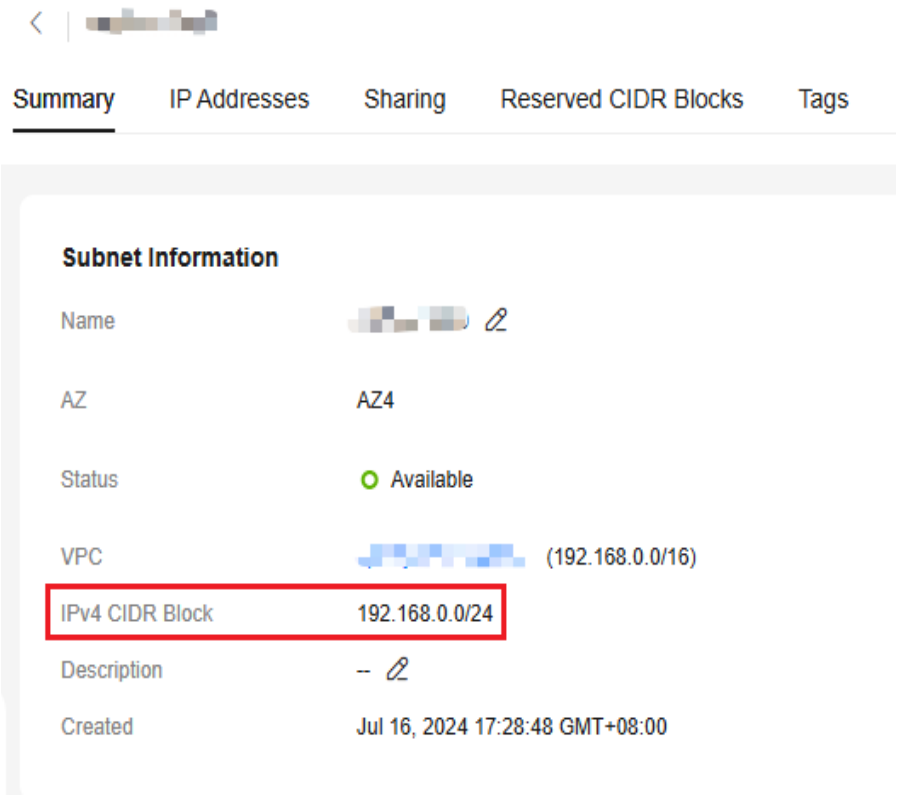
	Flavor	vCPUs | ROM | RAM	Screen Resolution	Quantity	EIP/VIP
○	rx1.cp.c15.d46.e1v1	4 vCPUs | 16 GB | 46 GB	1280x720	15	1/1
◉	rx1.cp.c60.d10.e0v60	2 vCPUs | 3.5 GB | 10 GB	1280x720	60	0/60

Procedure

1. Apply for a cross-border permit. For details, see **Cross-Border Permits**. Proceed with the next step only if you have obtained the cross-border permit.
2. Log in to the **CPH console**, choose **Servers**, and click the cloud phone server whose traffic is to be diverted. On the server details page, locate **Subnet**.

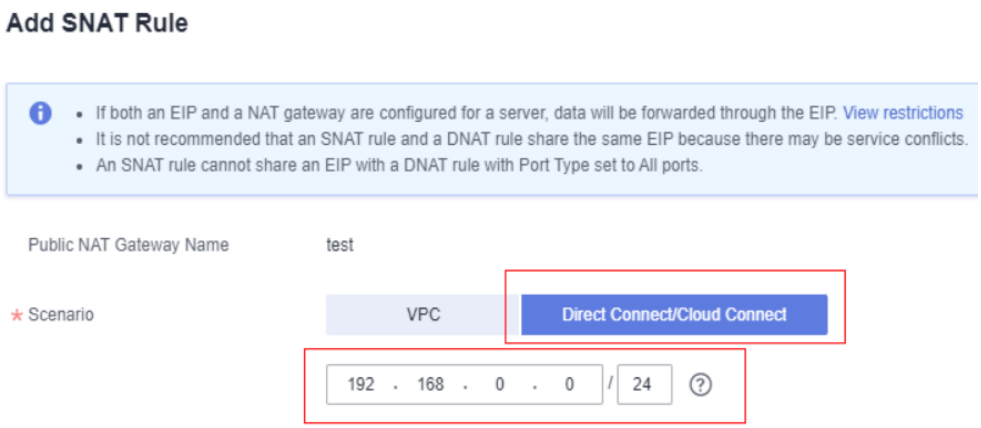


3. Click the subnet name. On the subnet details page, find **IPv4 CIDR Block** and record the subnet CIDR block of the cloud phone server, for example, **192.168.0.0/24**.

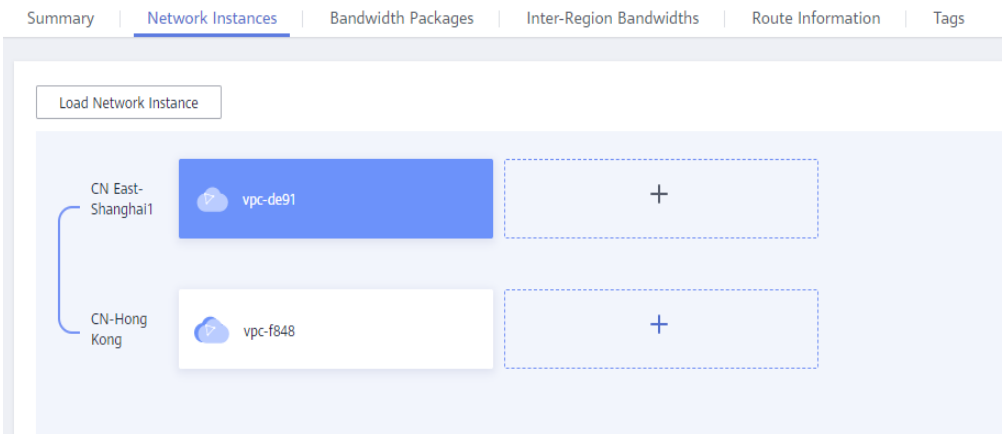


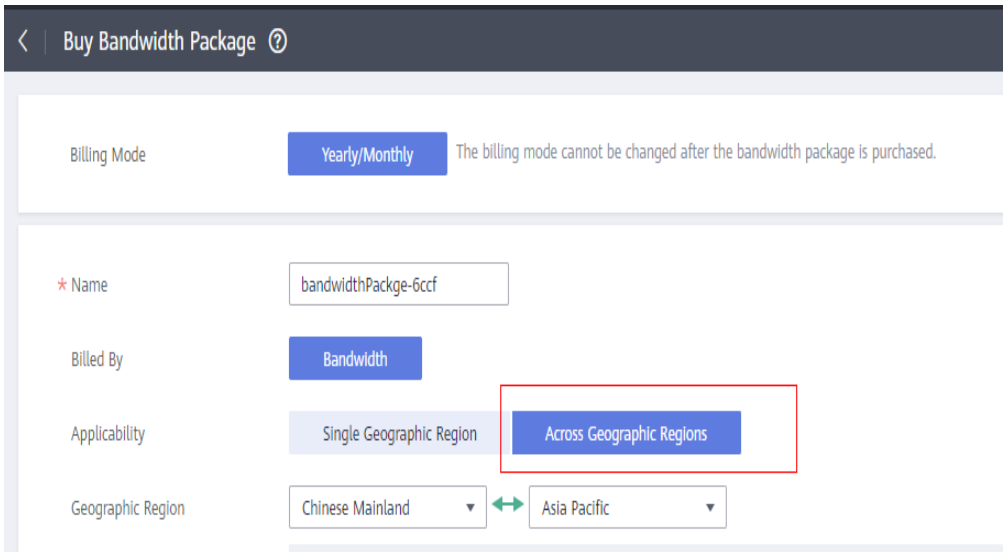
4. Choose **Networking** > **NAT Gateway**.
5. Select the region where you want your cloud phone server to access the public network outside the Chinese mainland, for example, **CN-Hong Kong**.
6. Purchase an EIP and a public NAT gateway. Configure an SNAT rule. Add a route with 0.0.0.0/0 as the destination and the public NAT gateway as the next hop. For details, see **Using a Public NAT Gateway to Enable Servers to Share One or More EIPs to Access the Internet**.

When you add the SNAT rule, select **Direct Connect/Cloud Connect** for **Scenario** and enter the subnet CIDR block of the cloud phone server recorded in 3.

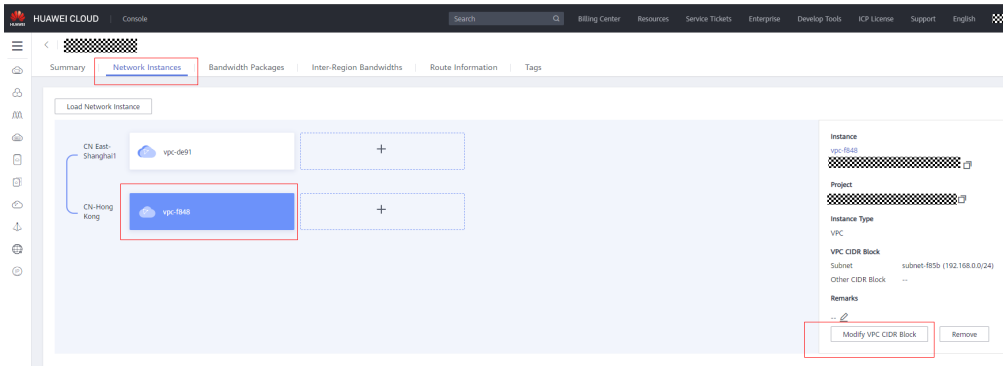


7. Choose **Networking** > **Cloud Connect**.
8. Create a cloud connection, and load the VPC where the cloud phone server is deployed and the VPC where the public NAT gateway (purchased in 6) is deployed to the cloud connection. For details, see **Using a Cloud Connection to Connect VPCs in the Same Account But Different Regions**. Purchase a bandwidth package for communications between geographic regions, in this example, from the Chinese mainland to Asia Pacific. Bind the bandwidth package to the cloud connection.

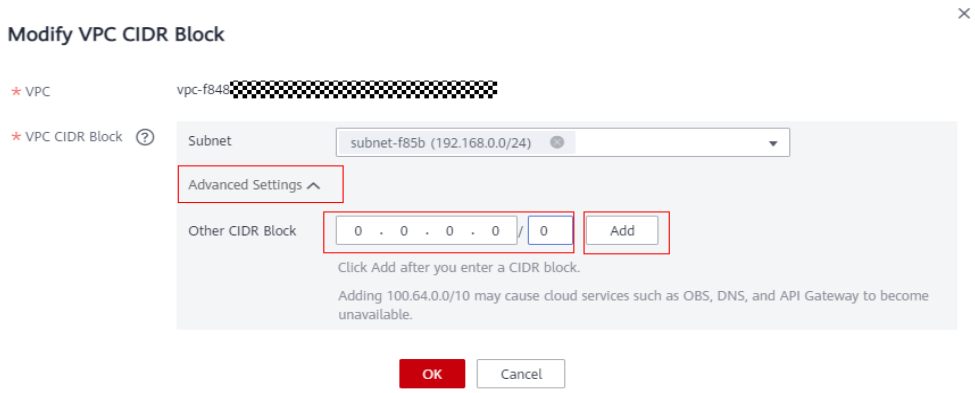




9. On the **Network Instances** tab of the cloud connection, select the loaded VPC in the CN-Hong Kong region and click **Modify VPC CIDR Block**.



In the displayed **Modify VPC CIDR Block** dialog box, click **Advanced Settings**, enter **0.0.0.0/0**, click **Add**, and click **OK**.

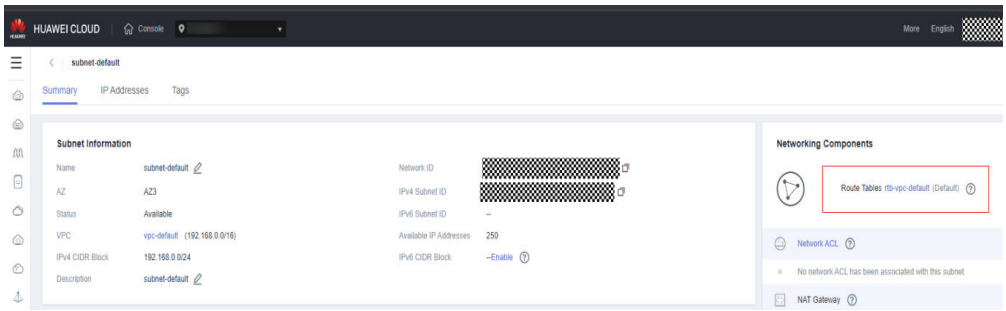


Now your cloud phone server can access the public network outside the Chinese mainland. All traffic from this server is diverted to the cloud connection and then to the public NAT gateway in the CN-Hong Kong region, so this server can use the EIP bound to the public NAT gateway to access the public network outside the Chinese mainland. To verify the above configurations, you can use your cloud phone server to access the public network outside the Chinese mainland.

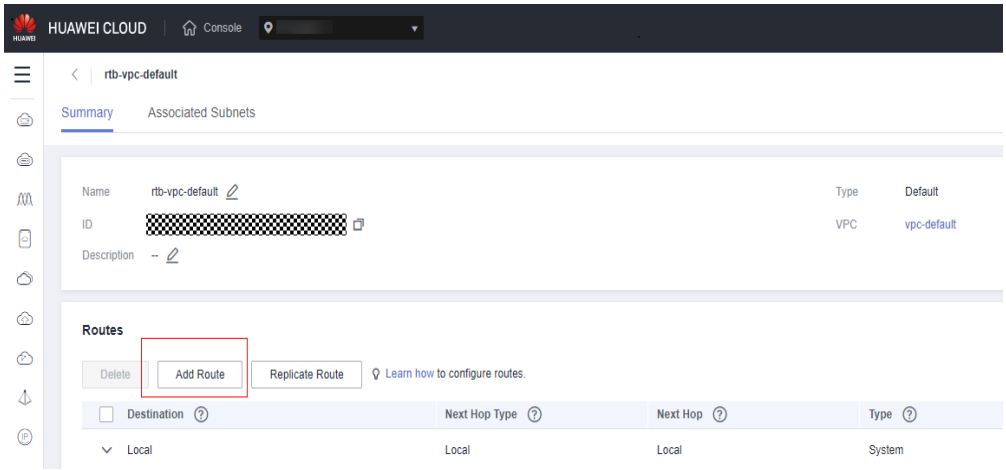
If you do not need your cloud phone server to access a public network inside the Chinese mainland, skip the following steps.

(Optional) Allowing a Cloud Phone Server to Access a Public Network Inside the Chinese Mainland

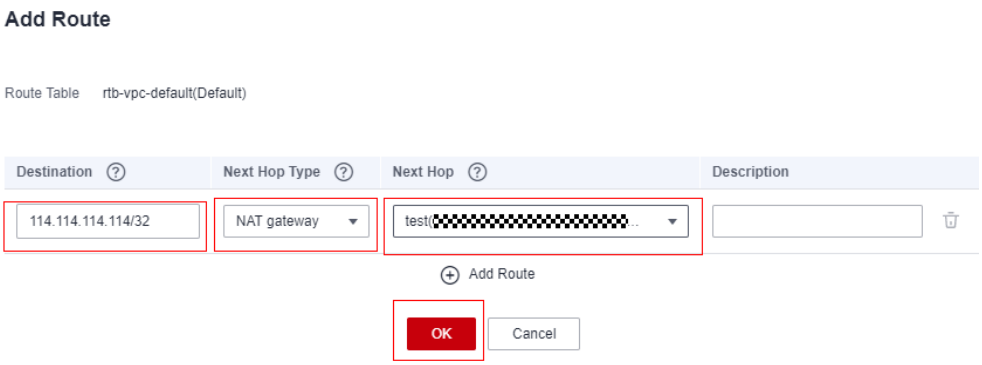
1. Purchase an EIP and a public NAT gateway in the region where the cloud phone server is deployed. Add an SNAT rule. For details, see **6**. You do not need to add a route with 0.0.0.0/0 as the destination and the public NAT gateway as the next hop.
2. Repeat **2** and **3** to view the subnet of the cloud phone server and the route table of the subnet.



3. Click the name of the route table. On the displayed page, click **Add Route**.



4. In the dialog box that is displayed, set **Destination** to the IP address or CIDR block where the cloud phone server traffic is to be diverted, set **Next Hop Type** to **NAT gateway**, set **Next Hop** to the public NAT gateway purchased in **1**, and click **OK**.



5. Repeat **4** if traffic to your cloud phone server needs to be diverted to other IP addresses or CIDR blocks.

Now when you access the IP address configured with traffic diversion from the cloud phone server, the traffic will be diverted from the EIP configured for the NAT gateway purchased in the Chinese mainland region, and other traffic will be diverted to the cloud connection from the EIP configured for the NAT gateway purchased outside the Chinese mainland region.

7 Delegating CPH to Operate OBS Buckets

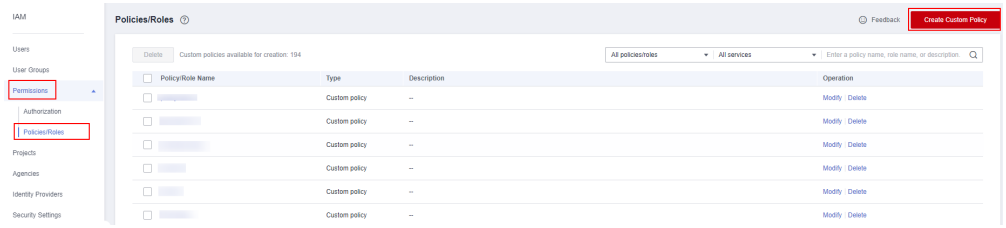
The administrator can create custom policies on IAM, assign custom policies to OBS buckets for refined access control, and delegate the policies to CPH to back up and restore cloud phone data and install applications.

Procedure

- Create a custom policy.

1. Log in to the **IAM console**.
2. In the navigation pane, choose **Permissions** > **Policies/Roles**. In the upper right corner, click **Create Custom Policy**.

Figure 7-1 Create Custom Policy



3. Enter a policy name.

   Select **Visual editor** for **Policy View**.

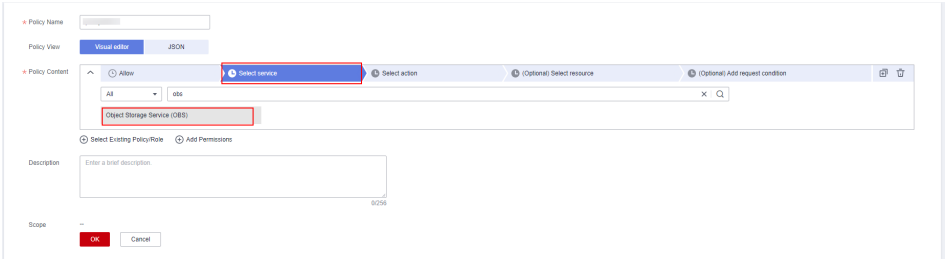
Figure 7-2 Visual editor

4. Set the policy content.

   a. Select **Allow**.

   b. Click **Select service** and select **Object Storage Service (OBS)**.

Figure 7-3 Configuring a policy



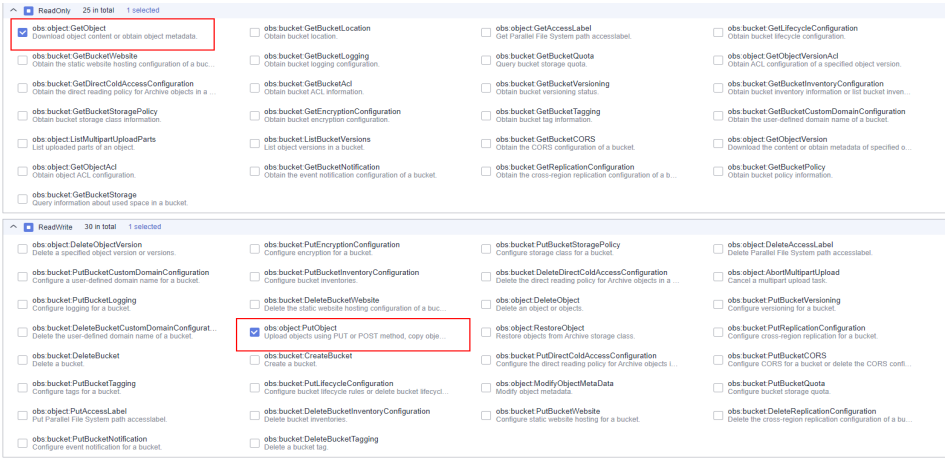
c. For **Actions**, select actions as required. For details, see **Object Actions**.

To delegate CPH to perform operations on OBS buckets, at least the following two actions must be selected:

obs:object:GetObject

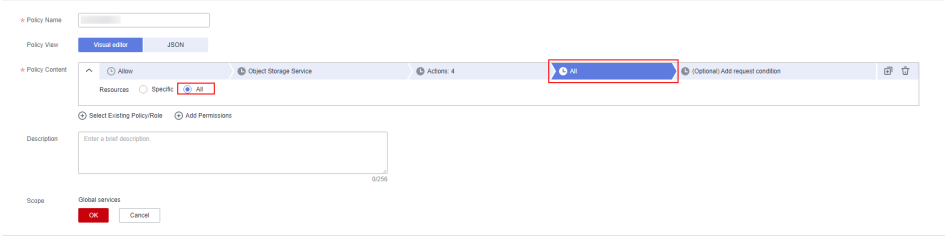
obs:object:PutObject

Figure 7-4 Selecting actions



d. Select **All** for **Resources**. For details about how to select specific resources, see descriptions of specific resources in **Creating a Custom Policy**.

Figure 7-5 Selecting resources

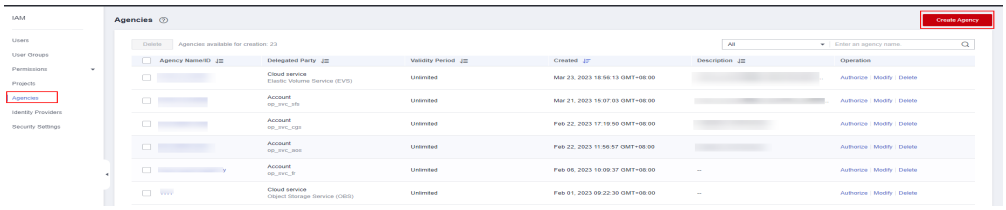


5. Click **OK**.

- Create an agency.

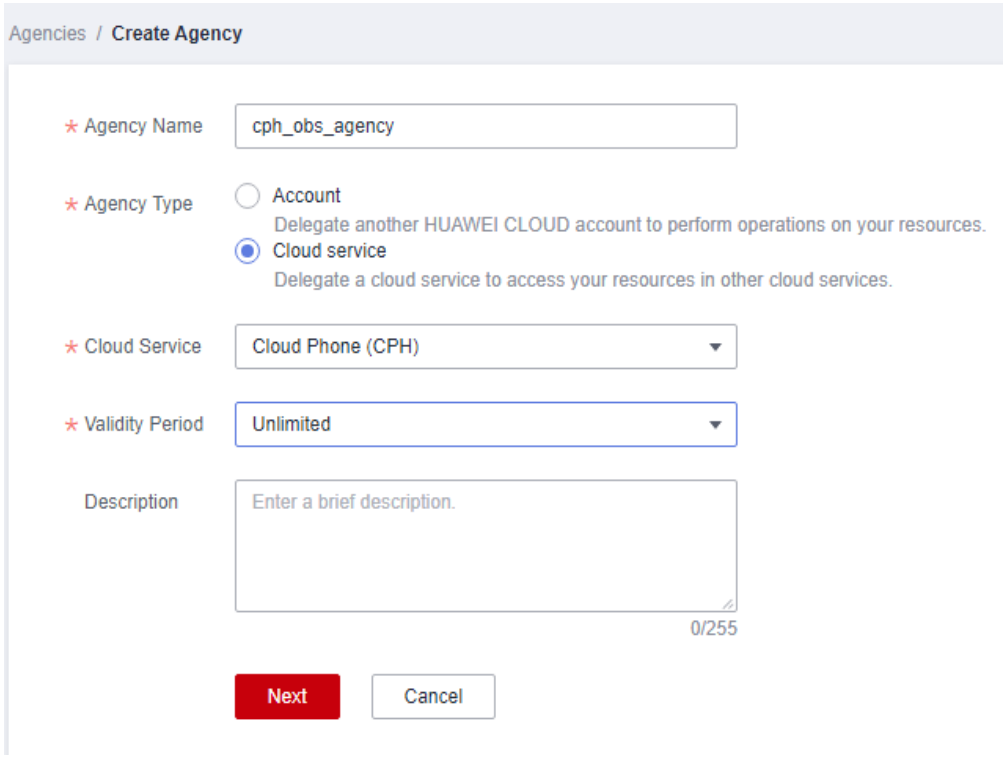
1. Log in to the **IAM console**.
2. In the navigation pane, choose **Agencies** and then click **Create Agency** in the upper right corner.

Figure 7-6 Agencies



3. Configure parameters shown in the following figure and click **Next**.

Figure 7-7 Create Agency

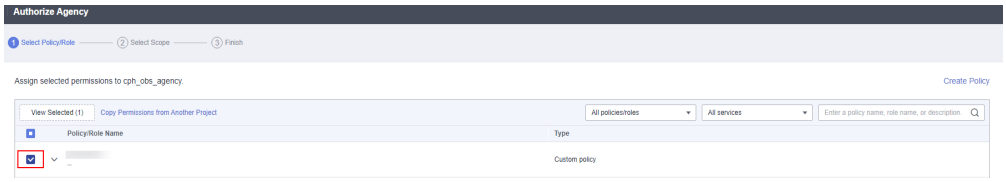


CAUTION

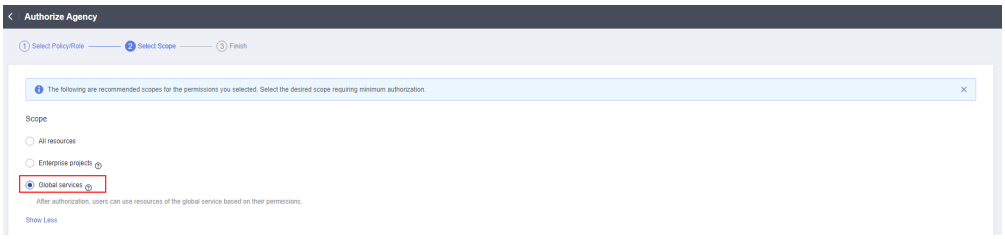
The agency name must be **cph_obs_agency**.

4. Select the custom policy created in **Create a custom policy** and click **Next**.

Figure 7-8 Selecting a policy



5. Select **Global services** and click **OK**.



8 Changing the AOSP Version of a Cloud Phone

This section describes how to change the AOSP version of a cloud phone.

Prerequisites

The image version of cloud phones on the target server is AOSP 7.

Precautions

1. A version change is a high-risk operation. To learn how to back up user data, refer to **Exporting Data from Cloud Phones in Batches**. To learn how to roll back the image to the original version, refer to **Restoring Cloud Phone Data**.
2. During a version change, the values of attributes such as **ro.build.version.release** and **ro.build.fingerprint**, which define the Android version, are automatically changed to values that identify the target image version.

Upgrading the AOSP Version

Method 1 (User data retained)

Call the API for **restarting cloud phones** to change the AOSP version and retain user data.

Notes:

- The restart API can only change the image from an earlier version to a later one.
- If you do not need to retain user data, use method 2 to change to the AOSP version, which reduces application incompatibility issues.

Method 2 (User data not retained)

To change the AOSP version, call the API for **resetting cloud phones**.

Notes:

- You can call the restart API to upgrade an image version or roll back the image to an earlier version.

Rolling Back the AOSP Image to an Earlier Version

For compatibility reasons, user data can be retained only when an earlier version is changed to a later version. To roll back to an earlier version, you can only call the API for **resetting cloud phones** that does not retain user data.

NOTE

If you roll back to an earlier image version by calling the API for resetting cloud phones, to view the screens of the cloud phones, run the **adb disconnect** *ip:port* command on the ADB clients of cloud phones of some versions and then run the **adb connect** *ip:port* command again.

