CloudApp

Service Provider Guide

Product Documentation

Contents

Service Provider Guide
Overview of the Listing Process

Last updated：2025-04-23 15:15:51

Overview

CloudApp provides standard processes and tools for service providers to list applications. The process mainly includes four parts: partner onboarding, installation package creation, software commercialization, and commercialization.

Partner Onboarding

Partners can submit enterprise information based on entry requirements. CloudApp will perform access control assessment based on partners' enterprise qualifications and service capabilities.

Production of Installation Package

Basic process for application developers to list applications: see the table below.

Process Stage	Description
Assess production cost	Assess the approximate production cost based on the situation of the packaging software and the technology stack of the technical team.
Become a Product Partner	Provide the application developer's entity, UIN, contact information, etc. to become a CloudApp product partner. If required, submit a ticket to contact us. We will arrange for an architect to connect exclusively.
Log in to the partner center	The product partner center is the backend for managing the production of installation packages.
Production of Installation Package	Start the production of the installation package, including preparing the application image, describing the resource stack, and describing the service orchestration.
Integrate License	Integrate the software with the Cloud App's License.

Installation Package Management	Push installation package, test, version management and release.
Application access	For listed applications, we provide application development parties with supported features related to application access, such as application APIs, user interfaces and cloud resource invocation.

Software Commercialization

Basic process from product creation to listing: see the table below.

Process Stage	Description
Create product	Complete the required product information according to the system's shelving requirements, including product specification, price, and description. Confirm that everything is correct and then submit the CloudApp for review.
CloudApp review	Wait for the CloudApp review of product information.
Associate installation package	Associate the created installation package with the product.
List product	Confirm product information and installation package information, then list the product.
Product access and installation permission settings	Set the customer scope for product accessibility and installability.

Commercialization

After product listing, it can be sold in the CloudApp store.

Making and Managing Installation Packages Estimating Production Costs

Last updated：2025-04-23 14:26:03

Before the publication of the application, it is recommended that the application developer first perform a cost estimation in terms of software and technology.

Software Self-Check

What was the previous delivery type of your software?

SaaS delivery: The cloud application version is a dedicated instance for customers. There is no need to process tenant logic. Tenant logic needs to be removed or processed in single-tenant mode.

Private deployment: The main workload is adapting to Tencent Cloud's runtime environment.

Software delivered as source code: It needs to be built into a runnable mirror on Tencent Cloud for listing.

What middleware does your software depend on?

Confirm the middleware that the software currently depends on. Commonly used middleware for software includes database MySQL, Redis, Cloud Object Storage, Message Queue, Cloud Log Service, etc.

Confirm whether there is a cloud native version corresponding to the required middleware.

If there is a corresponding cloud native version, we recommend that software select cloud native middleware, such as Cloud Database CDB. This way can help customers obtain the reliability and agility brought by Tencent Cloud Native services, thereby improving the competitiveness of the software.

If there is no corresponding cloud native version, the developer can use self-built middleware, such as the ETCD database self-built based on CVM.

If the software previously used self-built middleware and plans to be adjusted to use cloud native middleware, there is a development workload for compatibility adjustment.

What container orchestration technology does your software use?

If your software runs based on container orchestration and the used one is not Helm Chart orchestration, you need to produce an artifact of Helm Chart version orchestration.

What method does your software use for LICENSE authorization?

If it is free software, you do not need to manage the LICENSE.

If the software is charged and the LICENSE issuance and activation are based on a self-built LICENSE service, it needs to be modified to use the Cloud App LICENSE service.

If the software is charged but no LICENSE management has been done before, it is recommended to integrate with the Cloud App LICENSE service to manage the LICENSE.

Technology Self-Check

For development teams with the following technical experience, it will be of great help during the process of being listed on Cloud App. Teams lacking relevant experience can conduct technical pre-research in advance.

Terraform declaration syntax and deployment technology;

Cloud App uses Terraform grammar for resource declaration. The resource deployment engine is developed in-house.

If the software is based on a container, the following is required:

K8s container management technology.

K8s-based Helm Chart orchestration and delivery technology.

Commonly used IaaS/PaaS products on Tencent Cloud, including TKE, EKS, CVM, MySQL, Redis, COS, ES, TDMQ, CLS.

Preparing Developer Sub-User

Last updated：2025-04-23 14:25:52

Go to access management and quickly create a user.

Access methods: Select console access and programmatic access.

User permission: Grant preset permission policy for QcloudCloudappInstallationAccess.

If you want to use an existing sub-user, you can also grant the preset permission policy for QcloudCloudappInstallationAccess to the existing sub-user individually.

Note:

Console access and programmatic access, sub-users need to log in to the cloud application console.

Sub-user key is used to log in to the CLI developer tool and push the installation package.

Logging in to CloudApp Management Background

Last updated：2025-04-23 14:26:39

Logging In

After completing the Cloud App onboarding, the root account or authorized Sub-user can log in to cloudapp management backend to complete the production and management of installation packages.

Subsequent Operation

Perform production of the installation package.

Perform installation package management.

Producing an Installation Package

Last updated：2025-04-23 15:12:39

Terms and Descriptions

Term	Description
Installation Package	The installation package is a deliverable for automatic deployment submitted by partners that complies with CloudApp standards, integrating cloud resource orchestration, installation parameters, and output declarations, etc. Users through Cloud App one-click deployment, just automatically perform resource creation, application configuration and process startup, achieving full process automation installation.

Creating an Installation Package

Currently, the installation package supports two ways to create. As follows:

	Template Creation	Ordinary Creation
Advantage	Simple and quick, create using officially provided fixed templates. Just input a small amount of template parameters to quickly create an installation package and submit the development edition.	High flexibility, can create complex applications. Fully customize various types of resources and their parameters.
Shortage	Insufficient flexibility, only supports partial customization of parameters.	A certain level of development basis is required. You need to install the Node.js environment and the CLI developer tool.

The specific creation steps are as follows:

Template Quick Creation

Ordinary Creation

1. In Installation Package Management, click **Create Installation Package**, select **Created via Template**, and fill in the installation package name and some template parameters to complete the creation.

2. After the installation package is created, click **Version Management** under the operation column of the target application to enter the version management interface of the installation package.

1. In Installation Package Management, click **Create Installation Package**, fill in the installation package name, and select **Ordinary Creation** to complete the creation.

2. After the installation package is created, click **Version Management** under the operation column of the target application to enter the version management interface of the installation package.

Logging in to the Developer Tool (CLI)

1. In the version management interface, click **How to Upload Development Edition** to bring up the tool installation and login guide.
Note:
Node.js environment is required. Node.js environment is required. The CloudApp developer tool is developed based on Node.js. Please confirm that Node.js is installed in the development environment before installation. If not installed, please visit Node.js official website to download and install it.

2. After installing the developer tool as instructed, use the `cloudapp` command to view the instructions for using the tool.

3. After the tool is installed, please use the `cloudapp login` command to log in to the local developer center. You need to use the developer's CAM key to log in.
The SecretId and SecretKey are the corresponding CAM keys in the developer information submitted when registering the application developer entity. The developer and developer name will be displayed after a successful login. If you forget the key information, you can query it in the CAM console.

```
cloudapp login $SECRET_ID@$SECRET_KEY
```

Installation Package Initialization (CLI Tool)

Note:
The installation package is represented as a file system directory locally on the developer's machine. It is generally recommended to use the `.cloudapp` directory under the project root directory as the installation package directory.
Recommended to read Demo source code of the minimal installation package to understand the components of the installation package.
Use the `cloudapp init` command to generate a sample installation package:

```
# Initialize a local project. You can specify an initial template with -t. For
reference, supported templates are listed in the Github template list (default
template: CVM instance + public IP address).
cloudapp init hello-world

# Example
# cloudapp init hello-world -t CT_001
```

Note:

`cloudapp init` command is currently in preview edition. Its usage and parameters may be adjusted subsequently. Currently, `cloudapp init` only downloads template code locally. More template codes can be viewed in the Cloud App's GitHub repository.

Composition of Installation Package

The installation package consists of content delivered by the developer (divided into two modes: CVM deployment and container deployment). See the table below for specific differences:

CVM Deployment

Container Deployment

Note:

If your application uses container deployment, please switch to Container Deployment to view the content required for delivery.

Content	Position	Description
Meta-information of Installation Package	.cloudapp/package.yaml	The description file of the meta-information of the installation package. Contains declarations of information such as installation package ID, application installation parameters, and CAM permission policies.
Resource and variable declaration	.cloudapp/infrastructure/**/*.tf	Resource and variable declaration using Terraform syntax. Resources are declared in the form of resource blocks; variables are variables that can be used during resource deployment and need to be declared before use.
CVM image (artifact)	Cloud App Artifact Hosting Center	Cloud App provides CVM image hosting.

Note:

If your application uses CVM deployment, please switch to CVM Deployment to view the content required for delivery.

Content	Position	Description
Meta-information of Installation Package	.cloudapp/package.yaml	The description file of the meta-information of the installation package. Contains declarations of information such as installation package ID, application installation parameters, and TencentCloud API permissions.
Resource and variable declaration	.cloudapp/infrastructure/**.tf	Resource and variable declaration using Terraform syntax. Resources are declared in the form of resource blocks; variables are variables that can be used during resource deployment and need to be declared before use.
Container image (artifact)	Cloud App TCR repository	Cloud App provides TCR repositories to provide image hosting services for applications that need to host container images.
Service orchestration	.cloudapp/software/charts/**/*.yaml	Service orchestration, adopting Helm Chart orchestration specification. Orchestrate StatefulSet, Service, Job, Secret, etc. in a container cluster.

Meta-Information of Installation Package

The meta-information of the installation package is declared in `package.yaml`, including declarations of information such as installation package ID, application installation parameters, and TencentCloud API permissions.

```
# Installation package ID, corresponding to the app ID in the developer center
id: pkg-8lve2gn1

# Application parameters, defining the parameters that allow customers to set durin
args:
  - name: app_target
    label: target network
    widget: subnet-select

# Application permission declaration (optional). If you need to call TencentCloud A
role:
  policy:
    version: "2.0"
    statement:
      - action:
          - cloudapp:VerifyLicense
        resource: "*"
```

```
        effect: allow
```

Note:

Please provide the correct installation package ID. When the developer tool pushes the application, it will authenticate and push based on the installation package ID recorded in package.yaml. If the application is not registered, or the currently logged-in developer is not the corresponding application developer, the push will fail.

Resource and Variable Declaration

Resource Declaration

The resource declaration in the installation package is the core content of the installation package. Cloud applications use Terraform grammar to declare cloud resources for software dependencies.

```
resource "tencentcloud_eks_cluster" "eks" {
  availability_zone = var.app_target.subnet.zone
  k8s_version       = "1.20.6"
  vpc_id            = var.app_target.vpc.id
  subnet_ids        = [var.app_target.subnet.id]
  service_subnet_id = var.app_target.subnet.id
}
```

The above is the declaration syntax of a resource block. `resource` indicates a resource block declaration. `"tencentcloud_eks_cluster"` indicates the resource type. `"eks"` is the resource name.

Note:

The resources that can be declared supported by the installation package are roughly divided into two types:

Tencent Cloud IaaS / SaaS resources, prefixed with `tencentcloud` as the resource type.

Resources encapsulated by the Cloud App, with the resource prefix `cloudapp` .

Variable Declaration

The associated VPC, subnet and other information of the above-declared EKS resources are obtained from **variables**. In the installation package, all variables used need to be declared before use.

```
# Declare a variable using the variable keyword. app_target is the "variable name".
variable "app_target" {
  type = object({
    region    = string
    region_id = string
    vpc = object({
      id         = string
      cidr_block = string
    })
    subnet = object({
      id   = string
      zone = string
```

```
      })
    })
  }
```

Similar to a resource block, we use a variable block `variable` to declare a variable. In the above declaration, `app_target` is the variable name, which can be referred to as `var.app_target` in the resource block. Variables can be used to obtain users' installation parameters and can also be used to obtain the system's installation parameters. For details, see Using Installation Parameters, Variables and Output in the document description.

Image Hosting

CVM Image Hosting

Container Image Hosting

Note:

After you create a CVM image, you need to host the image in the Cloud Application Product Management Center so that during user installation, the Cloud App can automatically share the corresponding image with users.

1. In CloudApp developers, select **artifact management** of the application.

2. Select the CVM image that the current application needs to be hosted in the product management center.

3. Adjust the available regions of the mirror according to the regions where the application allows installation.

Note:

Cloud App provides TCR repositories to provide image hosting services for applications that need to host container images. Customers can pull from the private network when installing this repository, without the need for public network support.

1. Use the `cloudapp docker` command to obtain the repository login command.

2. Use `cloudapp docker --url` to obtain the repository push address.

3. Use `cloudapp docker --auth` to obtain the Docker authorization file separately, suitable for CI build scenarios.

4. After logging in to the Docker repository, you can push images to the hosted repository.

Valid period and permissions of the repository invoice

The repository invoice obtained by the cloudapp docker command can only be used for the application side to push software container images. The valid period of this invoice is 2 hours, so it should not be stored for use. It is recommended to obtain the invoice in real time through the cloudapp docker command after the CI process is built. The installation invoice used during customer installation is read-only, while the invoice obtained by cloudapp docker can operate the software images of the application developer.

Service Orchestration (Container Deployment)

Declare service orchestration by claiming resources of type `cloudapp_helm_app` .

```
resource "cloudapp_helm_app" "helm_charts" {
  cluster_id     = tencentcloud_eks_cluster.eks.id
  chart_src      = "../software/chart"
  chart_username = var.cloudapp_repo_username
  chart_password = var.cloudapp_repo_password

  chart_values = {
    SUBNET_ID = var.app_target.subnet.id
    IMAGE_CREDENTIALS = {
      REGISTRY = var.cloudapp_repo_server
      USERNAME = var.cloudapp_repo_username
      PASSWORD = var.cloudapp_repo_password
    }
  }
}
```

Requires specifying the target container cluster for orchestration as well as the location of the Helm Chart source code for orchestration (generally recommended to be placed at `.cloudapp/software/chart`).

Pass resource information to Helm Chart Values through `chart_values` . This information will be transmitted to Helm during deployment. Thus, infrastructure can be directly used during orchestration. See the above example.

Transmit the subnet ID to the orchestration. When orchestrate the service, specify this subnet ID to generate a private network CLB type. Another common example is to import the connection information of the instantiated DB into the orchestration, so that the workload can refer to it from the environmental variable or generate it into a configmap.

Note:

Transmitting the resource attributes declared in TF to Helm Chart Values is an important characteristic. Use this feature to directly orchestrate the actual deployment of hardware resources during software orchestration, without the need for secondary settings.

Advanced

Continue reading the follow-up content and learn the implementation of advanced installation package features.

Use installation parameters, variables and output

Provide HTTPS access through CLB

Call TencentCloud API

License integration

Resource

Go to Github repository to view more sample source code.

Next

After completing the production and uploading of the installation package, you need to push, verify, and list the installation package. For details, see Installation Package Management.

Using Installation Parameters, Variables and Output

Last updated：2025-04-23 14:27:48

Installation Parameters

Installation parameters can let customers install software in a suitable way. Common scenarios include:
Select and install in the specified Virtual Private Cloud (VPC) and subnet (Subnet) to implement network planning.
Select a customer security group (Security Group) to implement a network access policy.
Select the model to better fit the user's actual capacity.
Select the domain name and certificate for final deployment, so that the certificate can be automatically deployed.

Declare Installation Parameters in package.yaml

The installation package can declare installation parameters in package.yaml. The Cloud App will generate an appropriate installation interface for these declared installation parameters. The following example comes from the Cloud App template CT_001_cvm-ip.

```
# <package.yaml>
# Custom installation parameters, declare
args:
  - name: app_target
    label: target network
    widget: subnet-select
  - name: cvm_type
    label: Cloud Virtual Machine (CVM) ty
    widget: cvm-instance-type-select
    region: ${app_target.region}
    zone: ${app_target.subnet.zone}
  - name: sg
    label: security group
    widget: security-group-select
    regions:
      - ${app_target.region}
```

Fields used in declaring installation parameters in package.yaml are explained as follows:

Field	Description
args	Declare a parameter list with args. Each item is a parameter. The installation interface will render installation components in the declared order.
args[i].name	Parameter identification. This is also the variable name to be declared and used in Terraform subsequently.
args[i].label	Parameter name. The installation interface will show the parameter name.
args[i].widget	The installation components for parameter use. For supported components, see Installation Parameter Manual.
args[i].region args[i].zone ...	The configuration of installation components for parameters. Different component types have different configuration fields. See Installation Parameter Manual for details. The configuration of components generally provides a finer selection range when users enter data. For example, after the user has selected the target network, the server type can be limited to the example type range supported by the target subnet to avoid invalid selections.
args[i].when	(Advanced usage) Conditions for the occurrence of parameters. Parameters do not appear in the installation interface by default and only appear when the conditions of when are met. Applicable to the form linkage of installation components.
args[i].hidden	(Advanced usage) Conditions for hiding parameters. Parameters will appear in the installation interface by default and hide when the conditions of when are met. Applicable to the form linkage of installation components.
args[i].validator	(Advanced usage) Verification rules for parameters can help users reduce configuration errors.

Using Installation Parameters in a tf File

After defining the installation parameters in the package.yaml file, you can declare and use them in the tf file. The following example comes from CT_001_cvm-ip.

```
# <variable.tf>
# CVM model selection variable
variable "cvm_type" {
  type = object({
    region        = string
    region_id     = string
    zone          = string
    instance_type = string
  })
}
```

Different types of installation parameters have different variable value types. See Installation Parameter Manual for details.
After declaration, it can be used as an ordinary Terraform variable.

```
# <deployment.tf>
resource "tencentcloud_instance" "demo_cvm" {
  # CVM model
  instance_type = var.cvm_type.instance_type
}
```

Variable

Declare a Variable

All variables used in the installation package need to be declared before use.

```
# <variable.tf>
# CVM model selection variable
variable "cvm_type" {
  type = object({
    region        = string
    region_id     = string
    zone          = string
    instance_type = string
  })
}
```

Similar to a resource block, use a variable block `variable` to declare a variable. In the above declaration, `cvm_type` is the variable name, and `var.cvm_type` can be used to refer to it in the resource block.

Variable Separation Practice

Using variables can effectively manage orchestration parameters of resources. For example, if a service provider has two components with basically the same orchestration architectures but different server images, the image ID can be extracted as a variable.
In practice, it is recommended to separate variables and orchestration parts in the infrastructure directory of the installation package according to variable.tf and deployment.tf. See CloudApp Template Library for more practical ideas.

Variable Type

The available variables in the installation package are divided into two kinds: system variables and application variables.

System variable: Used to provide deployment requirements information, such as image repository address and credentials, CAM role information, etc. It is prefixed with `cloudapp_` and can be used by ALL installation packages.
Parameter variable: The application parameters defined in `package.yaml` will generate corresponding application variables. The variable name is exactly the parameter name.
Common variable: A variable declared and used by the application itself, such as CVM image ID, basic database settings, etc.
For a list and information about system variables, please see System Variable Manual. For the method of application parameter configuration, please see Installation Parameter Manual.

Variables used must be declared

Even if the values of system variables and application variables are automatically injected, they must be declared when using the .TF file. Variables used without declaration can cause the installation package to fail to package.

Output

The results after application installation need to be provided to the customer. For example, the access address of the application, the address of the management system, or the initial password of the management system, etc. This can be achieved by declaring output in the tf file.

```
output "main" {
  value = "https://my.${var.selected_domain}"
  # value = "http://${tencentcloud_instance.cloudapp_cvm.public_ip}:8080"
  description = "access entrance"
}
output "a_admin_url" {
  value = "https://admin.${var.selected_domain}"
  description = "operation system"
}
output "b_admin_default_user" {
  value = "admin"
  description = "default admin user"
}
output "c_admin_default_password" {
  value = random_password.default_admin.result
  description = "default administrator password"
  sensitive = true
}
```

Note:

Recommendation to increase the output with the variable name main and set the value to URL. This value will be used as the default open address for the customer application.

If sensitive = true is declared, it indicates sensitive information (customers can perform CAM or MFA security restrictions against sensitive information APIs).
The display order of the final result items is not based on the declaration sequence in the tf file, but sorted alphabetically according to the output variable names.
The declared output. Customers can see the corresponding installation result in the dashboard app details.

Providing HTTPS Access through CLB

Last updated：2025-04-23 14:28:38

Currently, most applications provide HTTP access capability. If you want to help customers directly integrate HTTPS certificates and domain name bindings, see this document.

Declare Installation Parameters

Declare the installation parameters in package.yaml as follows:

```
args:
  - name: app_certification
    label: SSL Certificates
    widget: certificate-select
    optional: true
  - name: app_domain
    label: domain name
    widget: domain-input
    optional: true
    certId: ${app_certification.certId}
```

Two parameters are declared through certificate-select and domain-input, which respectively receive the customer's certificate and domain name input. The domain name component binds the certificate ID to help users select or enter a domain name compatible with the certificate.

Declare a Resource Stack

Add two variables in variable.tf to receive installation parameters:

```
variable "app_certification" {
   type = object({
     certId = string
   })
}

variable "app_domain" {
   type = object({
     domain = string
   })
}
```

Declare instances, listeners and corresponding forwarding rules of CLB in deployment.tf:

```
# Declare a CLB instance
resource "tencentcloud_clb_instance" "open_clb" {
  # Network type of the load balancing instance. OPEN: public network. INTERNAL: pr
  network_type = "OPEN"
  # Security group
  security_groups = [var.sg.security_group.id]
  # VPC
  vpc_id = var.app_target.vpc.id
  # Subnet
  subnet_id = var.app_target.subnet.id
  # Enable default pass-through, that is, the Target allows traffic from the CLB.
  load_balancer_pass_to_target = true
}

# Declare a CLB HTTPS listener
resource "tencentcloud_clb_listener" "https_listener" {
  clb_id        = tencentcloud_clb_instance.open_clb.id
  listener_name = "https_listener"
  # CLB listens on port 443, support HTTPS access
  port          = 443
  protocol      = "HTTPS"
  # Use the certificate declared in the installation parameters here
  certificate_id       = var.app_certification.certId
  certificate_ssl_mode = "UNIDIRECTIONAL"
}

# Declare a CLB forwarding rule
resource "tencentcloud_clb_listener_rule" "https_rule" {
  clb_id      = tencentcloud_clb_instance.open_clb.id
  listener_id = tencentcloud_clb_listener.https_listener.id
  # Configure this as the domain name declared in the installation parameters. It m
  domain = var.app_domain.domain
  # Which path to forward. Here, configure the root path for site-wide forwarding.
  url = "/"
}

# Declare a CLB backend service binding to forward the traffic that hits the forwar
resource "tencentcloud_clb_attachment" "clb_backend" {
  clb_id      = tencentcloud_clb_instance.open_clb.id
  listener_id = tencentcloud_clb_listener.https_listener.id
  rule_id     = tencentcloud_clb_listener_rule.api_https_rule.id

  targets {
    # CVM instance ID (replace with the actual instance ID)
```

```
    instance_id = tencentcloud_instance.demo_cvm[0].id
    # Assume our CVM has an open http port 3000.
    port = 3000
    # weight
    weight = 100
  }
}
```

Principle

The product design of CLB is used as load balancing, but also supports being used as certificate offloading. It provides customers with HTTPS port, then forwards to the HTTP port of the software in the backend.

Complete Example

For a complete example, you can access Github to obtain.

Other Solutions

Besides CLB, TKE's layer-7 Service, CDN, EdgeOne and other acceleration products also support certificate deployment. Subsequently, Cloud App will provide more integration samples for reference.

Calling TencentCloud API

Last updated：2025-04-23 14:29:27

Overview

CloudApp provides a solution that allows application processes to call TencentCloud API in a secure and controlled way, thereby accessing or operating the customer's cloud resources. Commonly used scenarios include:

Cloud management software

Call public cloud SaaS to implement software features (such as SMS)

Automatic scaling

Verify software authorization

Directions

Claim a Permission Policy

Declare a permission policy in package.yaml to restrict the range of TencentCloud APIs that can be called by the application. When installing the installation package, the cloud application will generate a CAM policy based on the declared permission policy. Therefore, APIs not included in the declared permission policy have no invoke permission within the process.

```
# package.yaml
role:
  policy:
    version: "2.0"
    statement:
      - action:
          - cloudapp:VerifyLicense
          - cvm:DescribeInstances
        resource: "*"
        effect: allow
```

Claim a Role Used Variable

Declare a role used variable in variable.tf:

```
# variable.tf
variable "cloudapp_cam_role" {}
```

Inject the Assigned Role Name Into the Process

Cloud applications will provide the runtime role generated during application installation to the installation package via Terraform variables. The installation package needs to inject the role name variable `cloudapp_cam_role` into the environment variables or configuration files when declaring cloud resources. The following are examples of CVM and containers.

CVM Example

Container Example

Method 1: Inject Into Environment Variables

You can add environment variables in the user script declared on the CVM.

```
# deployment.tf
resource "tencentcloud_instance" "demo_cvm" {
  # Note: You need to bind the CAM role to the runtime role of the CVM instance.
  cam_role_name = var.cloudapp_cam_role
  user_data_raw = <<-EOT
#!/bin/bash
# Export the role name to the environment variable
export CLOUDAPP_CAM_ROLE=${var.cloudapp_cam_role}
# The process starting can read the role name from the environment variable.
node main.js
  EOT
}
```

Method 2: Inject Into the Configuration File

Alternatively, you can inject the role name into the configuration file.

```
# deployment.tf
resource "tencentcloud_instance" "demo_cvm" {
  # Note: You need to bind the CAM role to the runtime role of the CVM instance.
  cam_role_name = var.cloudapp_cam_role
  user_data_raw = <<-EOT
#!/bin/bash
# Export the role name to the configuration file
echo "${var.cloudapp_cam_role}" >> /usr/local/.cloudapp_cam_role
# The process starting can read the role name from the configuration file /usr/loca
node main.js
  EOT
}
```

First: Import the Variable As a Chart Variable

First, in Terraform, declare the runtime role bound to the Worker's corresponding CVM. Meanwhile, pass the variable to Helm's Chart Values:

```
# <deployment.tf>
resource "tencentcloud_kubernetes_cluster" "tke-cluster1" {
  worker_config {
    # Other worker_config content is omitted here.
    cam_role_name = var.cloudapp_cam_role
  }
}
resource "cloudapp_helm_app" "helm_charts" {
  chart_values = {
    CAM_ROLE = var.cloudapp_cam_role # Inject the role name variable here.
  }
}
```

Method 1: Use the Chart Variable As a StatefulSet Environment Variable

Refer to the Chart variable in the workload in Helm Chart:

```
# <values.yaml>
# CAM role name, which is used to obtain a temporary key for calling the TencentClo
CAM_ROLE: ""

# <templates/statefulset.yaml>
kind: StatefulSet
spec:
    spec:
      containers:
        - name: "my-container"
          image: {{ quote .Values.SERVER_IMAGE }}
          env:
            # Assign CAM_ROLE to the container's environment variable
            - name: CAM_ROLE
              value: {{ quote .Values.CAM_ROLE }}
```

After the container starts, the environment variable CAM_ROLE will contain the role name.

Method 2: Write the Variable Into ConfigMap and Use It in StatefulSet

Declare a Configmap in Helm Chart and refer to it in the workload

```
# <values.yaml>
# CAM role name, which is used to obtain a temporary key for calling the TencentClo
CAM_ROLE: ""

# <configmap.yaml>
```

```
# Declare ConfigMap
kind: ConfigMap
metadata:
  name: cloudapp-config
data:
  .cloudapp_cam_role: {{ quote .Values.CAM_ROLE }}

# <templates/statefulset.yaml>
kind: StatefulSet
spec:
  template:
    spec:
      containers:
       - name: "my-container"
         image: {{ quote .Values.SERVER_IMAGE }}
         valueMounts:
         - name: cloudapp-cam-role
           mountPath: /usr/local/cloudapp
           subPath: .cloudapp_cam_role
      volumes:
      - name: cloudapp-cam-role
        configMap:
          name: cloudapp-config
```

After the container starts, the process can read the CAM role from `/usr/local/cloudapp/.cloudapp_cam_role` .

Get a Temporary Key for a Role

After obtaining the role name in the process, we can obtain the temporary key of the role through an HTTP request.

The CURL example of the request is as follows:

```
curl http://metadata.tencentyun.com/meta-data/cam/security-
credentials/$CLOUDAPP_CAM_ROLE
```

The response results are as follows:

```
{
  "TmpSecretId": "(RESPONSED SECRET ID)",
  "TmpSecretKey": "(RESPONSED SECRET KEY)",
  "ExpiredTime": 1658866289,
  "Expiration": "2022-07-26T20:11:29Z",
  "Token": "(RESPONSED TOKEN)",
  "Code": "Success"
}
```

5. Calling TencentCloud API Using Temporary Key

Use the above temporary key to call the cloud API. The authentication credential for calling the cloud API must contain: SecretId, SecretKey, Token.

For the calling methods in different languages, please refer to the corresponding SDK documentation:

PHP

Python

Java

Go

.NET

Node.js

C++

Defining Application Permission Policies

Last updated：2025-04-23 14:30:12

Overview

Cloud applications support software processes in runtime roles calling TencentCloud API. By defining the permission policy for this role, you can finely control the cloud permissions owned by the process, meeting customers' requirements for permission control. Typical scenario:
Verify the status of the software LICENSE.
Access COS buckets in application, with access limited to specified buckets (such as the bucket selected during user installation).
Perform inspections on running resources in application and restart faulty nodes automatically.

Solution Description

Define the permission policy syntax in the `package.yaml` file, and you can control the application's permission scope.

Declaration Method

Declare the permission policy in the `package.yaml` file through `role.policy`. The syntax is consistent with the CAM permission policy. For details, see CAM Permission Policy Syntax Structure. The following is an example:

```
args:
  - name: app_cos
    label: select bucket
    widget: cos-bucket-select

role:
  policy:
    version: "2.0"
    statement:
      # Permission 1: support LICENSE verification
      - action:
          - cloudapp:VerifyLicense
        resource: "*"
        effect: allow

      # Permission 2: Support restarting CVM under the application instance tag
      - action:
```

```
          - cvm:RebootInstances
        resource: "*"
        condition:
          "for_any_value:string_equal":
             "qcs:tag":
                - "CloudappId&${var.cloudapp_id}"
        effect: "allow"

      # Permission 3: Support access to the selected COS storage bucket during inst
      - action:
          - "cos:*"
        resource:
          - "qcs::cos:${var.app_cos.region}:uid/${var.app_cos.app_id}:${var.app_cos
          - "qcs::cos:${var.app_cos.region}:uid/${var.app_cos.app_id}:${var.app_cos
        effect: allow
```

Note:

Declare permission policies through `role.policy` following the WYSIWYG principle. All privileges that a runtime role has need to be explicitly declared in `package.yaml` .

Use Variables in Policy Syntax

Supports the use of two types of variables in policy: system variable and installation parameters. In the above example:

In the example of permission 2, the `var.cloudapp_id` system variable is used to implement tag authorization.

In the example of permission 3, the `var.app_cos` installation parameter is used to achieve resource-level authorization.

Upgrade to the New Policy Syntax

Note:

The new permission policy generation solution will have forward compatibility. Permissions originally declared through `scopes.cloudAPI` will still take effect.

The new permission declaration syntax provides more granular permission control. Strongly recommend upgrading to the new declaration syntax and declare permission policies through `role.policy` .

The original version declares the permissions of the runtime role through `scopes.cloudAPI` . An example is as follows:

```
scopes:
  cloudAPI:
    - cvm:DescribeInstances
```

The above declaration grammar after upgrading is as follows:

```
role:
```

```
policy:
  version: "2.0"
  statement:
    - action:
        - cvm:DescribeInstances
        - cloudapp:VerifyLicense
      resource: "*"
      effect: allow
```

Need to pay attention during the upgrade process:

Older versions add the `cloudapp:VerifyLicense` API to the permission policy by default without the need to explicitly declare it. **In the new version, the permissions of the runtime role contain only explicitly declared APIs. You need to explicitly declare `role.policy` the `cloudapp:VerifyLicense`** , otherwise the application will be unable to integrate with License.

After using the new syntax, the original `scopes.cloudAPI` field needs to be deleted.

Complete Example

You can refer to the demonstration instructions of Custom Application Permission Policy.

Integrating License

Last updated：2025-04-23 15:51:46

For software installed via Cloud App, Cloud App will issue a License. Within the software process, the License can be read and verified.

What Are the Functions of a License?

A License is a permit for software end users. The authorization content includes:
Usage period: If the software expires, the software process can deny service or provide downgrade service.
Specifications: Software can run based on authorized specifications, such as limiting the number of concurrencies and the number of users.

Integrating Cloud App License: What Are the Advantages?

Smooth: Licenses are automatically issued upon placing an order without waiting.
flexible: Support needs such as renewal and specification adjustment.
Trustworthy: Issued by Tencent Cloud official, associated with a License bound to a role-based instance, difficult to forge and copy.

License Transaction and Delivery

Service providers need to define specifications for products corresponding to application packages. The defined specifications will be confirmed when users make purchases and serve as the specifications of the License.
The License is issued according to the software specifications purchased by the customer. The issued License contains the following information:
associated software package
associated software instance
authorized specification information
Authorization mode (permanent/subscription)
authorization status (valid/expired/invalid)
Issue time Activation time Expiration time Expiration time

Verify License

The application image of the service provider can verify the License through the TencentCloud API. The process is as follows.

To verify the software License, the core steps are:
Retrieve installation parameters `cloudapp_cam_role`, use CAM role to apply for temporary keys, and make an API call to the cloud API. See the document for details on calling the cloud API.
Call the cloudapp.VerifyLicense API to retrieve the license information associated with the current process.
Verify the License status of the software process as needed, or perform feature control based on specifications.
The License information is shown as follows:


```
{
    "Response": {
        "RequestId": "9026a0fe-992e-4015-a5d5-95a322333853",
        "License": {
            "ActivationDate": "2024-12-10T09:46:58+08:00",
            "AuthorizedCloudappId": "cloudapp-2gwvcfb1",
            "AuthorizedCloudappRoleId": "0",
            "AuthorizedSpecification": [
                {
                    "ParamKey": "version",
                    "ParamKeyName": "Version"
                    "ParamValue": "basic",
                    "ParamValueName": "basic version"
                },
                {
                    "ParamKey": "scale",
                    "ParamKeyName": "Specification"
                    "ParamValue": "single",
                    "ParamValueName": "single thread"
                }
            ],
            "AuthorizedUserUin": "100000888888",
            "BillingMode": 1,
            "ExpirationDate": "2025-01-10T09:46:58+08:00",
            "IssueDate": "2024-12-10T09:46:46+08:00",
            "LicenseId": "100000888888:pkg-la39nlb7:cloudapp-2gwvcfb1:6006",
            "LicenseMode": "Subscription",
            "LicenseStatus": "Active",
            "LicenseType": "Standard",
            "LifeSpan": 1,
```

```
            "LifeSpanUnit": "M",
            "ProviderId": 100000099,
            "SoftwarePackageId": "pkg-la39nlb7"
        },
        "Signature": "T2mbjUr8Q8I0nekShP7WJs7MvUq8Ovv0lrWkn5PEIPVlFYWpaRJOWBviQstCU
        "Timestamp": "2025-01-09T10:28:57+08:00"
    }
}
```


Fields that need attention here:

LicenseStatus: Authorization status. If it is Active, it indicates that the authorization is valid; if it is Deactive, it indicates that the authorization has expired. The service provider should use the LicenseStatus field to determine whether the authorization is valid.

AuthorizedSpecification: Authorized specification. A software process can control software behavior based on this specification.

LicenseMode: Authorization mode, subscription or permanent buyout.

Signature: Signature for the Response.License structure. For scenarios with high security requirements for the License, signature verification can be performed (see License Signature Verification Method for specific practices) to ensure that the returned License information hasn't been intercepted and tampered with by a man-in-the-middle.

The timing of verifying the License can be autonomously determined by the software process. Common ways include:

Scheduled verification, such as once a day or once an hour.

Verify as needed, for example, when calling product features that require License verification.

Development Debugging of License

Add the `specs` specification configuration in the `package.yaml` of the application package in the following format. Then you can select specification information and authorization mode to create a License when installing the development edition of the application.

```
# <package.yaml>
# Declare application package specifications
specs:
  - key: version
    name: Version
    enum:
      - value: standard
        name: standard version
      - value: advanced
        name: advanced edition
      - value: enterprise
        name: Enterprise Edition
  - key: cluster_mode
```

```
    name: cluster mode
    enum:
      - value: single
        name: Single Cluster
      - value: double
        name: Dual Cluster
      - value: triple
        name: Three Clusters
```

Verify License Signature

The signature is implemented using the RSA-PSS-SHA256 algorithm. Please use the following public key for signature verification:

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA5EQJv0v0f1hrB7NIGwXi
hFmw+ugrJi7gnmOqaiYp7oFrzf4RSJ3DPr4K01F+CrjTdCPghDLq4fsKVxxHAjNj
nbstbVlHZEVOfzQ4umeocJpxWFuyKyGwHv+obnEZ/4689fxVpTbG3IbUTGn1TRJs
9s3xM8nFd6LLAoh1Hhrdf2D4mLRToLvtRVat1l8fH3gsM+RoG4L4h+3hghn4bpyA
na2MBFDzvmBeVGUVzqRjSvUaexd+Bo1wTsllAdqjP6MTlAAWGmIAMStBSRS+YpRQ
xjhE9Rdb9zTE54q3Ui7UJg5BMe+R3kVrBINbnT6Va8/Lzjg4+THdpMTLr6fY6ObF
7r+i/924XgxqQOFvGaFJSyjXTORnK42T5YRr5TSqxr9CzhybPcdRvws2GdAq9f55
8whj1DYcgg0X8kR06Iu+/9Mk/CqssdrZ8LYDwSkDI8S/RwpdNQfifUa8wyY0R2xN
nY+bnkrjvGPz7Rokr0Ki9/orT9i4yQWA1mMCDi2vcP+oXqrEs7XAyH85gDSzuTp+
dXbTYPZpIAK6Kejwssw1IE1lGNP4PNQZk9EXU7+vB1csz4GUao7Mr7F5VbrGKvTs
aGxbIc6b0MDWMEFA7L/CWC9UtReWCk1MYwJzy105bWU/VBpYJPmyZTFRQaY2MEH4
fnsK2+jtZ1IYIQw/YsHU6CcCAwEAAQ==
-----END PUBLIC KEY-----
```

Signature verification involves the following two fields:

Response.Signature: The signature is generated by first compressing the Response.License JSON structure to remove line breaks and then performing an RSA-PSS-SHA256 signature.

Response.License: The signed JSON structure. During verification, ensure that the structure is JSON data compressed and with line breaks removed.

Example code for signature verification is currently available in the following Golang version. Examples in other languages will be provided subsequently.

Go Lang

```
// VerifySignWithSha256RSA verifies the signature using an RSA public key
// @Param   publicKey PKIX RSA public key in PEM format
// @Param   signature base64 encoded signature
// @Param   message Content to be verified by signature
// @Return  error Return nil if signature verification passes, otherwise return err
```

```
func VerifySignWithSha256RSA(publicKey string, signature string, message []byte) er
    // Parse public key
    publicKeyBlock, _ := pem.Decode([]byte(publicKey)) // Read PEM format public ke
    // Decode PKIX format RSA public key
    rsaPublicKeyPKIX, err := x509.ParsePKIXPublicKey(publicKeyBlock.Bytes)
    if err != nil {
       return err
    }
    rsaPublicKey, ok := rsaPublicKeyPKIX.(*rsa.PublicKey)
    if !ok {
       return fmt.Errorf("public key is not rsa public key")
    }
    // Calculate the SHA-256 hash of the message
    hash := sha256.New()
    hash.Write(message)
    hashedMessage := hash.Sum(nil)
    // Obtain signature
    signatureFromBase64, err := base64.StdEncoding.DecodeString(signature)
    if err != nil {
       return err
    }
    // Verify signature with public key
    return rsa.VerifyPSS(rsaPublicKey, crypto.SHA256, hashedMessage, signatureFromB
}

// Usage example
const publicKeyPkix = "-----BEGIN PUBLIC KEY-----\\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8

func TestVerifySignWithSha256RSA(t *testing.T) {
    type args struct {
       publicKey string
       signature string
       message   []byte
    }
    tests := []struct {
       name    string
       args    args
       wantErr bool
    }{
       {
          name:    "Verify signature"
          wantErr: false,
          args: args{
             publicKey: publicKeyPkix,
             signature: "T2mbjUr8Q8I0nekShP7WJs7MvUq8Ovv0lrWkn5PEIPVlFYWpaRJOWBviQs
             message:   []byte("{\\"ActivationDate\\":\\"2024-12-10T09:46:58+08:00\
          },
```

```
        },
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            if err := VerifySignWithSha256RSA(tt.args.publicKey, tt.args.signature, t
                t.Errorf("VerifySignWithSha256RSA() error = %v, wantErr %v", err, tt.w
            }
        })
    }
}
```

Common Issues About License

Is Integration of the License Required? Is It Possible to Generate and Verify the License by Oneself?

It is required. After integration, the software billing of users can be well integrated with authorization online. For example:
Free trial to paid conversion is merely a License update. Users do not need to update the instance and can retain data.
Renewal scenarios will automatically refresh the License, and there is no need to go to the service provider for offline updates.
In scaling scenarios, the License will also be automatically refreshed, and the instance does not need to be replaced.
If the software itself is free software and no need for verification of software authorization, then there is no need to pay attention to the access of the License.

Can the Service Provider Perceive the Issuance of the License?

Temporarily unable to perceive.

How to Store the Public Key for Signature Verification?

The public key is used to verify the signature of the License information to ensure that the License information in VerifyLicense is issued by Tencent Cloud and has not been tampered with by a man-in-the-middle. When verifying the License signature, the supplier should ensure that the loaded public key is the one provided in the Verify License Signature method; otherwise, it may lead to the License information being attacked and tampered with by a man-in-the-middle. Therefore, the public key should not be exposed in configuration files that are easily modified or other similar independent configurations. If it needs to be saved offline in the software client, at least the public key configuration should be ensured in the client executable file that has been obfuscated or hardened.

Installation Package Management

Last updated：2025-04-23 14:49:14

After the production of the installation package, the application developer can perform a series of management actions on the installation package, including push, test, version management, and allowlist control.

Installation Package Push

Once the installation preparation is completed, the application developer can use `cloudapp push` to push the installation package to the CloudApp repository.

Installation Test

After the installation package is pushed, the terminal will output a shortcut installation link. The developer can directly enter the application's installation interface from the link to conduct installation tests. Or select to proceed to install from the **Version Management** > **Development Version** interface of Installation Package Management. The steps are as shown below:

The installation interface matches the customer's final installation interface. Herein, you can test the installation workflow of the installation package. In case of installation failure, the developer can see the error information in the error bubble of the resource status or in the installation log to troubleshoot faster. Please refer to Installation FAQ.

Version Management

Application developers can configure different versions for the installation package. The published version of an application must sequentially go through the development edition, trial version, until the final online version. The introductions of each version are in the table below:

Version	Quantity	Purpose	Access Scope
Development Edition	Each developer under each	Developer verifies self-test.	Only developers themselves can install.

	application is limited to 1.		
Trial version	Only 1 per application.	Team experience, test.	Application experience allowlist.
Online version	Only 1 per application.	For customers to install officially.	If the online allowlist is not enabled, all customers can install; if the online allowlist is enabled, only whitelist customers can install.

Developers can set the development edition as the trial version. After configuration, accounts within the trial allowlist can install the trial version. When setting the trial version, a version number needs to be specified. After the trial version is officially shelved, it will serve as the version number of the official version. Only the trial version can be shelved as the online edition.

After preparing the version description materials, click **Publish** in the trial edition to complete the publication of the application.

Note:

The approval workflow for the current application version's release is not set yet. So once the trial version is clicked **Publish**, it is installable for all customers by default. It is recommended that the application developer first set the accessible customer accounts via the allowlist control method below.

Allowlist Control

We support developers in allowlist control of the installation package, including the trial version allowlist and the online version allowlist.

Note:

Trial version allowlist restrictions limit the accessible accounts of the trial version. It is advisable to use it to add application-related product research and development and test teams.

Details page access/installation permission settings control the access accounts of the online edition, suitable for scenarios where the software is delivered to a specific customer group or trialed on a small scale.

In application information, you can set the trial allowlist and the details page allowlist.

Application Access

Last updated：2025-04-23 14:57:28

For listed applications, we provide the following application access-related supported features for application developers:

Function	Description
Application API	Application developers can expose the features of software via TencentCloud API in a standard way, thereby enabling customers to manage and use the software by calling TencentCloud API. The standard TencentCloud API has built-in CAM authentication capability, so the software does not need to build its own API-level authentication.
Usage interface	The implementation of the software usage interface of the application developer can be a self-built public network HTTPS site or a software management interface integrated into the Tencent Cloud console.
Access to cloud resources	Application developers can pre-declare the APIs of underlying Tencent Cloud resources that the software needs to call. When installing software, the Cloud App will automatically generate a role used for the software. When using software, the Cloud App automatically applies for a temporary key of the role for the software to access APIs of these cloud resources, thereby achieving management of underlying cloud resources, such as built-in monitoring policies, capacity expansion ability, etc.

The above access-related configurations can all be completed in the application package production phase.

Installation FAQ

Last updated：2025-04-23 14:23:19

1. Helm App Installed Successfully, but Why Can'T the Pod Pull the Image in the Workload?

Deploying Cloud Applications to regions such as Beijing, Shanghai, Guangzhou, Nanjing, and Chengdu allows container images to be pulled through the VPC. For other regions, the application repository is accessed via the public network. Therefore, the target VPC for software installation must have public network access capability. If the customer's VPC lacks a public egress IP address, you can guide them to temporarily add a NAT Gateway and the corresponding routing configuration as a solution. For specific operations, refer to the NAT Gateway documentation.

2. Workloads Are Up, but Why Does the Helm Application Prompt an Installation Failure?

If there is a post_install script, it might be that the script has timed out. Please check whether there are processes that have not exited in the post_install job.

3. the Image Architecture Is Inconsistent with the Node Cvm Architecture, Causing a Startup Failure. the Error Information Is "Exec User Process Caused: Exec Format Error". How to Handle?

When building an image, select a platform. If it is TKE, generally select `linux/amd64`. For details, see Documentation.

4. Installation Failure Due to Insufficient Resource Inventory. How to Resolve?

You can replace the availability zone or submit a ticket for assistance.