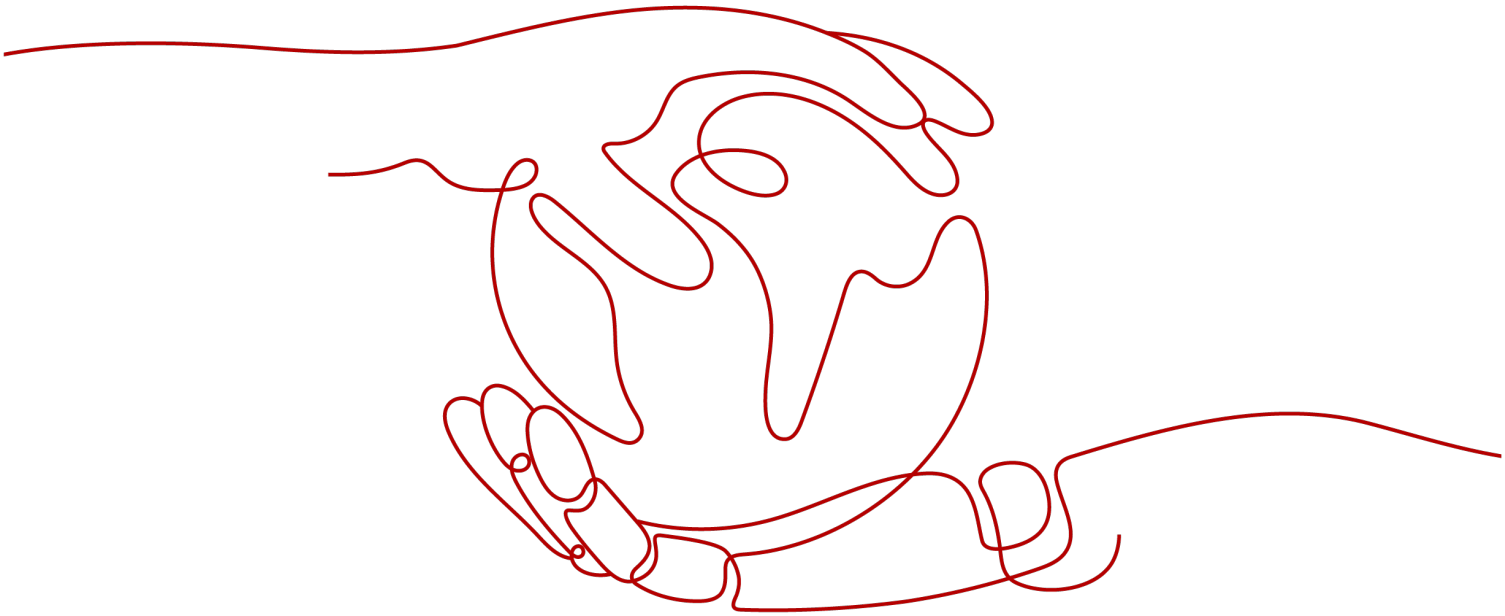
云容器实例

SDK 参考

文档版本 01

发布日期 2024-01-16



华为技术有限公司

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 使用 client-go 访问 CCI

本节将介绍如何将CCI认证工具cci-iam-authenticator与client-go结合使用以调用API。

安装 cci-iam-authenticator

请参考**使用kubectl**，下载安装及设置cci-iam-authenticator。

安装 kubernetes client-go

详情请参考**Installing client-go** 。

说明

当前**CCI服务开放的API对应的Kubernetes版本为1.19**，根据**Compatibility: client-go <-> Kubernetes clusters**，推荐使用的SDK版本为k8s.io/client-go@kubernetes-1.19.0。

使用 Go SDK

说明

示例已通过以下版本的测试：

1. k8s.io/client-go@kubernetes-1.15.0
2. k8s.io/client-go@kubernetes-1.16.0
3. k8s.io/client-go@kubernetes-1.17.0
4. k8s.io/client-go@kubernetes-1.18.0
5. k8s.io/client-go@kubernetes-1.19.0
6. k8s.io/client-go@kubernetes-1.20.0

使用用户名/密码进行认证

```
import (
    "fmt"

    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/tools/clientcmd"
    "k8s.io/client-go/tools/clientcmd/api"
)

const (
    apiVersion = "client.authentication.k8s.io/v1beta1"
    // 云容器实例 CCI，参考https://developer.huaweicloud.com/endpoint
```

```
    cciEndpoint = "<例如华北-北京四: https://cci.cn-north-4.myhuaweicloud.com>"
    // 统一身份认证服务 IAM，参考https://developer.huaweicloud.com/endpoint

    iamEndpoint = "<例如华北-北京四: https://iam.cn-north-4.myhuaweicloud.com>"
    // 地区和终端节点，参考https://developer.huaweicloud.com/endpoint

        projectName = "<例如华北-北京四: cn-north-4>"
)

// userName, domainName, password分别表示用户名、账号名及密码，需用户自行传入，因用户名、账号名及
密码用于认证，存在安全风险，请用户注意使用安全；
var userName, domainName, password string

// NewClient 通过username/password创建Clientset
func NewClient() (*kubernetes.Clientset, error) {
    config, err := clientcmd.BuildConfigFromFlags(cciEndpoint, "")
    if err != nil {
        return nil, err
    }
    var optionArgs []string
    optionArgs = append(optionArgs, fmt.Sprintf("--iam-endpoint=%s", iamEndpoint))
    optionArgs = append(optionArgs, fmt.Sprintf("--project-name=%s", projectName))
    optionArgs = append(optionArgs, fmt.Sprintf("--token-only=false"))
    optionArgs = append(optionArgs, fmt.Sprintf("--domain-name=%s", domainName))
    optionArgs = append(optionArgs, fmt.Sprintf("--user-name=%s", userName))
    optionArgs = append(optionArgs, fmt.Sprintf("--password=%s", password))

    config.ExecProvider = &api.ExecConfig{
        Command:    "cci-iam-authenticator",
        APIVersion: apiVersion,
        Args:       append([]string{"token"}, optionArgs...),
        Env:        make([]api.ExecEnvVar, 0),
    }
    return kubernetes.NewForConfig(config)
}
```

使用AK/SK进行认证

```
import (
    "fmt"
    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/tools/clientcmd"
    "k8s.io/client-go/tools/clientcmd/api"
)

const (
    apiVersion = "client.authentication.k8s.io/v1beta1"
    // 云容器实例 CCI，参考https://developer.huaweicloud.com/endpoint

    cciEndpoint = "<例如华北-北京四: https://cci.cn-north-4.myhuaweicloud.com>"
    // 统一身份认证服务 IAM，参考https://developer.huaweicloud.com/endpoint

    iamEndpoint = "<例如华北-北京四: https://iam.cn-north-4.myhuaweicloud.com>"
    // 地区和终端节点，参考https://developer.huaweicloud.com/endpoint

    projectName = "<例如华北-北京四: 'cn-north-4'>"
)

// 获取AK/SK参考: https://support.huaweicloud.com/devg-cci/
cci_kubectl_01.html#cci_kubectl_01__section17023744719
// ak，sk分别表示用户AK/SK，需用户自行传入，因AK/SK用于认证，存在安全风险，请用户注意使用安全。
var ak, sk string

// NewClient 通过AK/SK认证创建Clientset
func NewClient() (*kubernetes.Clientset, error) {
    config, err := clientcmd.BuildConfigFromFlags(cciEndpoint, "")
    if err != nil {
        return nil, err
    }
```

```
    var optionArgs []string
    optionArgs = append(optionArgs, fmt.Sprintf("--iam-endpoint=%s", iamEndpoint))
    optionArgs = append(optionArgs, fmt.Sprintf("--project-name=%s", projectName))
    optionArgs = append(optionArgs, fmt.Sprintf("--token-only=false"))
    optionArgs = append(optionArgs, fmt.Sprintf("--ak=%s", ak))
    optionArgs = append(optionArgs, fmt.Sprintf("--sk=%s", sk))
    config.ExecProvider = &api.ExecConfig{
        Command:    "cci-iam-authenticator",
        APIVersion: apiVersion,
        Args:       append([]string{"token"}, optionArgs...),
        Env:        make([]api.ExecEnvVar, 0),
    }
    return kubernetes.NewForConfig(config)
}
```

生成kubeconfig配置文件用于认证配置，参考**使用cci-iam-authenticator**的子命令generate-kubeconfig生成kubeconfig配置文件

```
import (
    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/tools/clientcmd"
)

// NewClient 通过kubeconfig配置文件创建Clientset
// kubeconfig配置文件需包含认证相关信息，具体请参考《cci-iam-authenticator使用参考》生成kubeconfig配置文件：https://support.huaweicloud.com/devg-cci/cci_kubectl_03.html
func NewClient() (*kubernetes.Clientset, error) {
    config, err := clientcmd.BuildConfigFromFlags("", "/path/to/kubeconfig")
    if err != nil {
        return nil, err
    }
    return kubernetes.NewForConfig(config)
}
```

代码示例

您可以前往**开发体验馆Codelabs / Namespace生命周期代码示例（Go）**下载相关代码，并在线调试。

FAQ

问：上述代码示例是否存在请求结果返回码为401的情况?

答：一般情况下，如果密码或AK/SK配置无误，client-go提供的定期刷新token的机制（即定期调用cci-iam-authenticator刷新token，具体实现参考**https://github.com/kubernetes/client-go/blob/master/plugin/pkg/client/auth/exec/exec.go**），能确保token不会因过期而失效（token的有效期为24小时），从而避免请求结果返回码为401。

但当账号权限发生变更时，token可能也会失效，不属于超期失效的情形，在该情形下，仍会出现请求结果返回码为401的情况。

2 使用 client-go 访问 CCI 的 CRD 资源 Network

您可以前往**开发体验馆Codelabs / Namespace生命周期代码示例（Go）**下载相关代码，并在线调试。

初始化项目

创建项目examples.com/cci-examples。

说明

项目依赖k8s.io/client-go、k8s.io/code-generator，以下版本可供参考

- k8s.io/client-go@kubernetes-1.15.0、k8s.io/code-generator@kubernetes-1.15.0
- k8s.io/client-go@kubernetes-1.16.0、k8s.io/code-generator@kubernetes-1.16.0
- k8s.io/client-go@kubernetes-1.17.0、k8s.io/code-generator@kubernetes-1.17.0
- k8s.io/client-go@kubernetes-1.18.0、k8s.io/code-generator@kubernetes-1.18.0
- k8s.io/client-go@kubernetes-1.19.0、k8s.io/code-generator@kubernetes-1.19.0
- k8s.io/client-go@kubernetes-1.20.0、k8s.io/code-generator@kubernetes-1.20.0

```
mkdir -p examples.com/cci-examples
cd examples.com/cci-examples/
go mod init examples.com/cci-examples
go get k8s.io/client-go@kubernetes-1.15.0
go get k8s.io/code-generator@kubernetes-1.15.0
```

定义 CRD 资源 Network

创建文件夹pkg/apis/networking.cci.io/v1beta1，其中networking.cci.io为CRD资源的group，v1beta1为CRD资源版本

```
mkdir -p pkg/apis/networking.cci.io/v1beta1
```

在新建文件夹中新建以下文件：

doc.go

```
// +k8s:deepcopy-gen=package
// +groupName=networking.cci.io
// +groupGoName=NetworkingCCI
package v1beta1
```

types.go

```
package v1beta1

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// NetworkList is a list of network resource in container.
type NetworkList struct {
    metav1.TypeMeta `json:",inline"`
    // Standard list metadata.
    // More info: https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata
    // +optional
    metav1.ListMeta `json:"metadata,omitempty" protobuf:"bytes,1,opt,name=metadata"`
    Items           []Network `json:"items" protobuf:"bytes,2,rep,name=items"`
}

// +genclient
// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// Network is a network resource in container.
type Network struct {
    metav1.TypeMeta `json:",inline"`
    // +optional
    metav1.ObjectMeta `json:"metadata,omitempty" protobuf:"bytes,1,opt,name=metadata"`
    // Spec defines the attributes on a network
    // +optional
    Spec NetworkSpec `json:"spec,omitempty" protobuf:"bytes,2,opt,name=spec"`
    // Status describes the network status
    // +optional
    Status NetworkStatus `json:"status,omitempty" protobuf:"bytes,3,opt,name=status"`
}

// NetworkSpec describes the attributes on a network resource.
type NetworkSpec struct {
    // network type
    NetworkType string `json:"networkType,omitempty" protobuf:"bytes,5,opt,name=networkType"`
    // ID of the VPC to attach
    AttachedVPC string `json:"attachedVPC,omitempty" protobuf:"bytes,4,opt,name=attachedVPC"`
    // network ID
    NetworkID string `json:"networkID,omitempty" protobuf:"bytes,7,opt,name=networkID"`
    // Subnet ID
    SubnetID string `json:"subnetID,omitempty" protobuf:"bytes,8,opt,name=subnetID"`
    // available zone
    AvailableZone string `json:"availableZone,omitempty" protobuf:"bytes,9,opt,name=availableZone"`
    // The CIDR of the network
    CIDR string `json:"cidr,omitempty" protobuf:"bytes,3,opt,name=cidr"`
}

// NetworkStatus describes the status of a network
type NetworkStatus struct {
    // State describes the network state
    // +optional
    State string `json:"state" protobuf:"bytes,1,opt,name=state"`
    // Message describes why network is in current state
    // +optional
    Message string `json:"message,omitempty" protobuf:"bytes,2,opt,name=message"`
}

const (
    // NetworkInitializing means the network is initializing
    NetworkInitializing = "Initializing"
    // NetworkPending means the network is processing
    NetworkPending = "Pending"
    // NetworkActive means the network is available
    NetworkActive = "Active"
    // NetworkFailed means the network is not available
    NetworkFailed = "Failed"
```

```
    // NetworkTerminating means the network is undergoing graceful termination
    NetworkTerminating = "Terminating"
)
```

register.go

```
package v1beta1

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/apimachinery/pkg/runtime"
    "k8s.io/apimachinery/pkg/runtime/schema"
)

// GroupName is the group name use in this package
const GroupName = "networking.cci.io"

// SchemeGroupVersion is group version used to register these objects
var SchemeGroupVersion = schema.GroupVersion{Group: GroupName, Version: "v1beta1"}

// Resource takes an unqualified resource and returns a Group qualified GroupResource
func Resource(resource string) schema.GroupResource {
    return SchemeGroupVersion.WithResource(resource).GroupResource()
}

var (
    // localSchemeBuilder and AddToScheme will stay in k8s.io/kubernetes.
    SchemeBuilder      = runtime.NewSchemeBuilder(addKnownTypes)
    localSchemeBuilder = &SchemeBuilder
    AddToScheme        = localSchemeBuilder.AddToScheme
)

// Adds the list of known types to the given scheme.
func addKnownTypes(scheme *runtime.Scheme) error {
    scheme.AddKnownTypes(SchemeGroupVersion,
        &Network{},
        &NetworkList{},
    )
    // Add the watch version that applies
    metav1.AddToGroupVersion(scheme, SchemeGroupVersion)
    return nil
}
```

创建构建脚本

新建hack文件夹用以存放构建脚本及依赖文件

```
mkdir hack
```

在hack文件夹中创建以下文件：

tools.go，建立该文件来依赖code-generator

```
// +build tools

/*
Copyright 2019 The Kubernetes Authors.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

```
*/

// This package imports things required by build scripts, to force `go mod` to see them as dependencies
package tools

import _ "k8s.io/code-generator"
```

update-codegen.sh

```
#!/usr/bin/env bash

# Copyright 2017 The Kubernetes Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

set -o errexit
set -o nounset
set -o pipefail

SCRIPT_ROOT=$(dirname "${BASH_SOURCE[0]}")/..
CODEGEN_PKG=${CODEGEN_PKG:-$(cd "${SCRIPT_ROOT}"; ls -d -1 ./vendor/k8s.io/code-generator 2>/dev/
null || echo ../code-generator)}

# generate the code with:
# --output-base    because this script should also be able to run inside the vendor dir of
#                  k8s.io/kubernetes. The output-base is needed for the generators to output into the vendor dir
#                  instead of the $GOPATH directly. For normal projects this can be dropped.
# generators deepcopy,client,informer,lister
export CLIENTSET_PKG_NAME=networking.cci.io
export CLIENTSET_NAME_VERSIONED=v1beta1
"${CODEGEN_PKG}"/generate-groups.sh "deepcopy,client" \
  examples.com/cci-examples/pkg/client examples.com/cci-examples/pkg/apis \
  networking.cci.io:v1beta1 \
  --output-base "$(dirname "${BASH_SOURCE[0]}")/../../.." \
  --go-header-file "${SCRIPT_ROOT}"/hack/boilerplate.go.txt

# To use your own boilerplate text append:
#   --go-header-file "${SCRIPT_ROOT}"/hack/custom-boilerplate.go.txt
```

boilerplate.go.txt构建出来的文件的文件头，可定制

```
/*
Copyright The Kubernetes Authors.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/
```

至此，完成所有准备过程，此时目录的文件结构如下：

```
├── go.mod
├── go.sum
```

```
├── hack
│   ├── boilerplate.go.txt
│   ├── tools.go
│   └── update-codegen.sh
└── pkg
    └── apis
        └── networking.cci.io
            └── v1beta1
                ├── doc.go
                ├── register.go
                └── types.go

5 directories, 8 files
```

执行命名生成代码

以下命令在linux环境下执行

```
# 生成vendor文件夹
go mod vendor
# 执行构建脚本
chmod 755 hack/update-codegen.sh
# hack/update-codegen.sh会执行vendor/k8s.io/code-generator/generate-groups.sh
chmod 755 vendor/k8s.io/code-generator/generate-groups.sh
./hack/update-codegen.sh
```

执行成功后，将会生成代码，目录结构将为

```
├── go.mod
├── go.sum
├── hack
│   ├── boilerplate.go.txt
│   ├── tools.go
│   └── update-codegen.sh
└── pkg
    ├── apis
    │   └── networking.cci.io
    │       └── v1beta1
    │           ├── doc.go
    │           ├── register.go
    │           ├── types.go
    │           └── zz_generated.deepcopy.go
    └── client
        └── networking.cci.io
            └── v1beta1
                ├── clientset.go
                ├── doc.go
                ├── fake
                │   ├── clientset_generated.go
                │   ├── doc.go
                │   └── register.go
                ├── scheme
                │   ├── doc.go
                │   └── register.go
                └── typed
                    └── networking.cci.io
                        └── v1beta1
                            ├── doc.go
                            ├── fake
                            │   ├── doc.go
                            │   ├── fake_network.go
                            │   └── fake_networking.cci.io_client.go
                            ├── generated_expansion.go
                            ├── network.go
                            └── networking.cci.io_client.go
```

示例代码：创建 Network

说明

该示例测试版本：

k8s.io/client-go@kubernetes-1.15.0

k8s.io/code-generator@kubernetes-1.15.0

```
import (
    "time"

    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/apimachinery/pkg/util/wait"
    "k8s.io/client-go/tools/clientcmd"

    "examples.com/cci-examples/pkg/apis/networking.cci.io/v1beta1"
    clientset "examples.com/cci-examples/pkg/client/networking.cci.io/v1beta1"
)

const (
    name      = "test-k8s-client-namespace-cn-north-1-default-network"
    namespace = "test-k8s-client-namespace"
)

// CreateNetwork 创建Network，并等待其状态变更为Active
// 参考《Namespace和Network》https://support.huaweicloud.com/devg-cci/cci_05_0023.html
// API参考：https://support.huaweicloud.com/api-cci/
createNetworkingCciIoV1beta1NamespacedNetwork.html
func CreateNetwork() (*v1beta1.Network, error) {
    config, _ := clientcmd.BuildConfigFromFlags("", "{path to kubeconfig}")
    cs, err := clientset.NewForConfig(config)
    if err != nil {
        return nil, err
    }

    projectId := "<账号ID，可以在我的凭证获取>"
    domainId := "<项目ID，可以在我的凭证获取>"
    securityGroupID := "<安全组ID，可以在安全组控制台获取>"
    availableZone := "<az名称，例如cn-north-1a、cn-north-4a或cn-east-3a>"
    vpcID := "虚拟私有云的ID，可在VPC控制台获取"
    cidr := "<子网网段，例如192.168.128.0/18>"
    networkID := "<子网的网络ID，可在VPC控制台 > 子网中获取>"
    subnetID := "<子网ID，可在VPC控制台 > 子网获取>"

    network := &v1beta1.Network{
        ObjectMeta: metav1.ObjectMeta{
            Annotations: map[string]string{
                "network.alpha.kubernetes.io/default-security-group": securityGroupID,
                "network.alpha.kubernetes.io/domain-id":              domainId,
                "network.alpha.kubernetes.io/project-id":             projectId,
            },
            Name: name,
        },
        Spec: v1beta1.NetworkSpec{
            AvailableZone: availableZone,
            CIDR:          cidr,
            AttachedVPC:   vpcID,
            NetworkID:     networkID,
            NetworkType:   "underlay_neutron",
            SubnetID:      subnetID,
        },
    }
    network, err = cs.NetworkingCCIV1beta1().Networks(namespace).Create(network)
    if err != nil {
        return nil, err
    }

    // 查询Network状态，等待其状态变为"Active"
    err = wait.Poll(time.Second*5, time.Second*30, func() (done bool, err error) {
```

```
        network, err = cs.NetworkingCCIV1beta1().Networks(namespace).Get(name, metav1.GetOptions{})
        if err != nil {
            return false, err
        }
        if network.Status.State == v1beta1.NetworkActive {
            return true, nil
        }
        return false, nil
    })
    return network, err
}
```

参考文档

Code Generation for CustomResources

code-generator

sample-controller

3 使用 kubernetes 官方 Python SDK 访问 CCI

本节介绍如何将cci认证工具cci-iam-authenticator与kubernetes-client/python结合使用以调用API。

安装 cci-iam-authenticator

请参考**使用kubectl**，下载安装及设置cci-iam-authenticator。

安装 kubernetes-client/python

您可参考**Installation**，下载安装kubernetes-client/python。

说明

当前**CCI的kubernetes API对应社区版本为1.19**，根据**Compatibility**，推荐使用的SDK版本为client 11.y.z。

使用 Python SDK

您可以前往**开发体验馆Codelabs / Namespace生命周期代码示例（Python）**下载相关代码，并在线调试。

首先需要先生成kubeconfig配置文件，参考**cci-iam-authenticator使用参考**，使用子命令generate-kubeconfig生成kubeconfig配置文件。

说明

- 这里的示例代码采用了定期刷新token的方式来防止token过期（缓存值token有效期为24小时），您可以增加获取失败重试的操作，以提升可用性。
- 定期刷新token的方式不适用于该账号权限发生变更的情形，如果账号权限发生变更（如主账号变更子账号权限，导致子账号权限发生变更），变更前获取的token会失效，需要重新获取

```
# -*- coding: utf-8 -*-
import logging
import time
import threading

from kubernetes import client, config
```

```
NAMESPACE = "test-k8s-client-namespace"

logging.basicConfig(
    level=logging.INFO,
    datefmt="%Y-%m-%d %H:%M:%S",
    format="%(asctime)s %(levelname)s %(message)s",
)


def create_namespace():
    flavor = "general-computing"
    pool_size = "10"

    namespace = client.V1Namespace(
        metadata=client.V1ObjectMeta(
            name=NAMESPACE,
            annotations={
                "namespace.kubernetes.io/flavor": flavor,
                "network.cci.io/warm-pool-size": pool_size,
            },
            labels={
                "rbac.authorization.cci.io/enable-k8s-rbac": "false",
            }
        )
    )

    logging.info("start to create namespace %s", NAMESPACE)
    client.CoreV1Api().create_namespace(namespace)
    logging.info("namespace created")


def create_network():
    name = "test-k8s-client-namespace-cn-north-7-default-network"
    project_id = "{project_id}"
    domain_id = "{domain_id}"
    security_group_id = "{security_group_id}"
    available_zone = "{available_zone}"
    vpc_id = "{vpc_id}"
    cidr = "{cidr}"
    network_id = "{network_id}"
    subnet_id = "{subnet_id}"

    body = {
        "apiVersion": "networking.cci.io/v1beta1",
        "kind": "Network",
        "metadata": {
            "annotations": {
                "network.alpha.kubernetes.io/default-security-group": security_group_id,
                "network.alpha.kubernetes.io/domain-id": domain_id,
                "network.alpha.kubernetes.io/project-id": project_id,
            },
            "name": name,
        },
        "spec": {
            "availableZone": available_zone,
            "cidr": cidr,
            "attachedVPC": vpc_id,
            "networkID": network_id,
            "networkType": "underlay_neutron",
            "subnetID": subnet_id,
        }
    }

    api = client.CustomObjectsApi()
    logging.info("start to create network")
    api.create_namespaced_custom_object(
        group="networking.cci.io",
        version="v1beta1",
        namespace=NAMESPACE,
```

```
        plural="networks",
        body=body,
    )
    logging.info("network created")


def create_deployment():
    app = "test-k8s-client-deployment"
    image = "library/nginx:stable-alpine-perl"

    body = client.V1Deployment(
        api_version="apps/v1",
        kind="Deployment",
        metadata=client.V1ObjectMeta(name=app),
        spec=client.V1DeploymentSpec(
            replicas=2,
            selector={"matchLabels": {"app": app}},
            template=client.V1PodTemplateSpec(
                metadata=client.V1ObjectMeta(labels={"app": app}),
                spec=client.V1PodSpec(
                    containers=[
                        client.V1Container(
                            name="container-0",
                            image=image,
                            resources=client.V1ResourceRequirements(
                                requests={"cpu": "500m", "memory": "1024Mi"},
                                limits={"cpu": "500m", "memory": "1024Mi"},
                            ),
                        )
                    ],
                    image_pull_secrets=[
                        client.V1LocalObjectReference(name="imagepull-secret")],
                    priority=0),
            ),
        )
    )
    logging.info("start to create deployment %s/%s", NAMESPACE, app)
    client.AppsV1Api().create_namespaced_deployment(NAMESPACE, body)
    logging.info("deployment created")


def get_deployment():
    app = "test-k8s-client-deployment"
    resp = client.AppsV1Api().read_namespaced_deployment(app, NAMESPACE)
    logging.info("deployment detail: %s", resp)


def delete_deployment():
    app = "test-k8s-client-deployment"
    logging.info("start to delete deployment")
    client.AppsV1Api().delete_namespaced_deployment(app, NAMESPACE)
    logging.info("deployment deleted")


def delete_namespace():
    logging.info("start to delete namespace: %s", NAMESPACE)
    client.CoreV1Api().delete_namespace(NAMESPACE)


def main():
    # Configs can be set in Configuration class directly or using helper
    # utility. If no argument provided, the config will be loaded from
    # default location.
    path = '{path to kubeconfig}'
    config.load_kube_config(path)

    # 因为token有效期为24小时，所以这里设置了一个每12小时获取新的token的定时任务
    # 注意：如果账号权限发生变更（如主账号变更子账号权限，导致子账号权限发生变更），变更前获取的
token会失效，需要重新获取。
```

```
    # 另外，您可以增加获取失败重试的操作，以提升可用性
    def _refresh():
        while True:
            time.sleep(12 * 3600)
            try:
                config.load_kube_config(path)
            except Exception as e:
                print("load_kube_config error: %s" % e)
    t = threading.Thread(target=_refresh)
    t.daemon = True
    t.start()

    create_namespace()
    create_network()
    # wait for namespace and network to be active
    logging.info("waiting for namespace and network to be active")
    time.sleep(30)
    create_deployment()
    get_deployment()
    delete_deployment()
    delete_namespace()


if __name__ == '__main__':
    main()
```

FAQ

问：以上示例是否适用于其他版本的kubernetes-client/python?

答：上述示例已通过测试，测试环境python3.7.4，测试版本包括：

1. 9.0.1
2. 10.1.0
3. 11.0.0
4. 12.0.1
5. 17.17.0
6. 18.17.0a1
7. 19.15.0

4 使用 kubernetes 官方 Java SDK 访问 CCI

本节将介绍如何将CCI认证工具cci-iam-authenticator与**kubernetes-client/java**结合使用以调用API。

安装 cci-iam-authenticator

请参考**使用kubectl**，下载安装及设置cci-iam-authenticator。

安装 kubernetes-client/java

详情请参考**Installation** 。

说明

当前**CCI服务开放的API对应的Kubernetes版本为1.19**，根据**Versioning-and-Compatibility**，推荐使用的SDK版本为11.0.2及以上。

如果要使用GenericKubernetesClient（参考**Code-Examples**），则需要9.0.0+以上版本

使用 Java SDK

您可以前往**开发体验馆Codelabs / Namespace生命周期代码示例（Java）**下载相关代码，并在线调试。

说明

示例已通过以下版本的测试：

1. 11.0.2

将以下依赖添加到项目的POM文件中：

```
<dependency>
    <groupId>io.kubernetes</groupId>
    <artifactId>client-java</artifactId>
    <version>11.0.2</version>
</dependency>
```

通过kubeconfig配置文件创建ApiClient（参考**使用cci-iam-authenticator**的子命令generate-kubeconfig生成kubeconfig配置文件）

```
public class CommonCases {

    // ...

    public static void main(String[] args) throws IOException, ApiException, InterruptedException {
```

```
        // file path to your KubeConfig
        String kubeConfigPath = "<path to kubeconfig>";

        // loading the out-of-cluster config, a kubeconfig from file-system
        File file = new File(kubeConfigPath);
        KubeConfig config = KubeConfig.loadKubeConfig(new FileReader(file));
        config.setFile(file);
        ApiClient client = buildClient(config).build();

        // ...

    }

    public static ClientBuilder buildClient(KubeConfig config) throws IOException {
        final ClientBuilder builder = new ClientBuilder();

        String server = config.getServer();
        if (!server.contains("://")) {
            if (server.contains(":443")) {
                server = "https://" + server;
            } else {
                server = "http://" + server;
            }
        }

        builder.setVerifyingSsl(config.verifySSL());

        builder.setBasePath(server);
        builder.setAuthentication(new CCIKubeconfigAuthentication(config));
        return builder;
    }
}
```

CCIKubeconfigAuthentication.java

```
package com.huawei.demos;

import io.kubernetes.client.openapi.ApiClient;
import io.kubernetes.client.util.KubeConfig;
import io.kubernetes.client.util.credentials.Authentication;
import okhttp3.Interceptor;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.time.Instant;


/**
 * Uses a {@link KubeConfig} to configure {@link ApiClient} authentication to the CCI Kubernetes API.
 *
 * <p> Only try to use AccessTokenAuthentication mechanisms, which is enough for CCI.
 */
public class CCIKubeconfigAuthentication implements Authentication, Interceptor {

    private static final Logger log = LoggerFactory.getLogger(CCIKubeconfigAuthentication.class);
    private final KubeConfig config;
    private String token;
    private Instant expiry;

    public CCIKubeconfigAuthentication(final KubeConfig config) throws IOException {
        this.config = config;
        this.expiry = Instant.MIN;
    }
```

```
    private String getToken() {
        // get access token every 600 seconds
        if (Instant.now().isAfter(this.expiry)) {
            log.debug("Token expired, get new one from kubeconfig");
            this.token = config.getAccessToken();
            if (this.token != null) {
                this.expiry = Instant.now().plusSeconds(600);
            }
        }
        return this.token;
    }

    @Override
    public void provide(ApiClient client) {
        OkHttpClient httpClient = client.getHttpClient().newBuilder().addInterceptor(this).build();
        client.setHttpClient(httpClient);
    }

    @Override
    public Response intercept(Interceptor.Chain chain) throws IOException {
        Request request = chain.request();
        Request authRequest;
        authRequest = request.newBuilder().header("Authorization", "Bearer " + getToken()).build();
        return chain.proceed(authRequest);
    }
}
```

访问CCI服务

```
public class CommonCases {

    // ...

    private static void createNamespace(CoreV1Api api) throws ApiException {
        String enableK8sRbac = "false";
        String flavor = "general-computing";
        String warmPoolSize = "10";

        Map<String, String> labels = new HashMap<>();
        labels.put("rbac.authorization.cci.io/enable-k8s-rbac", enableK8sRbac);
        Map<String, String> annotations = new HashMap<>();
        annotations.put("namespace.kubernetes.io/flavor", flavor);
        annotations.put("network.cci.io/warm-pool-size", warmPoolSize);
        V1Namespace namespace = new V1Namespace()
                .metadata(new V1ObjectMeta().name(NAMESPACE).labels(labels).annotations(annotations));
        LOGGER.info("start to create namespace {}", NAMESPACE);
        api.createNamespace(namespace, null, null, null);
        LOGGER.info("namespace created");
    }

    private static void createNetwork(GenericKubernetesApi<Network, NetworkList> networkApi) throws
ApiException {
        String name = NETWORK;
        String projectID = "<账号ID，可以在我的凭证获取>";
        String domainID = "<项目ID，可以在我的凭证获取>";
        String securityGroupID = "<安全组ID，可以在安全组控制台获取>";
        String availableZone = "<az名称，例如cn-north-1a、cn-north-4a或cn-east-3a>";
        String vpcID = "虚拟私有云的ID，可在VPC控制台获取";
        String cidr = "<子网网段，例如192.168.128.0/18>";
        String networkID = "<子网的网络ID，可在VPC控制台 > 子网中获取>";
        String subnetID = "<子网ID，可在VPC控制台 > 子网获取>";
        String networkType = "underlay_neutron";

        Map<String, String> annotations = new HashMap<>();
        annotations.put("network.alpha.kubernetes.io/default-security-group", securityGroupID);
        annotations.put("network.alpha.kubernetes.io/domain-id", domainID);
        annotations.put("network.alpha.kubernetes.io/project-id", projectID);

        Network network = new Network()
```

```
            .metadata(new
V1ObjectMeta().name(name).namespace(NAMESPACE).annotations(annotations))
            .spec(new NetworkSpec()
                    .availableZone(availableZone)
                    .cidr(cidr)
                    .attachedVPC(vpcID)
                    .networkID(networkID)
                    .networkType(networkType)
                    .subnetID(subnetID));

        LOGGER.info("start to create network {}/{}", NAMESPACE, name);
        networkApi.create(network).throwsApiException();
        LOGGER.info("network created");
    }

    private static void waitNamespaceActive(CoreV1Api api) throws ApiException, InterruptedException {
        for (int i = 0; i < 5; i++) {
            V1Namespace ns = api.readNamespace(NAMESPACE, null, null, null);
            if (ns.getStatus() != null && NAMESPACE_ACTIVE.equals(ns.getStatus().getPhase())) {
                return;
            }
            Thread.sleep(WAIT_ACTIVE_MILLIS);
        }
        throw new IllegalStateException("namespace not active");
    }

    private static void waitNetworkActive(GenericKubernetesApi<Network, NetworkList> networkApi) throws
ApiException, InterruptedException {
        for (int i = 0; i < 5; i++) {
            Network network = networkApi.get(NAMESPACE, NETWORK).throwsApiException().getObject();
            if (network.getStatus() != null && NETWORK_ACTIVE.equals(network.getStatus().getState())) {
                return;
            }
            Thread.sleep(WAIT_ACTIVE_MILLIS);
        }
        throw new IllegalStateException("network not active");
    }

    private static void createDeployment(AppsV1Api api) throws ApiException {
        String app = APP;
        String cpu = "500m";
        String memory = "1024Mi";
        String containerName = "container-0";
        String image = "library/nginx:stable-alpine-perl";

        Map<String, Quantity> limits = new HashMap<>();
        limits.put("cpu", Quantity.fromString(cpu));
        limits.put("memory", Quantity.fromString(memory));
        V1Container container = new V1Container()
                .name(containerName)
                .image(image)
                .resources(new V1ResourceRequirements().limits(limits).requests(limits));

        Map<String, String> labels = new HashMap<>();
        labels.put("app", app);
        V1PodTemplateSpec podTemplateSpec = new V1PodTemplateSpec()
                .spec(new V1PodSpec()
                        .priority(0)
                        .imagePullSecrets(Collections.singletonList(new
V1LocalObjectReference().name("imagepull-secret")))
                        .containers(Collections.singletonList(container)))
                .metadata(new V1ObjectMeta().labels(labels));

        V1Deployment deployment = new V1Deployment()
                .metadata(new V1ObjectMeta().name(app))
                .spec(new V1DeploymentSpec()
                        .replicas(2)
                        .selector(new V1LabelSelector().matchLabels(labels))
                        .template(podTemplateSpec));
```

```
        LOGGER.info("start to create deployment {}/{}", NAMESPACE, APP);
        api.createNamespacedDeployment(NAMESPACE, deployment, null, null, null);
        LOGGER.info("deployment created");
    }

    private static void getDeployment(AppsV1Api api) throws ApiException {
        V1Deployment deployment = api.readNamespacedDeployment(APP, NAMESPACE, null, null, null);
        LOGGER.info("deployment metadata: {}", deployment.getMetadata());
    }

    private static void deleteDeployment(AppsV1Api api) throws ApiException {
        LOGGER.info("start to delete deployment");
        api.deleteNamespacedDeployment(APP, NAMESPACE, null, null, null, null, null, null);
        LOGGER.info("deployment deleted");
    }

    private static void deleteNamespace(CoreV1Api api) throws ApiException {
        LOGGER.info("start to delete namespace: {}", NAMESPACE);
        api.deleteNamespace(NAMESPACE, null, null, null, null, null, null);
        LOGGER.info("namespace deleted");
    }
}
```

FAQ

问：以上示例是否适用于其他版本的kubernetes-client/java?

答：由于以上示例使用了GenericKubernetesClient（参考**Code-Examples**，需要9.0.0+以上版本），所以不适用9.0.0以下版本SDK。

另外由于不同版本SDK之间存在一定差别，上述示例代码需要作一些细微调整才能适用于不同版本SDK，请自行调试。