目录

容器实例访问IDC内网域名

如果您已成功在自建Kubernetes集群中创建云上的容器实例，对于容器实例访问自建Kubernetes集群内网服务的场景，需通过集群内的Coredns服务进行域名解析。

为实现从容器实例到集群Coredns服务的连通性，需进行如下配置：

步骤1：指定虚拟节点启动参数

虚拟节点启动参数中指定--cluster-dns为Coredns的服务地址。

Yaml示例如下：

```
...
          args:
            - --nodename=rbkci-virtual-kubelet
            # 指定虚拟节点的DNS配置，为集群内Coredns服务的IP地址
            - --cluster-dns=10.254.0.10
            - --cluster-domain=cluster.local
            - --kcilet-kubeconfig-path=/root/.kube/config
            - --enable-node-lease
 ...
```

步骤2：对目标容器使能kube-proxy，以支持ClusterIP访问

创建实例时可通过Pod template annotation对该pod开启kube-proxy：

Annotation Key	Annotation Value示例	是否必填	描述
k8s.ksyun.com/kci-kube-proxy-enabled	'true' / 'false'	否	默认值：‘false’。当为true时，为该pod开启kube-proxy，使该pod具备访问集群内clusterIP类型服务的能力；否则不开启。

Yaml示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-rbkci
  namespace: default
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
        annotations:
            k8s.ksyun.com/kci-kube-proxy-enabled: 'true'  #对该pod使能kube-proxy
        labels:
            app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: rbkci-virtual-kubelet   #指定nodeName将pod调度到虚拟节点上
```

注：

Kube-proxy也支持通过启动参数--kube-proxy-enable在vk（即虚拟节点）维度进行全局配置（参数缺省值为false），开启后则默认该vk管理的所有pod都会开启Kube-proxy，支持ClusterIP访问。Pod annotation中配置优先级高于vk全局配置。

vk yaml示例如下：

```
...
       args:
            - --nodename=rbkci-virtual-kubelet
            # 指定虚拟节点的DNS配置，为集群内Coredns服务的IP地址
            - --cluster-dns=10.254.0.10
            - --cluster-domain=cluster.local
            - --kcilet-kubeconfig-path=/root/.kube/config
            - --enable-node-lease
            # 虚拟节点管理的所有实例使能kube-proxy
```

```
            - --kube-proxy-enable
    ...
```

自定义sidecar方式采集容器实例日志

对于通过filebeat采集容器实例日志至Kafka服务的场景，若您对filebeat有自定义需求，可通过如下方式进行配置。

前提条件

1. 已在Kubernetes集群中部署虚拟节点，部署方式：KCE集群参考Kubernetes集群对接KCI，自建集群参考自建Kubernetes集群中对接KCI。
2. 容器实例所属VPC已与Kafka集群所属网络打通。

   注：若Kafka集群有安全组配置，入站规则中需配置放行broker监听端口。
3. 目标采集的容器实例日志类型为容器文件日志，自定义sidecar方式下不支持采集容器标准输出日志。

步骤1：创建filebeat配置文件

在集群Kube-system命名空间下创建configmap filebeat-config用于配置Kafka output。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-config
  namespace: kube-system
data:
  filebeat.yml: |
    ---
    filebeat.config:
      inputs:
        path: "${path.config}/inputs.d/*.yml"
        reload.enabled: true
        reload.period: "10s"
      modules:
        path: "${path.config}/modules.d/*.yml"
        reload.enabled: true
    output.kafka:
      # 配置Kafka broker地址
      hosts: ["10.0.0.***:9092", "10.0.0.***:9092", "10.0.0.***:9092"]

      # 动态匹配topic地址 + 分区配置
      topic: '%{[fields.log_topic]}'
      partition.round_robin:
      reachable_only: false

      required_acks: 1
      compression: gzip
      max_message_bytes: 1000000
```

注：更多Kafka output配置可参考filebeat官网文档Configure the Kafka output。

步骤2：创建自定义sidecar配置

在集群kube-system命名空间下创建configmap用于自定义sidecar配置，yaml示例如下：

注：

1. configmap名称与对应虚拟节点同名。
2. 配置key必须为config.yaml。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: rbkci-virtual-kubelet
  namespace: kube-system
data:
  config.yaml: |
    mutation:
      customMutation:
        containers:    #自定义sidecar容器配置
          - args:
              - -c
              - /usr/share/filebeat/config/filebeat.yml
              - -e
```

```
            command:
              - /usr/share/filebeat/filebeat
            image: docker.elastic.co/beats/filebeat:7.17.0    #自定义filebeat镜像
            imagePullPolicy: Always
            name: filebeat
            volumeMounts:
              - mountPath: /usr/share/filebeat/config
                name: filebeat-config
              - mountPath: /usr/share/filebeat/inputs.d
                name: filebeat-inputs
              - mountPath: /usr/share/filebeat/data
                name: filebeat-data
              - mountPath: /home/q/logs/collected   #自定义业务容器日志的hostPath路径
                name: filebeat-logdir
            SecurityContext:
              runAsUser: 0
        volumes:    #对应container的volume定义
          - configMap:
              name: filebeat-config
            name: filebeat-config
          - configMap:
              name: filebeat-inputs
            name: filebeat-inputs
          - hostPath:
              path: /usr/share/filebeat/data
              type: DirectoryOrCreate
            name: filebeat-data
          - hostPath:
              path: /home/q/logs/collected
            name: filebeat-logdir
```

步骤3：配置Kafka服务端域名解析

容器实例通过CoreDNS服务解析消费端地址，Kafka服务端域名需通过集群Coredns hosts配置，示例如下：

```
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
          pods insecure
          fallthrough in-addr.arpa ip6.arpa
        }
        # hosts can add hosts's item into dns, see https://coredns.io/plugins/hosts/
        hosts {
            198.18.96.191 hub.kce.ksyun.com
            10.0.0.*** kmr-c0b4eaab-gn-e2a4babf-broker-1-1.ksc.com  // kafka broker 域名
            10.0.0.*** kmr-c0b4eaab-gn-e2a4babf-broker-1-2.ksc.com  // kafka broker 域名
            10.0.0.*** kmr-c0b4eaab-gn-e2a4babf-broker-1-3.ksc.com  // kafka broker 域名
            fallthrough
        }
        prometheus :9153
        forward . /etc/resolv.conf
        cache 30
        loop
        reload
        loadbalance
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2021-12-15T11:14:52Z"
  name: coredns
  namespace: kube-system
  resourceVersion: "6152795"
  uid: c1e29f37-d37d-4c90-9ca4-418a628cc04b
```

步骤4： 配置日志采集规则

在集群kube-system命名空间下创建configmap `filebeat-inputs`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-inputs
  namespace: kube-system
data:
  kci.yml: |
    ---
    - type: "log"
```

```
      symlinks: true
      enabled: true
      fields:
        log_topic: filelog
      paths:
      - "/home/q/logs/collected/*.log"    #指定日志采集文件路径
```

示例：为容器实例开启自定义sidecar日志采集

以下以nginx pod为例，通过定义annotation，为pod指定底层系统镜像及开启kube-proxy。

注：

1. 指定底层系统镜像：目前自定义sidecar能力通过非标准镜像方式支持，具体镜像ID以容器团队提供为准。
2. 开启Kube-proxy：容器实例需通过CoreDNS服务解析消费端地址，需开启Kube-proxy以使能pod访问ClusterIP类型服务。
3. 关闭Klog配置：暂不支持同时开启klog日志采集与自定义sidecar日志采集。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-rbkci
  namespace: default
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
        annotations:
            k8s.ksyun.com/kci-base-image: "11a28ef4-588e-4592-9a70-c23a0abd8d29"    #自定义镜像/共享镜像ID
            k8s.ksyun.com/kci-kube-proxy-enabled: "true"     #开启Kube-proxy
        labels:
            app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        volumeMounts:
            - name: rbkcilog
              mountPath: /var/log/nginx   #将底层目录"/home/q/logs/collected"挂载到容器"/var/log/nginx"路径下
      volumes:
        - hostPath:
            path: /home/q/logs/collected  #对应自定义filebeat的日志采集路径
          name: rbkcilog
      nodeName: rbkci-virtual-kubelet
```

若需要在虚拟节点维度开启自定义日志采集，可修改virtual-kubelet启动参数，在vk级别指定自定义镜像及开启kube-proxy，示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rbkci-virtual-kubelet
  namespace: kube-system
  labels:
    k8s-app: rbkci-virtual-kubelet
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: rbkci-virtual-kubelet
  template:
    metadata:
      name: rbkci-virtual-kubelet
      labels:
        k8s-app: rbkci-virtual-kubelet
    spec:
      serviceAccountName: virtual-kubelet-sa
      containers:
        - name: virtual-kubelet
          image: hub.kce.ksyun.com/ksyun/rbkci-virtual-kubelet:v1.1.0-beta
          args:
            - --nodename=rbkci-virtual-kubelet
            - --cluster-dns=10.254.0.10
            - --cluster-domain=cluster.local
            - --kcilet-kubeconfig-path=/root/.kube/config
            - --enable-node-lease
```

```
          # 虚拟节点管理的所有实例使能kube-proxy
          - --kube-proxy-enable
        imagePullPolicy: Always
        env:
          - name: VKUBELET_POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
          - name: TEMP_AKSK_CM
            value: user-temp-aksk
          - name: KCI_CLUSTER_ID
            value: ${cluster_id}
          - name: KCI_SUBNET_ID
            value: ${subnet_id}
          - name: KCI_SECURITY_GROUP_IDS
            value: ${security_group_ids}
          # 指定虚拟节点管理的所有实例底层镜像
          - name: KCI_BASE_IMAGE
            value: ${kci_base_image}
        volumeMounts:
          - mountPath: /root/.kube
            name: kubeconfig
          - mountPath: /var/log/kci-virtual-kubelet
            name: kci-provider-log
      volumes:
        - name: kubeconfig
          secret:
            secretName: rbkci-kubeconfig-secret
        - name: kci-provider-log
          hostPath:
            path: /var/log/kci-virtual-kubelet
```

验证日志投递效果

模拟容器日志，查询Kafka消费端消息，检查目标容器实例日志是否投递成功。

从上图中可以看到，filebeat的版本（7.17.0）和采集路径（/home/q/logs/collected）已与自定义sidecar的配置相同。

自定义IDC资源水位触发调度

在用户将自建集群和容器实例结合使用，利用云上资源来应对业务高峰的场景下，KCI调度插件支持在自建资源的分配率达到一个可配置的阈值后，就将新创建的Pod优先调度到容器实例上。

KCI调度插件简介

金山云提供了配合virtual-kubelet使用的scheduler-extender，支持如下调度策略：

- 当用户自建集群资源分配率未达到指定阈值时，优先使用自建资源创建pod
- 当自建集群资源分配超过阈值时，优先调度到virtual-kubelet节点上（即使用云上资源创建pod），并且可为不同vk节点设置权重

使用须知

- 目前提供了2个指标，“CPU分配率”和“内存分配率”。只要有一个指标超过阈值，即优先调度到vk上
- 计算资源分配率时，只统计role为node的节点（不统计master节点），为保证调度插件功能正常，使用前需要确认集群node节点上，已经打上label：kubernetes.io/role=node
- 使用前，需要移除virtual-kubelet上的污点或为pod添加污点容忍，使得vk可以参与k8s的正常调度

插件安装和部署

部署extender

yaml详情如下：

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: extender-conf
  namespace: kube-system
data:
  cpulimit: "0.5"   # 所有pod的request资源/所有node的allocatable资源的值，如果不关心它则设置为-1
  memlimit: "0.7"   # 如果不关心这个指标，设置为-1
  weight: '[        # weight是可选配置，配置不同vk节点的调度权重
```

```
    {
      "nodeName": "rbkci-virtual-kubelet",   # vk节点的名字
      "weight": 5
    },
    {
      "nodeName": "virtual-kubelet-cn-beijing-i",
      "weight": 1
    }
  ]'
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: scheduler-extender
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: scheduler-extender-admin
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    namespace: kube-system
    name: scheduler-extender
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-scheduler-policy
  namespace: kube-system
data:
 policy.cfg : |
  {
    "kind" : "Policy",
    "apiVersion" : "v1",
    "extenders" : [{
      "urlPrefix": "http://kci-scheduler-extender.kube-system/scheduler",
      "filterVerb": "predicates/always_true",
      "prioritizeVerb": "priorities/group_score",
      "preemptVerb": "preemption",
      "bindVerb": "",
      "weight": 1,
      "enableHttps": false,
      "nodeCacheCapable": false
    }]
  }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kci-scheduler-extender
  namespace: kube-system
  labels:
    app: kci-scheduler-extender
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kci-scheduler-extender
  template:
    metadata:
      labels:
        app: kci-scheduler-extender
    spec:
      serviceAccountName: scheduler-extender
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: type
                operator: NotIn
                values:
                - virtual-kubelet
      containers:
      - name: kci-scheduler-extender
        image: hub.kce.ksyun.com/ksyun/ksc-scheduler-extender:latest
        imagePullPolicy: IfNotPresent
        ports:
```

```
          - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: kci-scheduler-extender
  name: kci-scheduler-extender
  namespace: kube-system
spec:
  ports:
    - name: server-port
      port: 80
      protocol: TCP
      targetPort: 80
  selector:
    app: kci-scheduler-extender
  type: ClusterIP
```

确认kube-scheduler配置了configmap的访问权限

检查kube-scheduler是否配置了configmap的访问权限

```
# kubectl get clusterrole system:kube-scheduler -o yaml
...
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
  - watch
```

如果没有上面的配置，需要执行kubectl edit clusterrole system:kube-scheduler，把上面几行添加进去

修改kube-scheduler配置

修改kube-scheduler的配置，具体改动如下：

1. command中增加 --policy-configmap=custom-scheduler-policy
2. dnsPolicy设置为：ClusterFirstWithHostNet

   注意：

   - kube-scheduler一般以static pod的方式部署，其配置文件通常位于/etc/kubernetes/manifests目录下。如您想了解更多关于static pod的信息，请查看Kubernetes官方文档。
   - kube-scheduler一般部署在多个节点上，每个副本都要修改。

重启kube-scheduler生效

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  creationTimestamp: null
  labels:
    component: kube-scheduler
    tier: control-plane
  name: kube-scheduler
  namespace: kube-system
spec:
  containers:
  - command:
    - /usr/local/bin/kube-scheduler
    - --logtostderr=true
    - --v=10
    - --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig
    - --leader-elect=true
    - --leader-elect-lease-duration=60s
    - --leader-elect-renew-deadline=30s
    - --leader-elect-retry-period=10s
    - --kube-api-qps=100
    - --policy-configmap=custom-scheduler-policy   # 新增--policy-configmap配置
    image: hub.kce.ksyun.com/ksyun/kube-scheduler:v1.17.6-mp
    imagePullPolicy: Always
    livenessProbe:
      failureThreshold: 8
```

```
      httpGet:
        host: 127.0.0.1
        path: /healthz
        port: 10251
        scheme: HTTP
      initialDelaySeconds: 15
      timeoutSeconds: 15
    name: kube-scheduler
    resources:
      requests:
        cpu: 100m
    volumeMounts:
    - mountPath: /etc/kubernetes
      name: k8s
      readOnly: true
    - mountPath: /etc/localtime
      name: time-zone
      readOnly: true
  hostNetwork: true
  dnsPolicy: ClusterFirstWithHostNet  # dnsPolicy设置为ClusterFirstWithHostNet
  tolerations:
  - operator: Exists
  - key: CriticalAddonsOnly
    operator: Exists
  - effect: NoExecute
    operator: Exists
  volumes:
  - hostPath:
      path: /etc/kubernetes
    name: k8s
  - hostPath:
      path: /etc/localtime
    name: time-zone
```

插件使用示例

1.查看extender-conf的ConfigMap文件，了解当前的调度策略

```
kubectl -n kube-system describe cm extender-conf
```

预期输出：

```
Name:         extender-conf
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
cpulimit:
----
0.5
memlimit:
----
0.7
weight:
----
[ { "nodeName": "rbkci-virtual-kubelet", "weight": 3 }, { "nodeName": "cn-zhangjiakou.vnd-8vb0w6ot6evayqdha0a3", "weight": 1 } ]
Events:  <none>
```

当前调度策略表示：当自建集群的cpu分配率超过50%，或者内存分配率超过70%时，新创建的Pod将会调度到virtual-kubelet节点上，且此时每创建4个pod，将有3个调度到权重为3的vk上，有1个调度到权重为1的vk上。

2.查看调度策略效果

（1）执行以下命令，查看集群节点

```
kubectl get node -o wide
```

预期输出：

```
NAME                                    STATUS   ROLES    AGE    VERSION
10.0.0.179                              Ready    node     46d    v1.21.3
10.0.0.214                              Ready    master   46d    v1.21.3
10.0.0.8                                Ready    master   46d    v1.21.3
10.0.0.83                               Ready    master   46d    v1.21.3
10.0.0.96                               Ready    node     46d    v1.21.3
cn-zhangjiakou.vnd-8vbahalwna5205drcwvr Ready    agent    135m   v1.21.3
rbkci-virtual-kubelet                   Ready    agent    34d    v1.19.3-vk-v1.1.0
```

从以上预期输出可以看出，当前集群有10.0.0.179、10.0.0.96这两个Worker节点和rbkci-virtual-kubelet、cn-zhangjiakou.vnd-8vbahalwna5205drcwvr这两个virtual-kubelet节点。

（2）部署测试Deployment，其yaml如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
      tolerations:   # 如未移除virtual-kubelet上的污点，请增加此污点容忍使vk可以参与调度
      - key: "rbkci-virtual-kubelet.io/provider"
        operator: "Equal"
        value: "kingsoftcloud"
        effect: "NoSchedule"
```

（3）执行以下命令，查看Pod分布节点

```
kubectl get pod -o wide
```

预期输出：

```
NAME                     READY   STATUS    RESTARTS   AGE     IP             NODE
nginx-68c8867f7b-rj9dg   1/1     Running   0          4s      10.244.1.74    10.0.0.179
```

从以上预期输出可以看出，由于cpu或内存分配率没有到达阈值，新创建的Pod调度到了自建集群的Worker节点上

（4）执行以下命令，将Deployment副本数扩容为7

```
kubectl scale deploy nginx --replicas=7
```

再次查看Pod分布节点

```
kubectl get pod -o wide
```

预期输出：

```
NAME                     READY   STATUS    RESTARTS   AGE     IP             NODE
nginx-68c8867f7b-94r4v   1/1     Running   0          3m19s   10.0.0.219     rbkci-virtual-kubelet
nginx-68c8867f7b-9wskx   1/1     Running   0          3m6s    10.0.0.219     cn-zhangjiakou.vnd-8vbahalwna5205drcwvr
nginx-68c8867f7b-pl4xl   1/1     Running   0          3m13s   10.0.0.219     rbkci-virtual-kubelet
nginx-68c8867f7b-pmxjf   1/1     Running   0          3m45s   10.0.0.219     rbkci-virtual-kubelet
nginx-68c8867f7b-rj9dg   1/1     Running   0          3m52s   10.0.0.219     10.0.0.179
nginx-68c8867f7b-sf7xk   1/1     Running   0          5m9s    10.0.0.219     10.0.0.179
nginx-68c8867f7b-twjl2   1/1     Running   0          4m9s    10.0.0.219     10.0.0.179
```

从以上预期输出可以看出，前3个Pod由于cpu或内存分配率没有到达阈值，调度到了自建集群的Worker节点上。通过查看节点的资源分配率，可以看到从第4个Pod起集群中所有pod的cpu request资源/所有node的allocatable资源的值将大于0.5，因此后4个Pod调度到了vk节点上，且根据权重设置，有3个调度到权重为3的vk上，有1个调度到权重为1的vk上。

虚拟节点维度配置自动匹配机型

在对容器实例的性能有特殊需求的场景（如：网络吞吐量、网卡队列数等），可在虚拟节点维度配置自动匹配机型。这样您无需在创建Pod时通过Annotation指定云服务器套餐，系统会根据Pod的request/limit值从虚拟节点配置的机型中，自动匹配相应规格的套餐。

您可以配置多个自动匹配机型，系统会根据配置的顺序依次匹配套餐，例如：当第一个机型没有符合要求的套餐时，会从第二个机型中匹配套餐，以此类推。

前提条件

1. 已在Kubernetes集群中部署虚拟节点，部署方式：KCE集群参考Kubernetes集群对接KCI，自建集群参考自建Kubernetes集群中对接KCI。
2. 使用本功能需要将virtual-kubelet的镜像升级至不低于v1.3.6的版本，以确保支持虚拟节点维度配置自动匹配机型的环境变量。

配置虚拟节点

执行如下命令，对虚拟节点对应的Deployment进行编辑：

```
kubectl -n kube-system edit deployment rbkci-virtual-kubelet
```

配置KCI_INSTANCE_FAMILY的环境变量，示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rbkci-virtual-kubelet
  namespace: kube-system
  labels:
    k8s-app: rbkci-virtual-kubelet
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: rbkci-virtual-kubelet
  template:
    metadata:
      name: rbkci-virtual-kubelet
      labels:
        k8s-app: rbkci-virtual-kubelet
    spec:
      serviceAccountName: virtual-kubelet-sa
      containers:
        - name: virtual-kubelet
          image: hub.kce.ksyun.com/ksyun/rbkci-virtual-kubelet:v1.3.6 # 需使用不低于v1.3.6版本的virtual-kubelet镜像
          args:
            - --nodename=rbkci-virtual-kubelet
            - --cluster-dns=10.254.0.10
            - --cluster-domain=cluster.local
            - --kcilet-kubeconfig-path=/root/.kube/config
            - --enable-node-lease
            - --kube-proxy-enable
            - --base-system-disk-size=50
          imagePullPolicy: Always
          env:
            - name: VKUBELET_POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: TEMP_AKSK_CM
              value: user-temp-aksk
            - name: KCI_CLUSTER_ID
              value: ${cluster_id}
            - name: KCI_SUBNET_ID
              value: ${subnet_id}
            - name: KCI_SECURITY_GROUP_IDS
              value: ${security_group_ids}
            # 配置虚拟节点的自动匹配机型，配置时请根据想要使用机型的优先级，按顺序设置多个机型
            - name: KCI_INSTANCE_FAMILY
              value: S6,I3
          volumeMounts:
            - mountPath: /root/.kube
              name: kubeconfig
            - mountPath: /var/log/kci-virtual-kubelet
              name: kci-provider-log
      volumes:
        - name: kubeconfig
          secret:
            secretName: rbkci-kubeconfig-secret
        - name: kci-provider-log
          hostPath:
            path: /var/log/kci-virtual-kubelet
```

使用示例

按照上一章节中的示例配置了虚拟节点之后，在该虚拟节点上创建的容器实例将按照顺序从S6、I3两个机型中自动匹配云服务器套餐，使用示例如下：

验证虚拟节点维度配置的自动匹配机型是否生效

1.部署测试Deployment，其yaml如下：

注：

1. 金山云容器服务会根据Pod的request/limit值自动规整KCI Pod的规格，详细计算方法请参考指定KCI Pod规格。
2. 您也可以通过k8s.ksyun.com/kci-instance-cpu和k8s.ksyun.com/kci-instance-memory这两个Annotation指定KCI Pod的规格。
3. 如您在Pod中通过k8s.ksyun.com/kci-instance-type的Annotation指定了云服务器套餐，则其优先级高于虚拟节点维度配置的自动匹配机型，创建的容器实例将使用Annotation中指定的云服务器套餐。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        resources:
          limits:  # 通过limit配置容器的cpu和内存限制
            cpu: "4"
            memory: "8Gi"
      nodeName: rbkci-virtual-kubelet  # 指定nodeName将pod调度到虚拟节点上
```

2.调用DescribeContainerGroup接口，查询容器实例的规格和云服务器套餐：

```
{
...
    "ContainerGroups": [
        {
            "ContainerGroupId": "e8059037-4693-4a4d-b0ef-0ee3323a17b8",
            "ContainerGroupName": "default-nginx-6ff6c84699-7jgv4",
            "AvailabilityZone": "cn-beijing-6a",
            "ChargeType": "HourlyInstantSettlement",
            "NetworkInterfaceAttributes": [
                ...
            ],
            "Cpu": 4,  # 容器实例的CPU规格
            "Memory": 8,  # 容器实例的内存规格
            "Gpu": null,
            "CreateTime": "2023-02-15 09:54:20",
            "RestartPolicy": null,
            "Status": "Running",
            "SucceededTime": null,
            "Volumes": null,
            "DnsConfig": null,
            "Containers": null,
            "Events": null,
            "Labels": [
                ...
            ],
            "HostAliases": null,
            "IngressBandwidth": null,
            "EgressBandwidth": null,
            "InstanceType": "S6.4B",  # 容器实例的云服务器套餐
            "KciType": "RBKCI",
            "KciMode": "CLUSTER",
            "ProjectId": 0,
            "RetainIp": null,
            "RetainIpHours": null
        }
    ]
...
}
```

可以看到，容器实例匹配到了S6机型下实例规格为4C8G的S6.4B套餐，虚拟节点维度设置的自动匹配机型生效。

验证自动匹配机型的优先级

1.部署测试Deployment，指定其系统盘类型为Local_SSD(本地SSD硬盘)。由于S6机型不支持Local_SSD类型的系统盘，而I3机型支持该类型的系统盘，因此预期容器实例将匹配到I3机型的云服务器套餐。测试Deployment的yaml如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        k8s.ksyun.com/kci-base-system-disk-type: Local_SSD  # 指定系统盘类型为Local_SSD
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        resources:
          limits:  # 通过limit配置容器的cpu和内存限制
            cpu: "4"
            memory: "8Gi"
      nodeName: rbkci-virtual-kubelet  # 指定nodeName将pod调度到虚拟节点上
```

2.调用DescribeContainerGroup接口，查询容器实例的规格和云服务器套餐：

```
{
...
    "ContainerGroups": [
        {
            "ContainerGroupId": "c5ee7b0c-6d17-4065-8a7c-2b0dc362021d",
            "ContainerGroupName": "default-nginx-869c5f7fd4-6jxps",
            "AvailabilityZone": "cn-beijing-6a",
            "ChargeType": "HourlyInstantSettlement",
            "NetworkInterfaceAttributes": [
                ...
            ],
            "Cpu": 4,  # 容器实例的CPU规格
            "Memory": 8,  # 容器实例的内存规格
            "Gpu": null,
            "CreateTime": "2023-02-15 10:42:37",
            "RestartPolicy": null,
            "Status": "Running",
            "SucceededTime": null,
            "Volumes": null,
            "DnsConfig": null,
            "Containers": null,
            "Events": null,
            "Labels": [
                ...
            ],
            "HostAliases": null,
            "IngressBandwidth": null,
            "EgressBandwidth": null,
            "InstanceType": "I3.4B",  # 容器实例的云服务器套餐
            "KciType": "RBKCI",
            "KciMode": "CLUSTER",
            "ProjectId": 0,
            "RetainIp": null,
            "RetainIpHours": null
        }
    ]
}
```

可以看到，容器实例匹配到了I3机型下实例规格为4C8G的I3.4B套餐，自动匹配机型的优先级生效：虚拟节点维度按S6、I3的顺序配置了自动匹配机型，当S6机型中没有符合要求的套餐时，会从I3机型中匹配套餐。

按项目管理容器实例

项目功能用于按项目管理云资源，如您需要通过分项目进行财务核算、权限控制等，可以将容器实例分配到各个项目中来进行管理。

前提条件

1. 已在Kubernetes集群中部署虚拟节点，部署方式：KCE集群参考Kubernetes集群对接KCI，自建集群参考自建Kubernetes集群中对接KCI。
2. 使用本功能需要将virtual-kubelet的镜像升级至不低于v1.3.2的版本，以确保支持容器实例指定项目ID的Annotation和环境变量。

容器实例按项目出账

对于需要分项目进行财务核算的场景，您可以在创建容器实例时将容器实例分配至指定项目，后续即可通过项目名称筛选容器实例的账单。

步骤1：查看项目ID

1. 登录项目管理控制台。

2. 记录项目ID。

步骤2：指定项目ID创建容器实例

创建容器实例并指定项目，配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-project-id: "106123"  # 指定项目ID
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: rbkci-virtual-kubelet  # 指定nodeName将pod调度到虚拟节点上
```

步骤3：按项目查看账单

1. 登录费用中心控制台，查看账单明细。

2. 按项目名称筛选容器实例的账单。

容器实例按项目进行权限控制

通过项目制可以对容器实例的权限进行控制，子用户只能在所加入的项目下创建和管理容器实例，不能在未加入的项目下创建容器实例。

步骤1：将子用户加入项目

1. 登录项目管理控制台。

2. 点击**成员管理**。

3. 点击**添加成员**，将子用户加入项目中。

步骤2：配置虚拟节点使用子用户的AK/SK

1. 参考获取AKSK文档，获取子账号的AK/SK。

2. 执行如下命令，对虚拟节点对应的Deployment进行编辑：

```
kubectl -n kube-system edit deployment rbkci-virtual-kubelet
```

3. 配置KCI_ACCESS_KEY、KCI_SECRET_KEY两个环境变量。

```
...
  env:
        - name: VKUBELET_POD_IP
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: status.podIP
        - name: KCI_ACCESS_KEY
          value: AKL************  # 子账号的AK
        - name: KCI_SECRET_KEY
          value: OEL******************  # 子账号的SK
        - name: KCI_REGION
          value: cn-beijing-6
        - name: KCI_CLUSTER_ID
          value: ${cluster_id}
        - name: KCI_SUBNET_ID
          value: ${subnet_id}
        - name: KCI_SECURITY_GROUP_IDS
          value: ${security_group_ids}
...
```

注：

1. 如虚拟节点之前配置了TEMP_AKSK_CM的环境变量，请删掉该环境变量，否则虚拟节点仍将使用集群中的临时AK/SK，配置的子用户AK/SK不会生效。
2. 如之前未配置过KCI_REGION的环境变量，则需配置此环境变量以指定虚拟节点所管理容器实例的地域，容器实例支持的地域请参考支持地域。

步骤3：管理已加入项目下的容器实例

1. 创建容器实例并指定子用户已加入项目的ID，配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-project-id: "106123"  # 指定项目ID
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: rbkci-virtual-kubelet  # 指定nodeName将pod调度到虚拟节点上
```

2. 容器实例创建成功，并可以对容器实例进行修改、删除等操作。

步骤4：在未加入项目下创建容器实例失败

1. 创建容器实例并指定子用户未加入的项目ID，配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-project-id: "106523"  # 指定子用户未加入的项目ID
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: rbkci-virtual-kubelet  # 指定nodeName将pod调度到虚拟节点上
```

2.创建容器实例失败，出现下图所示报错提示没有权限。