基于VK实现多集群管理

王琼 (YY直播 SRE)

集群管理

集群接管
节点上下线
离线资源池管理
资源池管理
驱逐

持续集成

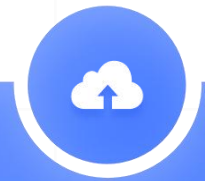
代码构建
镜像构建

服务发布

资源大小选择
资源池选择
调度策略选择
固定ip支持
回滚
灰度
配置管理

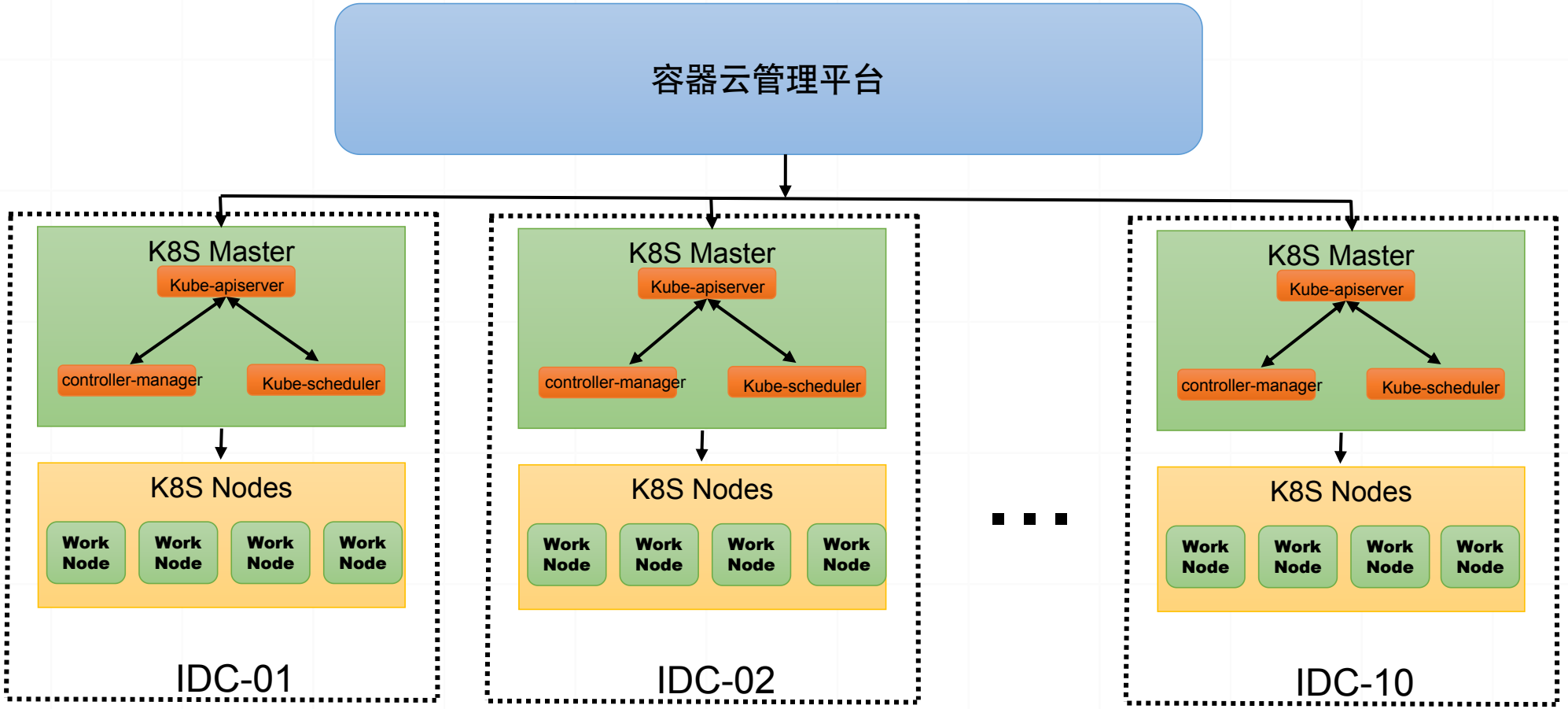
API提供

提供统一的部署接口
提供查询功能



报表查询

容器化接入率
集群资源使用详情
集群监控


容器云管理平台
K8S Master
Kube-apiserver
controller-manager
Kube-scheduler
K8S Nodes
Work Node
Work Node
Work Node
Work Node
IDC-01
K8S Master
Kube-apiserver
controller-manager
Kube-scheduler
K8S Nodes
Work Node
Work Node
Work Node
Work Node
IDC-02
K8S Master
Kube-apiserver
controller-manager
Kube-scheduler
K8S Nodes
Work Node
Work Node
Work Node
Work Node
IDC-10

单集群故障

单集群越来越大，降低单集群故障对业务的影响

集群平滑升级

通过新增集群实现集群的平滑升级

第三方集群接入

业务无需代码改动即可发布服务至第三方集群

边缘机房区域自治

防止前端机房计算节点与控制平面失联

公有云资源接入

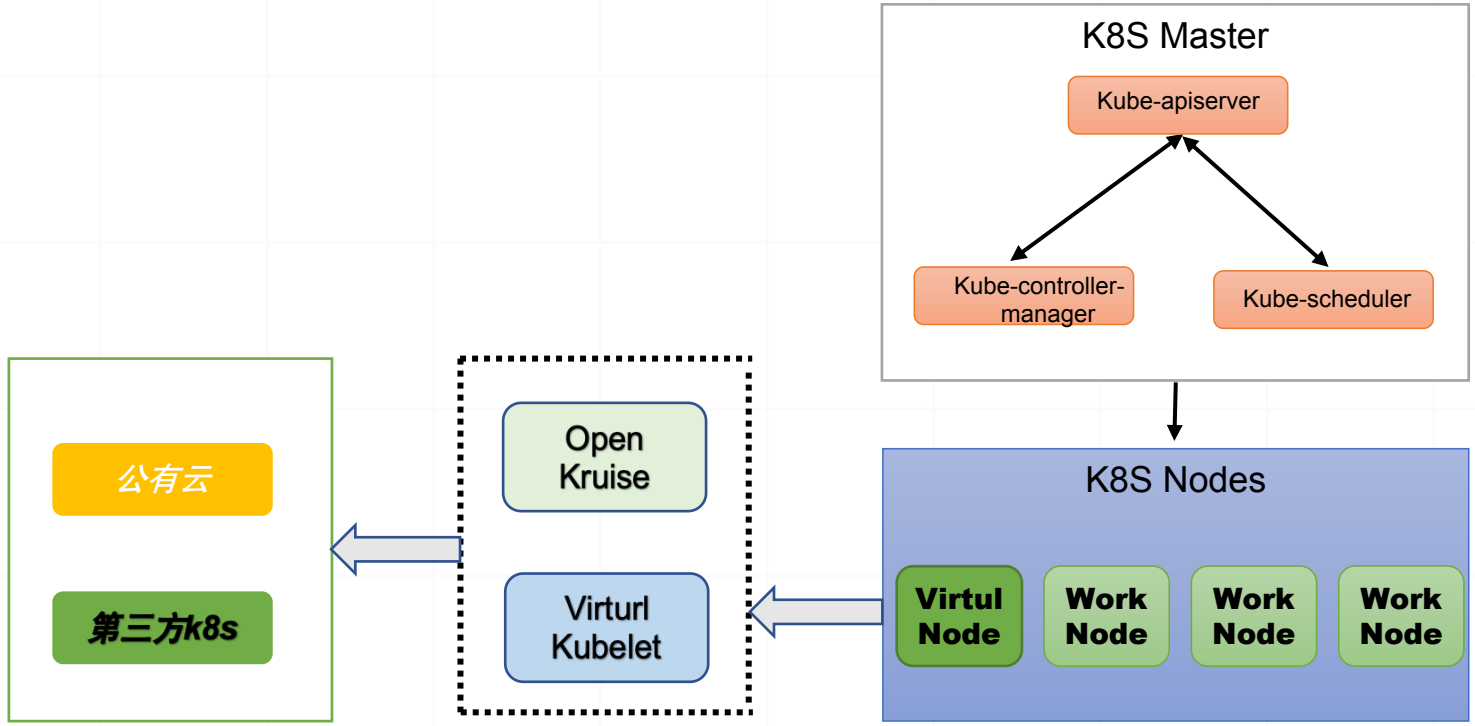
接入公有云弹性资源实现快速弹性功能

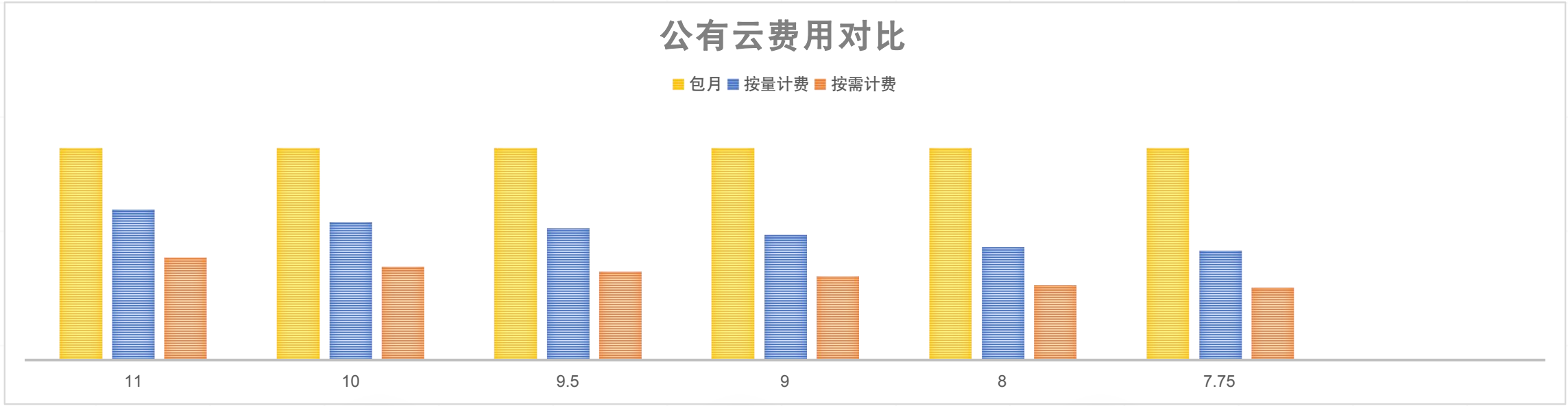
解决的问题：

依靠云资源可秒级弹性扩容
无需预留离线资源
支持优先部署到自建机房，资源不足时部署到弹性机房
快速接入第三方集群

成本优化：

云厂商即用即计费，优化计算资源成本，按需计费，不使用不计费
抢占式实例成本优化明显





01

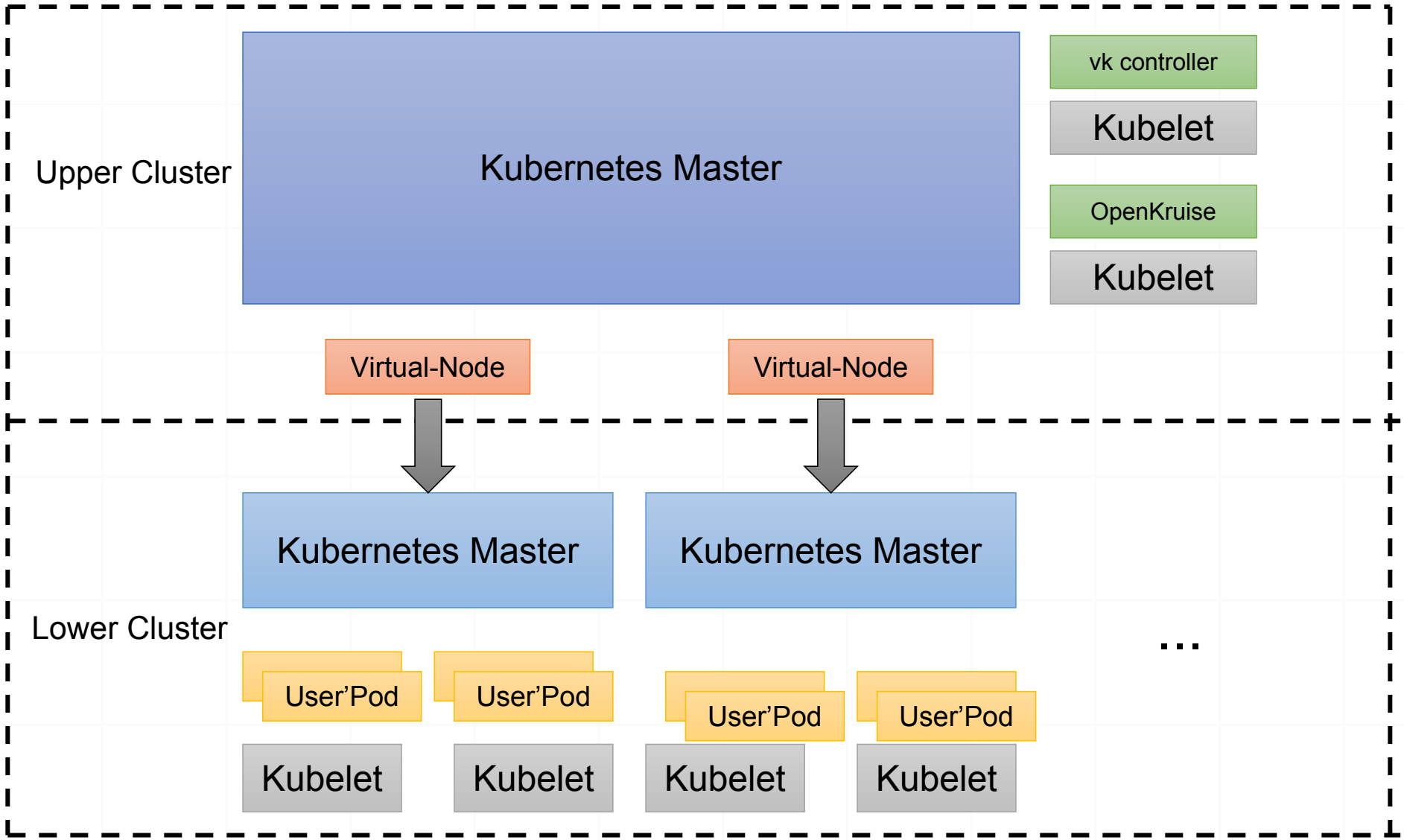
节点弹性伸缩在定时扩缩容节点满足目前的需求，扩容需要5分钟左右，在非定时伸缩情况下需要15分钟完成扩容

02

弹性伸缩可实现秒级扩容，与正常pod实例无差别

03

弹性伸缩比起节点弹性伸缩成本上表现更优，可节省更多的计算机缘成本


Upper Cluster
Kubernetes Master
vk controller
Kubelet
OpenKruise
Kubelet
Virtual-Node
Virtual-Node
Lower Cluster
Kubernetes Master
Kubernetes Master
...
User'Pod
User'Pod
User'Pod
User'Pod
Kubelet
Kubelet
Kubelet
Kubelet

Virturl Kubelet

- 虚拟 WorkerNode

  Liqo Provider

  Tensile Kube Provider

OpenKruise

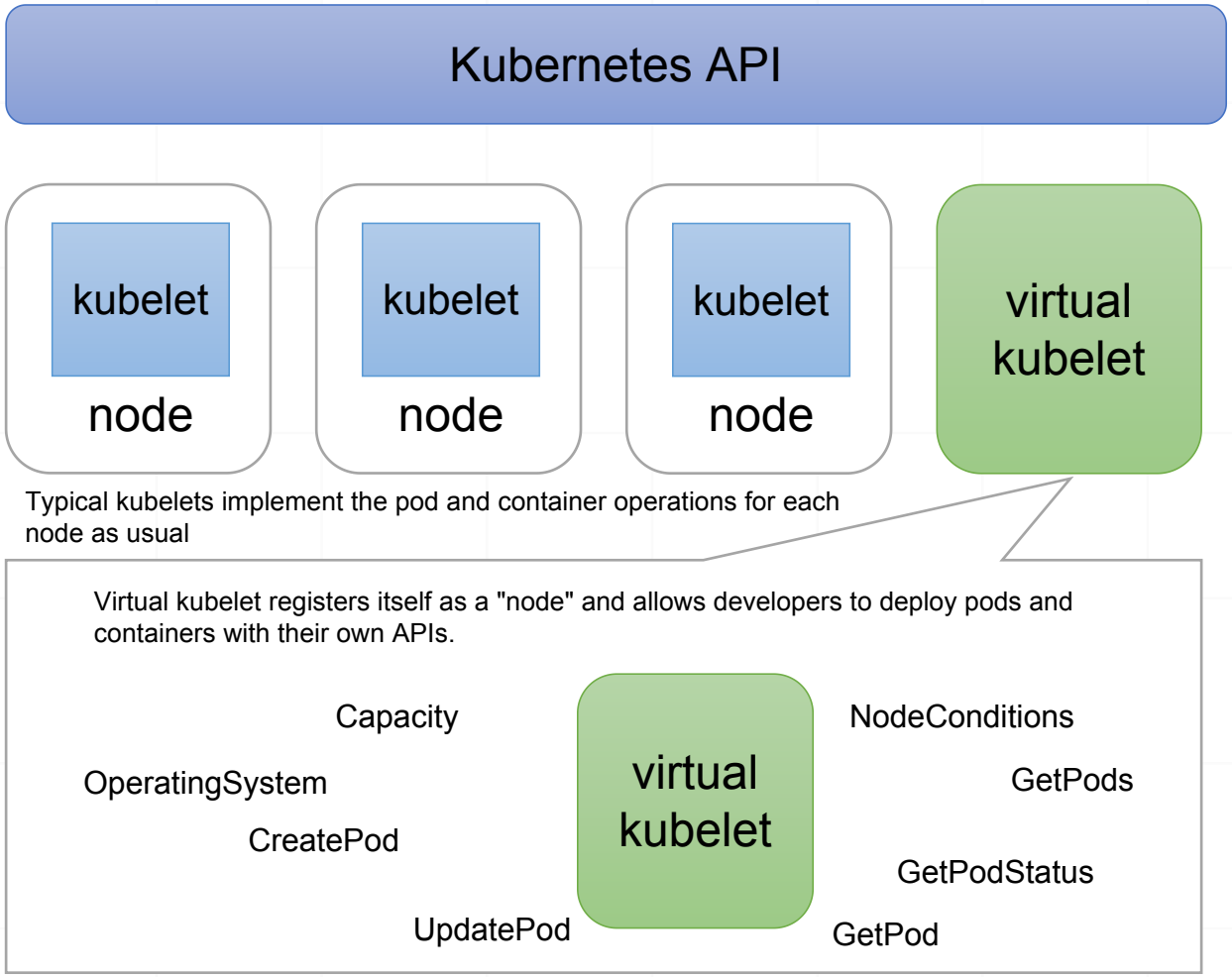
- WorkloadSpread

  能够将 Workload 的 Pod 按一定规则分布到不同类型的 Node 节点上，赋予单一 Workload 多区域部署和弹性部署的能力

- SidecarSet

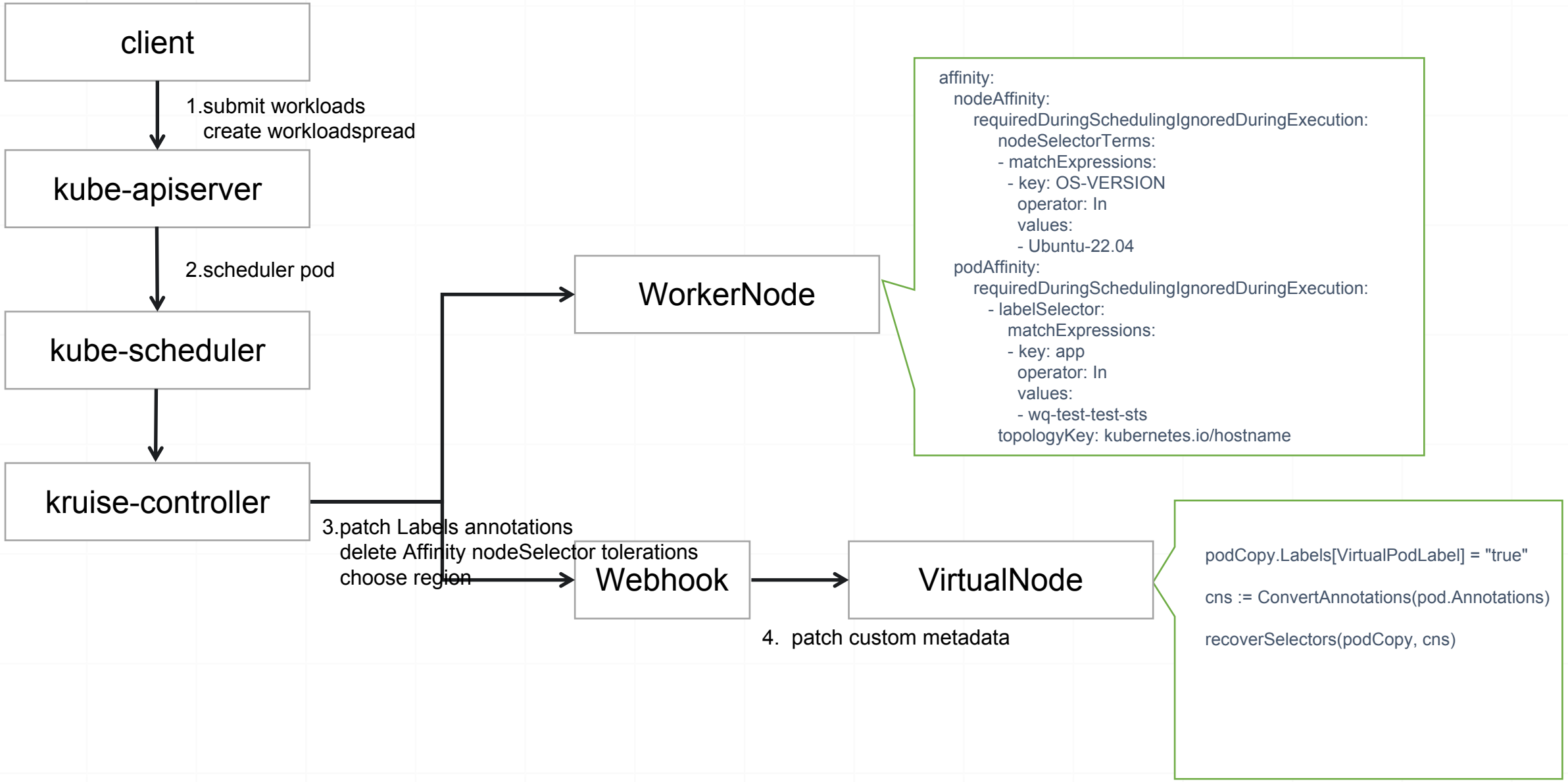
  自动注入 Container 至 Pod

- Webhook



功能：

- create, delete and update pods,cm,secrets
- container logs, exec, and metrics
- get pod, pods and pod status
- capacity
- node addresses, node capacity, node daemon endpoints
- operating system
- bring your own virtual network


client
1.submit workloads
create workloadspread
kube-apiserver
2.scheduler pod
kube-scheduler
kruise-controller
3.patch Labels annotations
delete Affinity nodeSelector tolerations
choose region
WorkerNode
Webhook
VirtualNode
4. patch custom metadata
affinity:
nodeAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
nodeSelectorTerms:
- matchExpressions:
- key: OS-VERSION
operator: In
values:
- Ubuntu-22.04
podAffinity:
requiredDuringSchedulingIgnoredDuringExecution:
- labelSelector:
matchExpressions:
- key: app
operator: In
values:
- wq-test-test-sts
topologyKey: kubernetes.io/hostname
podCopy.Labels[VirtualPodLabel] = "true"
cns := ConvertAnnotations(pod.Annotations)
recoverSelectors(podCopy, cns)

WorkloadSpread

能够将 Workload 的 Pod 按一定规则分布到不同类型的 Node 节点上，赋予单一 Workload 多区域部署和弹性部署的能力

优先部署到自建机房，资源不足时部署到 VK

优先部署固定数量个 Pod 到自建机房，其余到 VK

```
apiVersion: apps.kruise.io/v1alpha1
kind: WorkloadSpread
metadata:
  name: workloadspread-demo
spec:
  scheduleStrategy:
    adaptive:
      rescheduleCriticalSeconds: 30
    type: Adaptive
  subsets:
  - name: common
    maxReplicas: 50%
    patch:
      metadata:
        annotations:
          subset: common
    requiredNodeSelectorTerm:
      matchExpressions:
      - key: node-role.kubernetes.io/WorkerNode
        operator: In
        values:
        - "true"
    tolerations:
    - effect: NoSchedule
      key: unscheduer
      operator: Equal
      value: "true"
  - name: local-virtual-kubelet
    patch:
      metadata:
        annotations:
          clusterSelector: '{"tolerations":[{"key":"node.kubernetes.io/not-
ready","operator":"Exists","effect":"NoExecute"},{"key":"node.kubernetes.io/unreachable","operator":"Exists","effect":"No
Execute"},{"key":"test","operator":"Exists","effect":"NoExecute"},{"key":"test1","operator":"Exists","effect":"NoExecute"},{"
key":"test2","operator":"Exists","effect":"NoExecute"},{"key":"test3","operator":"Exists","effect":"NoExecute"}]}'
          subset: local-virtual-kubelet
      spec:
        affinity:
          nodeAffinity: null
          podAntiAffinity: null
        nodeSelector: null
    requiredNodeSelectorTerm:
      matchExpressions:
      - key: node-role.kubernetes.io/local-virtual-kubelet
        operator: In
        values:
        - "true"
```

```
func recoverSelectors(pod *corev1.Pod, cns *ClustersNodeSelection) {
	if cns != nil {
		pod.Spec.NodeSelector = cns.NodeSelector
		pod.Spec.Tolerations = cns.Tolerations
		if pod.Spec.Affinity == nil {
			pod.Spec.Affinity = cns.Affinity
		} else {
			if cns.Affinity != nil && cns.Affinity.NodeAffinity != nil {
				if pod.Spec.Affinity.NodeAffinity != nil {
					pod.Spec.Affinity.NodeAffinity.RequiredDuringSchedulingIgnoredDuringExecution =
cns.Affinity.NodeAffinity.RequiredDuringSchedulingIgnoredDuringExecution
				} else {
					pod.Spec.Affinity.NodeAffinity = cns.Affinity.NodeAffinity
				}
			} else {
				pod.Spec.Affinity.NodeAffinity = nil
			}
			if cns.Affinity != nil {
				if cns.Affinity.PodAffinity != nil {
					pod.Spec.Affinity.PodAffinity = cns.Affinity.PodAffinity
				}
				if cns.Affinity.PodAntiAffinity != nil {
					pod.Spec.Affinity.PodAntiAffinity = cns.Affinity.PodAntiAffinity
				}
			}
		}
	} else {
		pod.Spec.NodeSelector = nil
		pod.Spec.Tolerations = nil
		if pod.Spec.Affinity != nil && pod.Spec.Affinity.NodeAffinity != nil {
			pod.Spec.Affinity.NodeAffinity.RequiredDuringSchedulingIgnoredDuringExecution = nil
		}
	}
	if pod.Spec.Affinity != nil {
		if pod.Spec.Affinity.NodeAffinity != nil {
			if pod.Spec.Affinity.NodeAffinity.RequiredDuringSchedulingIgnoredDuringExecution == nil &&
				pod.Spec.Affinity.NodeAffinity.PreferredDuringSchedulingIgnoredDuringExecution == nil {
				pod.Spec.Affinity.NodeAffinity = nil
			}
		}
		if pod.Spec.Affinity.NodeAffinity == nil && pod.Spec.Affinity.PodAffinity == nil &&
			pod.Spec.Affinity.PodAntiAffinity == nil {
			pod.Spec.Affinity = nil
		}
	}
}
```

SidecarSet

自动注入sidecar,可以将日志采集、监控上报等agent,通过sidecar的形式自动注入到pod

```yaml
# sidecarset.yaml
apiVersion: apps.kruise.io/v1alpha1
kind: SidecarSet
metadata:
  name: test-sidecarset
spec:
  selector:
    matchLabels:
      vk: true
  updateStrategy:
    type: RollingUpdate
    maxUnavailable: 1
  containers:
  - name: sidecar1
    image: centos:6.7
    command: ["sleep", "999d"] # do nothing at all
    volumeMounts:
    - name: log-volume
      mountPath: /var/log
  volumes: # this field will be merged into pod.spec.volumes
  - name: log-volume
    emptyDir: {}
```


01
Service
02
PVC
03
CRD

感谢观看

云原生社区
Cloud Native Community