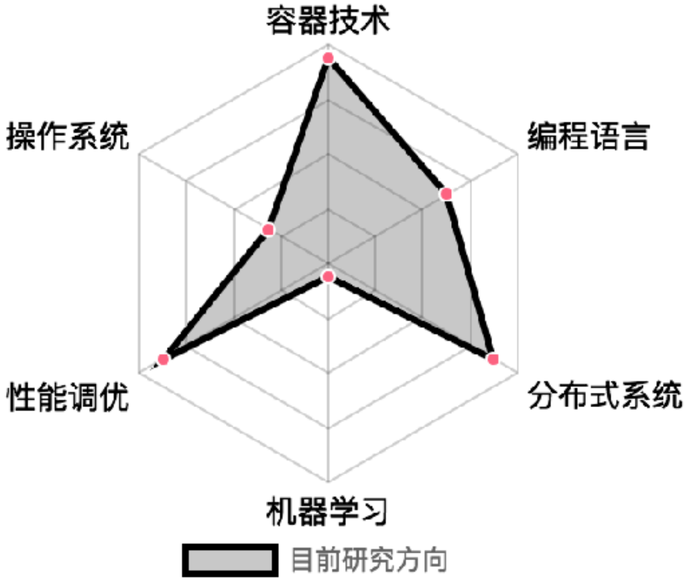
聊聊k8s自动伸缩那些事

莫源

个人简介



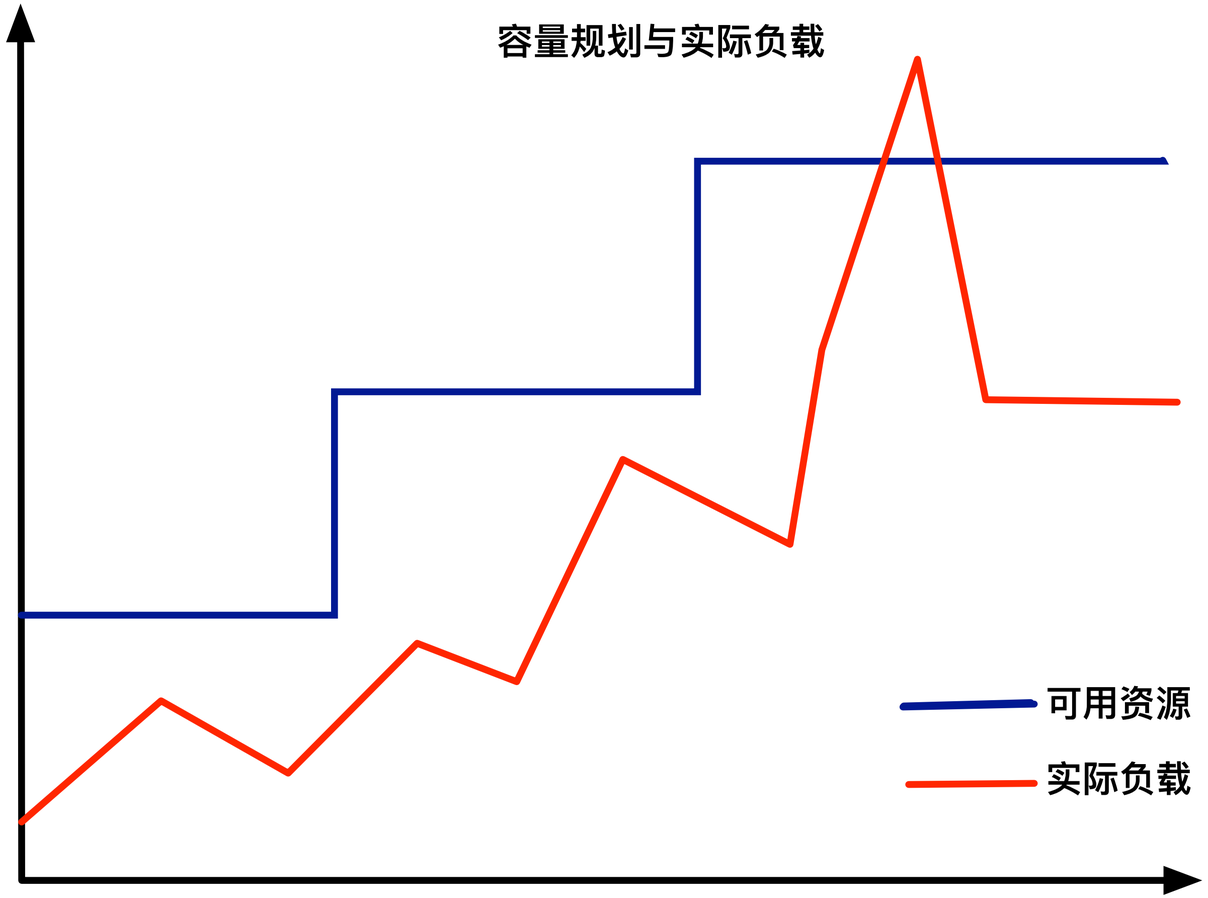
Microsoft 图像处理与虚拟现实方向

阿里云 持续交付与容器相关领域方向

课程介绍

- 弹性伸缩概念在容器场景下的延伸
- Kubernetes弹性伸缩设计思路解析
- Kubernetes弹性伸缩相关组件详解
  - HPA的介绍、原理与演进
  - Cluster-Autoscaler的介绍、原理与演进
  - VPA的设计原理
  - Cluster-Proportional-Autoscaler的设计原理

冗余资源与峰值流量的博弈?

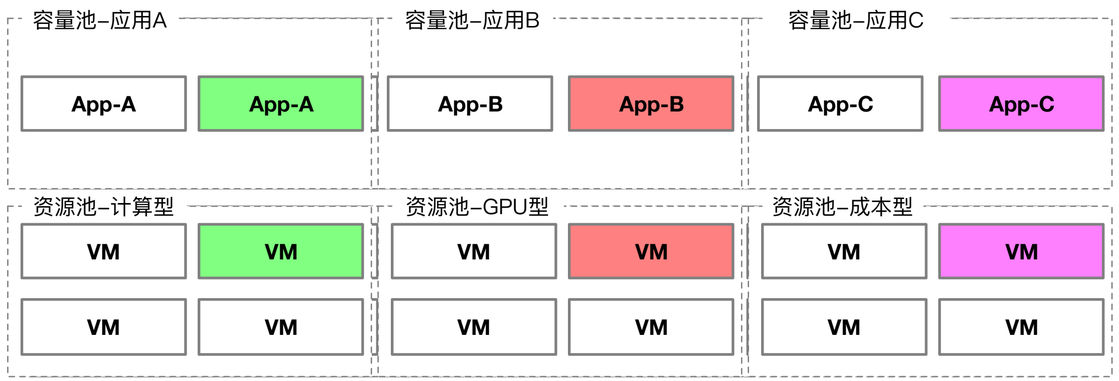


不是所有的业务都存在峰值流量，越来越细分的业务形态带来更多成本节省与可用性之间的挑战。

1. 在线负载型 – 微服务、网站、API
2. 离线任务型 – 离线计算、机器学习/深度学习
3. 定时任务型 – 定时批量计算
4. 特殊场景型 – 闲时计算、自定义伸缩

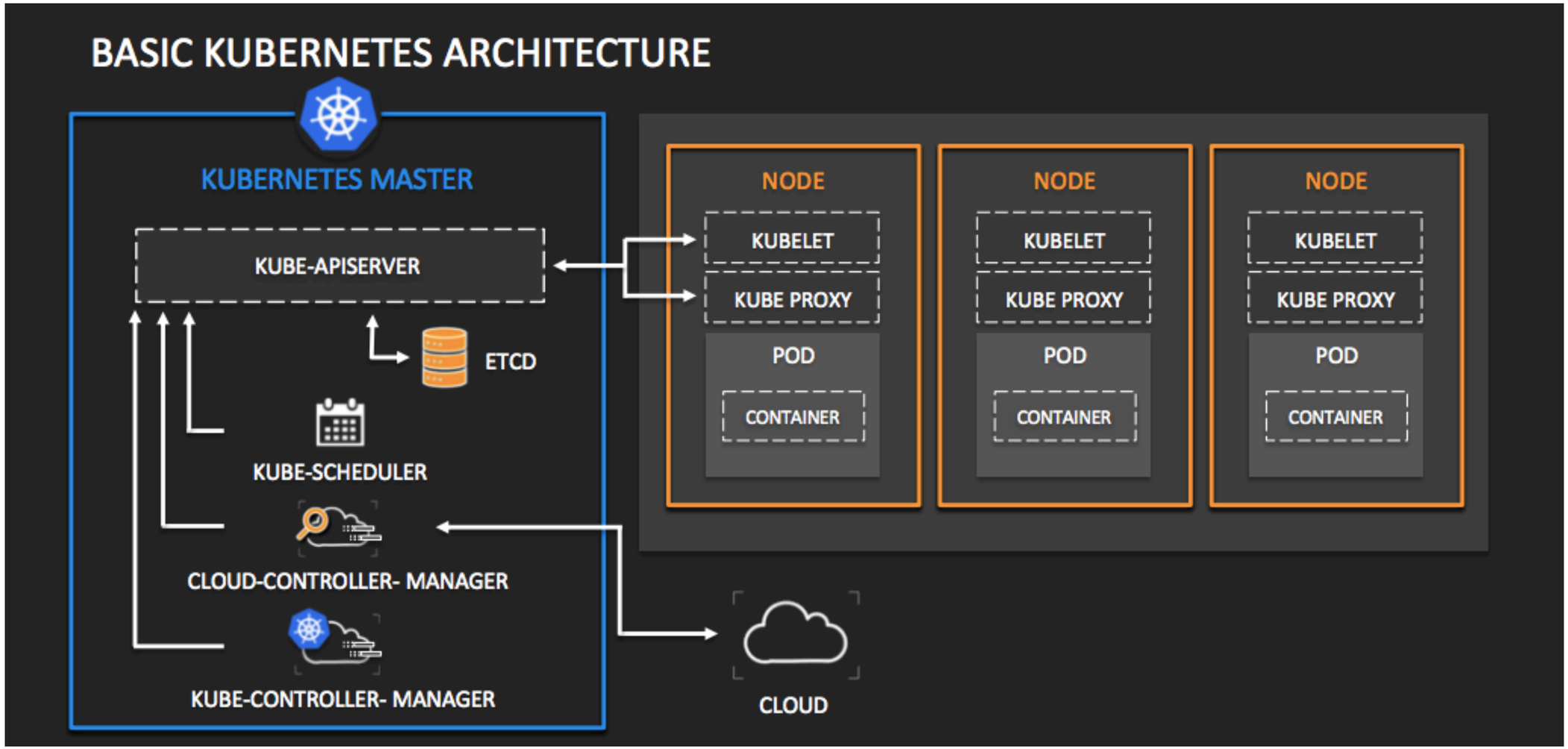
不同类型的负载对于弹性伸缩的要求有所不同，在线负载对弹出时间敏感，离线任务对价格敏感，定时任务对调度敏感，特殊场景对弹出稳定性敏感。

“调度”与“资源” – 弹性伸缩的二象限

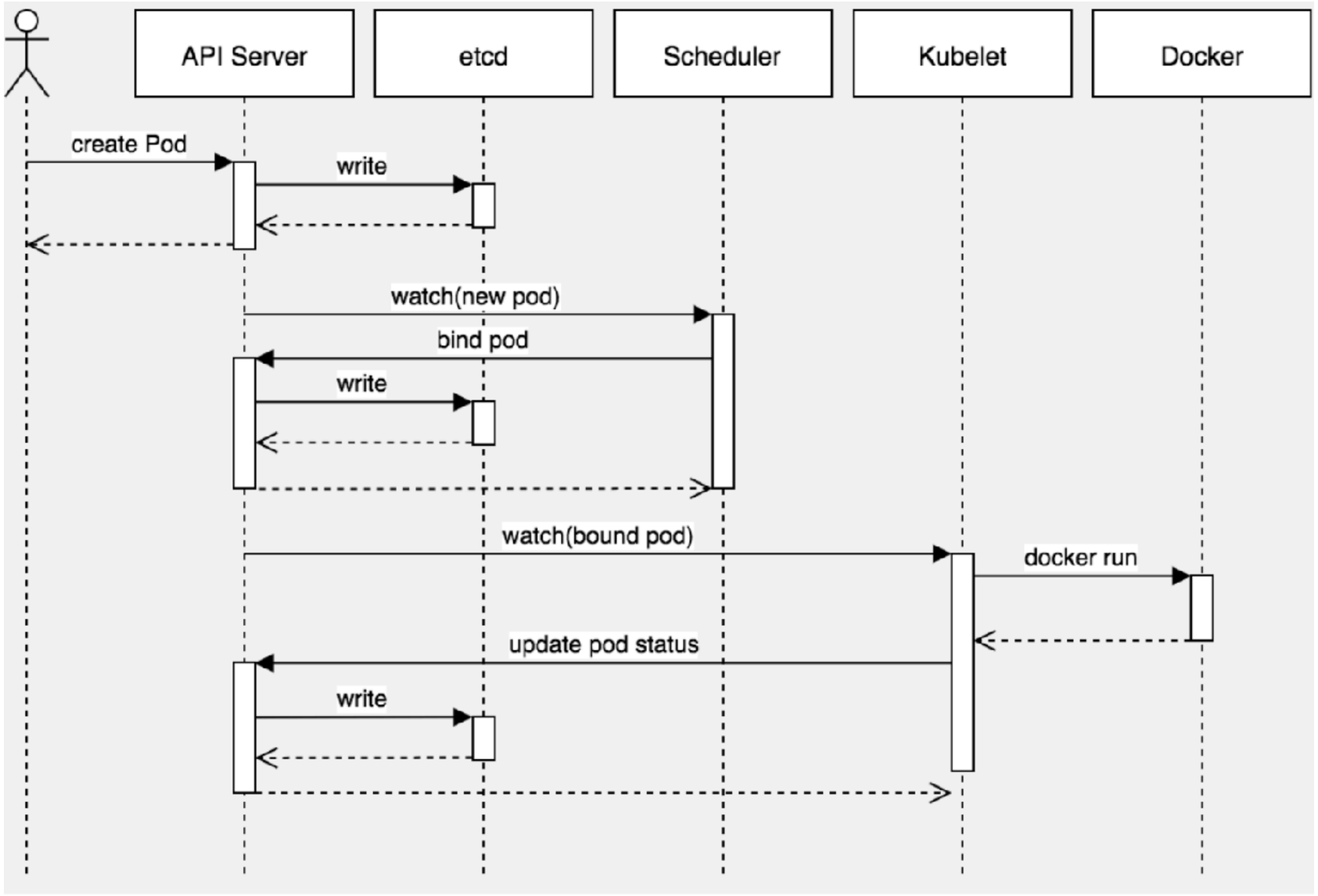


“调度”在集群容量充裕的前提下提供尽可能的弹性
集群容量不符合场景的情况下需要弹性调整“资源”

Kubernetes的架构设计理念–架构原理



Kubernetes的架构设计理念–组件交互



Kubernetes的架构设计理念-设计原则

-面向资源简化模型（go-restful）
所有在kubernetes中操作的实体都可以用资源进行抽象，所有的资源都有restful的API与之对应。

-异步动作保证性能（informers）
所有依赖资源的组件都通过异步进行监听，具体的执行由各自的消费者决定频度。

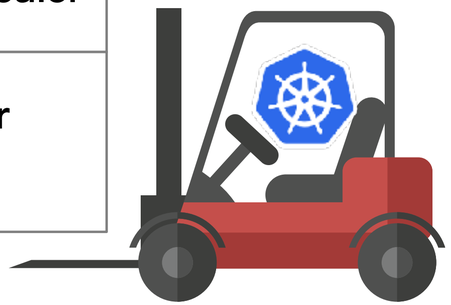
-状态机提供状态基线 （etcd）
所有的信息流都通过期望、实施、反馈的机制存储在etcd中，数据即状态。

-反馈机制保证状态
informers中可以实现定期的sync，可以通过sync来处理中间状态。

-组件松耦合可插拔
组件之间通信要么是通过APIServer进行中转，要么是通过APIGroup进行解耦，组件之间没有强依赖关系，部分组件自带熔断器。

Kubernetes弹性伸缩矩阵

	Nodes	Pods
Horizontal	cluster autoscaler	HPA cluster proportional autoscaler
Vertical	none	vertical pod autoscaler addon resizer

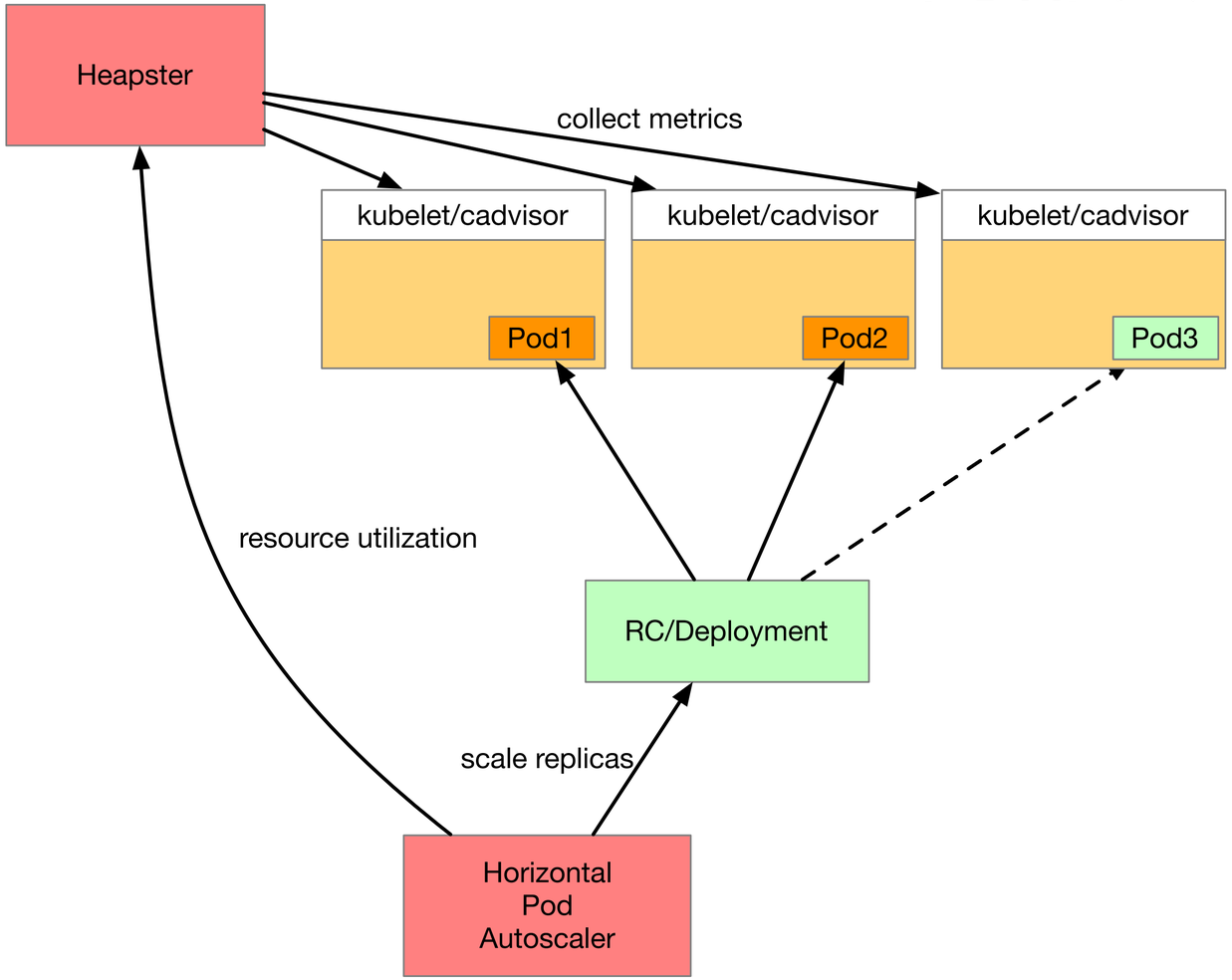


HPA – 容量规划与资源水位

HPA的弹性伸缩基于负载的Request值以及实际利用率来进行计算的，当资源利用率超过阈值，即由ReplicationController进行replicas的增加，若低于阈值，则进行replicas的减少。

$$\left\lceil \frac{\sum_{i \in \text{Pods}} \text{Usage}_i}{\text{Target}} \right\rceil$$

HPA – 内部实现原理



噪声处理

Starting或者Stopping Pod存在时会直接进入下一个计算周期

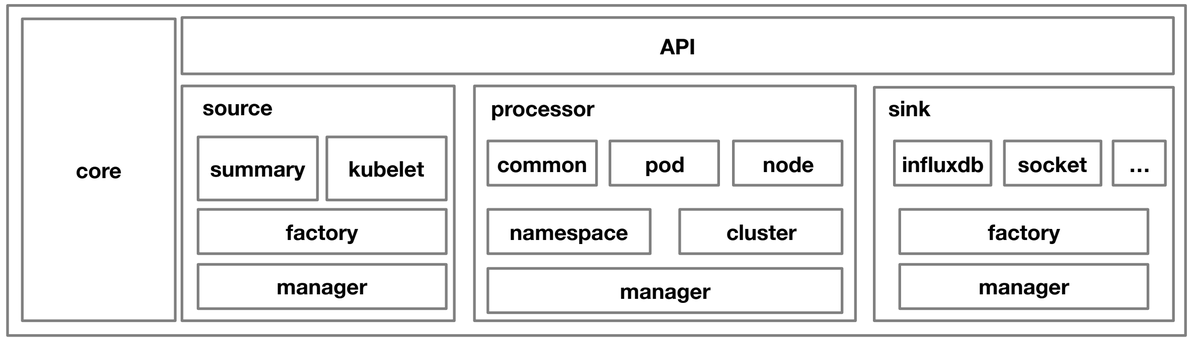
冷却周期

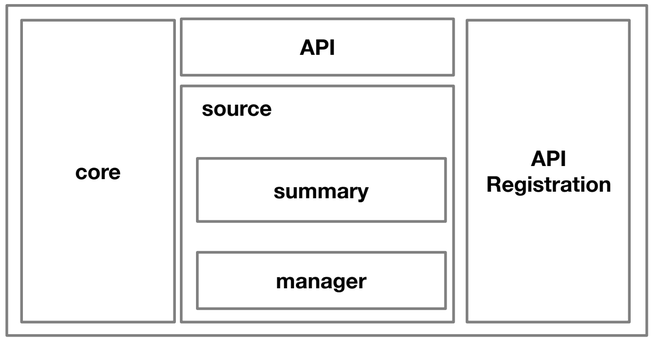
扩容冷却时间3分钟，缩容冷却时间5分钟

边界值计算△

通过10%的△缓冲阈值防止扩缩容震荡

HPA – 监控组件的演进促进HPA演进





Heapster：

必选组件，集群无法解耦与替换，所有的client依赖直接Service Proxy的调用，扩展性差，集成Sink较多，但maintainer不活跃

Metrics-Server (0.1-0.2)：

采集能力完全等同Heapster，剥离所有Sink，增加全新的metrics api。

Metrics-Server (0.3):

采集能力精简，数据处理精简，Sink精简到只保留接口，70%以上代码重写，基本无法兼容原有的API。0.3版本最重要的意义是确定了Metrics-Server的定位，以及Prometheus的未来地位。

HPA – Metrics Server的API注册方式

```yaml
apiVersion: apiregistration.k8s.io/v1beta1
kind: APIService
metadata:
  name: v1beta1.metrics.k8s.io
spec:
  service:
    name: metrics-server
    namespace: kube-system
  group: metrics.k8s.io
  version: v1beta1
  insecureSkipTLSVerify: true
  groupPriorityMinimum: 100
  versionPriority: 100
```

```
[root@iZwz99zrzfnfq8wllk0dvcZ ~]# kubectl get apiservice
NAME                                   AGE
v1.                                    120d
v1.apps                                120d
v1.authentication.k8s.io               120d
v1.authorization.k8s.io                120d
v1.autoscaling                         120d
v1.batch                               120d
v1.networking.k8s.io                   120d
v1.rbac.authorization.k8s.io           120d
v1.storage.k8s.io                      120d
v1alpha1.admissionregistration.k8s.io  120d
v1beta1.admissionregistration.k8s.io   120d
v1beta1.alicloud.com                   94d
v1beta1.apiextensions.k8s.io           120d
v1beta1.apps                           120d
v1beta1.authentication.k8s.io          120d
v1beta1.authorization.k8s.io           120d
v1beta1.batch                          120d
v1beta1.certificates.k8s.io            120d
v1beta1.events.k8s.io                  120d
v1beta1.extensions                     120d
v1beta1.metrics.k8s.io                 1h
v1beta1.policy                         120d
v1beta1.rbac.authorization.k8s.io      120d
v1beta1.servicecatalog.k8s.io          27d
v1beta1.storage.k8s.io                 120d
v1beta2.apps                           120d
v2beta1.autoscaling                    120d
```

Metrics-Server的可替代性?

HPA – API版本演进

```
metricsClient := metrics.NewHeapsterMetricsClient(
    hpaClient,
    metrics.DefaultHeapsterNamespace,
    metrics.DefaultHeapsterScheme,
    metrics.DefaultHeapsterService,
    metrics.DefaultHeapsterPort,
)
```

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        kind: AverageUtilization
        averageUtilization: 50
  - type: Pods
    pods:
      metric:
        name: packets-per-second
      targetAverageValue: 1k
```

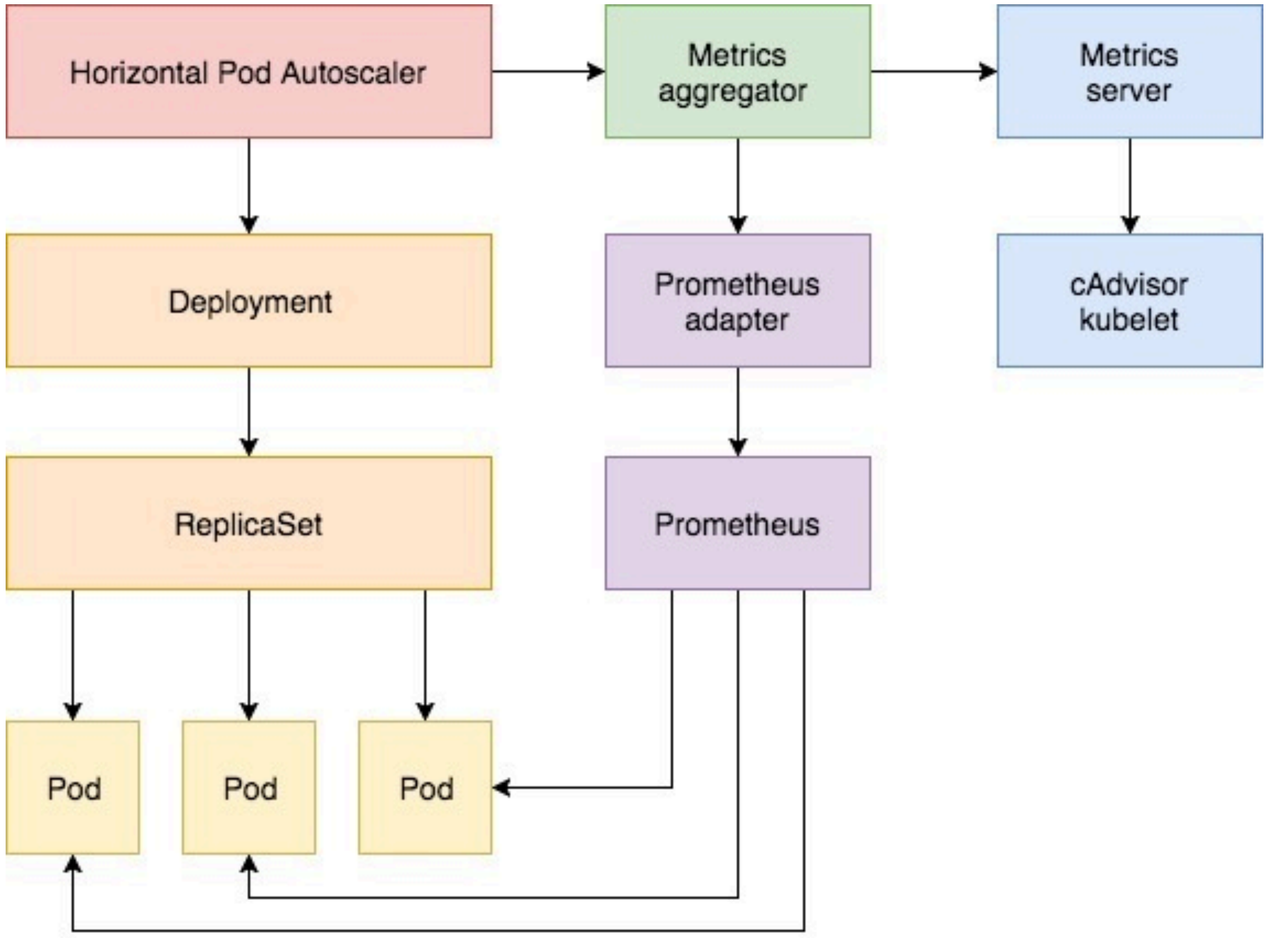
```
metricsClient := metrics.NewRESTMetricsClient(
    resourceclient.NewForConfigOrDie(clientConfig),
    custom_metrics.NewForConfigOrDie(clientConfig),
    external_metrics.NewForConfigOrDie(clientConfig),
)
```

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

HPA – 多维度Pod弹性伸缩指标演进

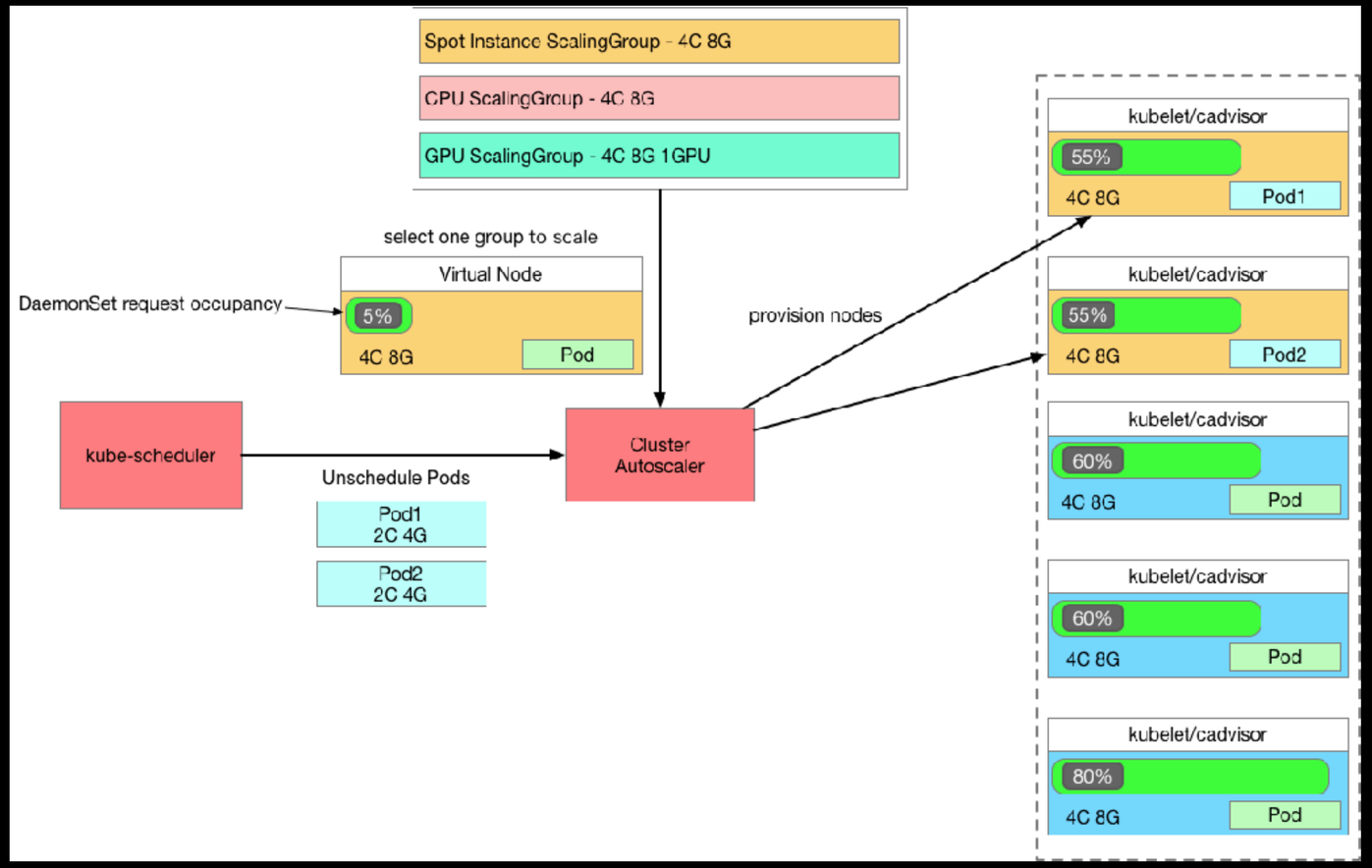
	API	注释
Resource	metrics.k8s.io	Pod资源类型的指标，计算的时候要除以Pod数目，再对比阈值进行判断。
Object/Pods	custom.metrics.k8s.io	Object：直接计算指标不进行平均比对阈值 Pods:　计算Pods数目进行平均比对阈值
External	external.metrics.k8s.io	External：通常由云厂商实现，对接云产品

HPA – ingress custom metrics




```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        kind: AverageUtilization
        averageUtilization: 50
  - type: Pods
    pods:
      metric:
        name: packets-per-second
      targetAverageValue: 1k
  - type: Object
    object:
      metric:
        name: requests-per-second
      describedObject:
        apiVersion: extensions/v1beta1
        kind: Ingress
        name: main-route
      target:
        kind: Value
        value: 10k
```

Cluster-Autoscaler - 集群节点伸缩的组件

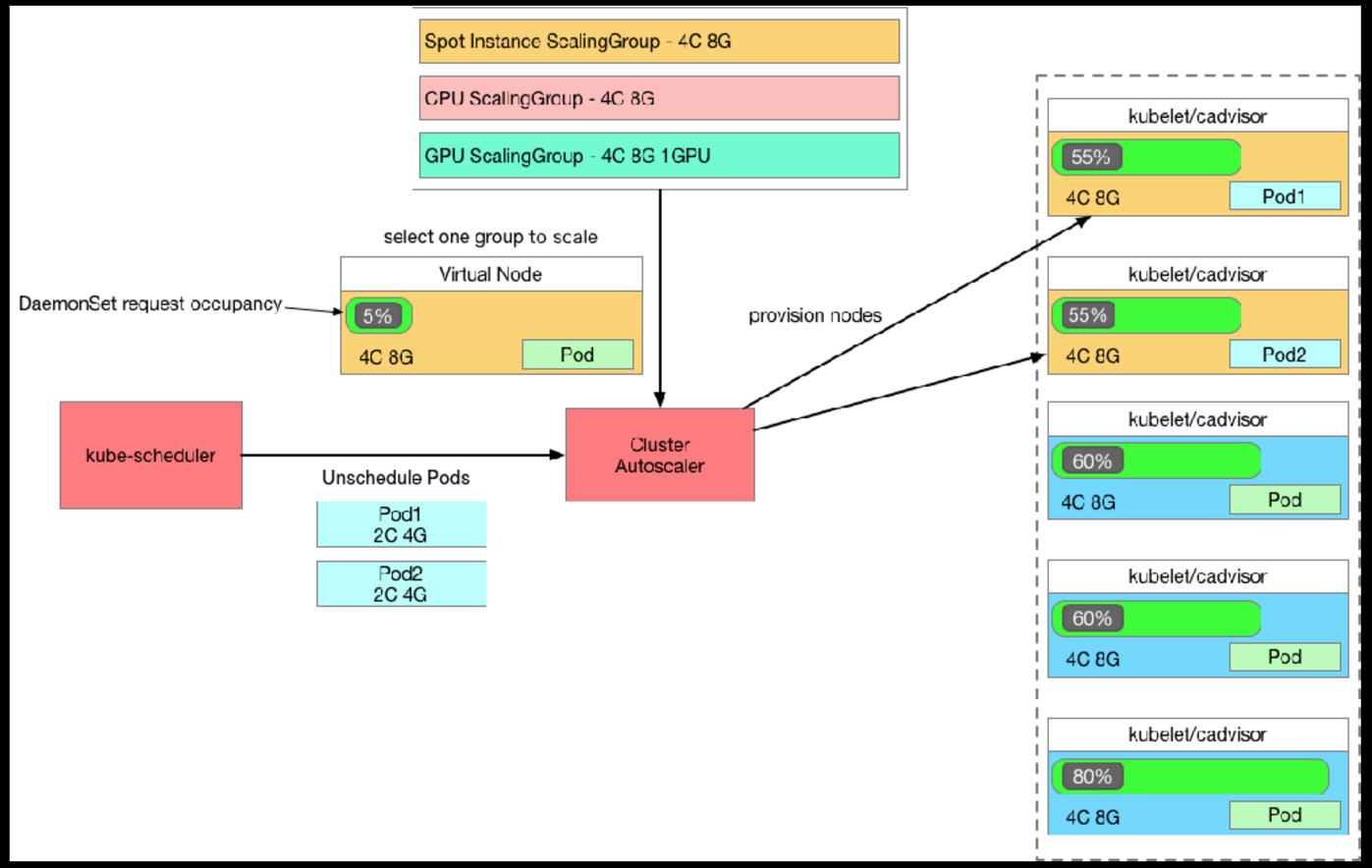


Cluster–Autoscaler扩容的条件是存在未调度的Pod
Cluster–Autoscaler缩容的条件是节点利用率低于阈值

Cluster–Autoscaler中核心的概念ASG –> ESS，增减节点即触发ASG增加节点的接口，移除节点即指定节点从ASG中移除。

Cluster–Autoscaler会监听所有的Pod，当Pod出现未调度的时候，会尝试将配置的ASG模拟成为一个虚拟节点，尝试将未调度的容器进行重新调度，并选择一个符合ASG进行节点伸缩。没有扩容任务的时候，会便利每个节点的request资源，检查资源的申请值，低于阈值的时候会逐个删除。

Cluster-Autoscaler - 扩容



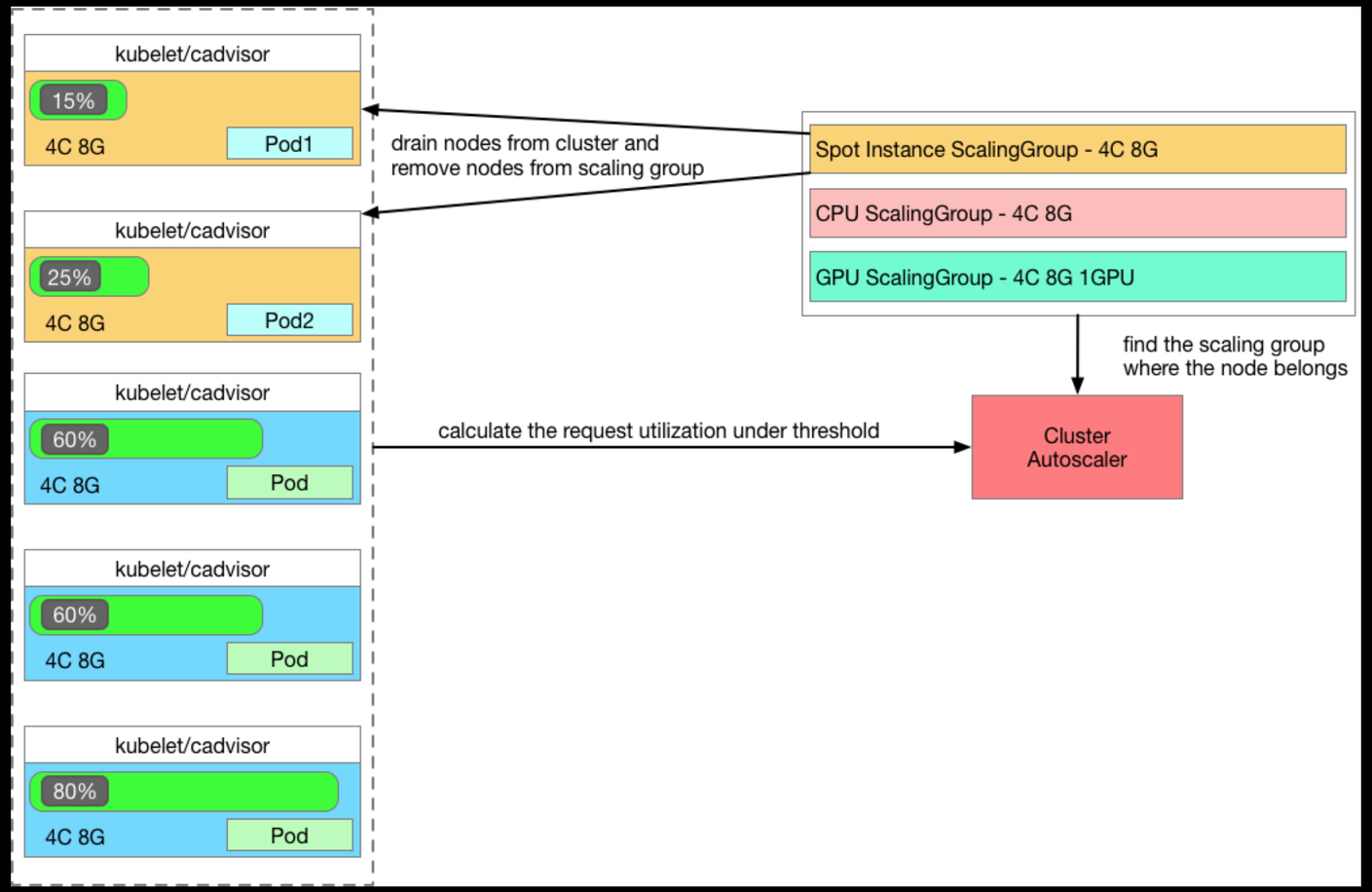
扩容判断的判断条件为有未调度的容器且不在冷却阈值内。

扩容时会进行模拟调度，同时会将DaemonSet与Static Pod进行模拟，因此可调度资源会略低于机器配置。

模拟调度使用同版本的kube-shecudler，因此最好与kubernetes的版本保持一致。

fleet控制器可能是方向，但目前对autoscaler来讲还不可行。

Cluster-Autoscaler - 缩容



缩容是Cluster-Autoscaler最复杂的逻辑

如下场景节点不可缩容：
- 有不可缩容label的
- 有cretical Pod的
- 节点上有statefulset的
- PDB到达最小副本的
- 未被autoscaler的伸缩组管理的

缩容目前的只能一个节点一个节点缩容，缩容后的重新调度会带来利用率的重新计算。

缩容的具体操作为当利用率低于阈值到达指定时间，进行打label禁止调度，驱逐容器，删除节点。

Cluster-Autoscaler - 使用场景

	特点	例子
在线任务	弹出时延敏感	流量负载型应用
离线任务	缩容时延敏感，价格敏感	AI/大数据类型任务
定时任务	稳定性敏感	定时弹出，定时回收
特殊任务	稳定性敏感	基于网络带宽与资源利用率

Cluster-Autoscaler - 实现资源预留的占位“气球🎈”

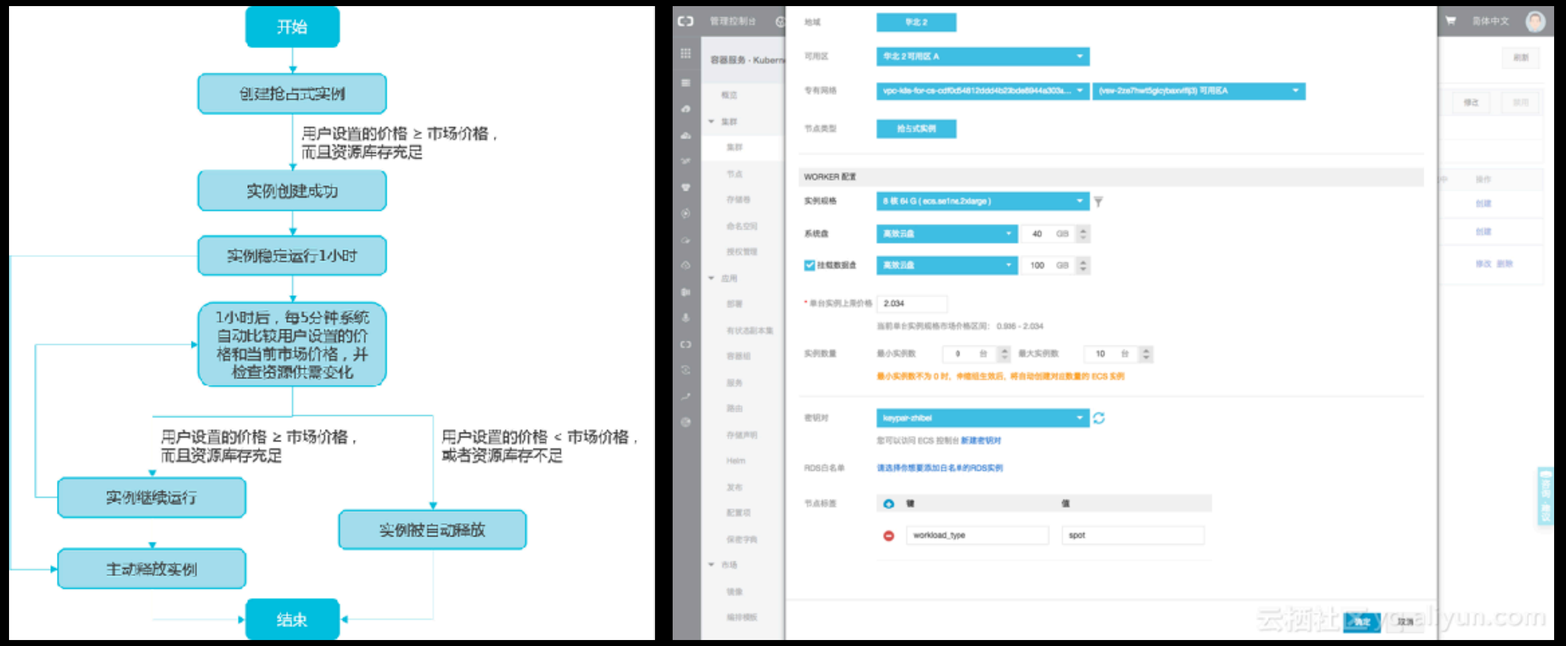
```yaml
apiVersion: scheduling.k8s.io/v1beta
kind: PriorityClass
metadata:
  name: overprovisioning
value: -1
globalDefault: false
description: "Priority class used by overprovisioning."
```

设置Priority非常低的Pod，默认申请占位资源例如4C8G * 4，作为占位“气球🎈”，当系统的常规资源无法调度的时候，会尝试扎破气球，牺牲Priority很低的容器，此时Priority很低的容器进入未调度的状态，Cluster-Autoscaler进行扩容处理，提供扩容缓冲。

更进一步，根据集群size，动态增减占位“气球🎈”?

```yaml
containers:
  - image: gcr.io/google_containers/cluster-proportional-autoscaler-amd64:1.1.2
    name: autoscaler
    command:
      - /cluster-proportional-autoscaler
      - --namespace=default
      - --configmap=nginx-autoscaler
      - --target=deployment/nginx-autoscale-example
      - --default-params={"linear":{"coresPerReplica":2,"nodesPerReplica":1,"preventSinglePointFailure":true}}
      - --logtostderr=true
      - --v=2
```

Cluster-Autoscaler - 离线任务的福音Spot Instance



合理的使用阿里云ECS竞价实例，最高可以降低50% – 90% 的运营成本(相比按量付费的实例)，可以用相同的预算，将计算容量提升 2 – 10 倍。为了保证尽可能高概率的弹出竞价实例，可以设置多个AZ多个规格进行竞价，大大提升了竞价实例的创建成功率。

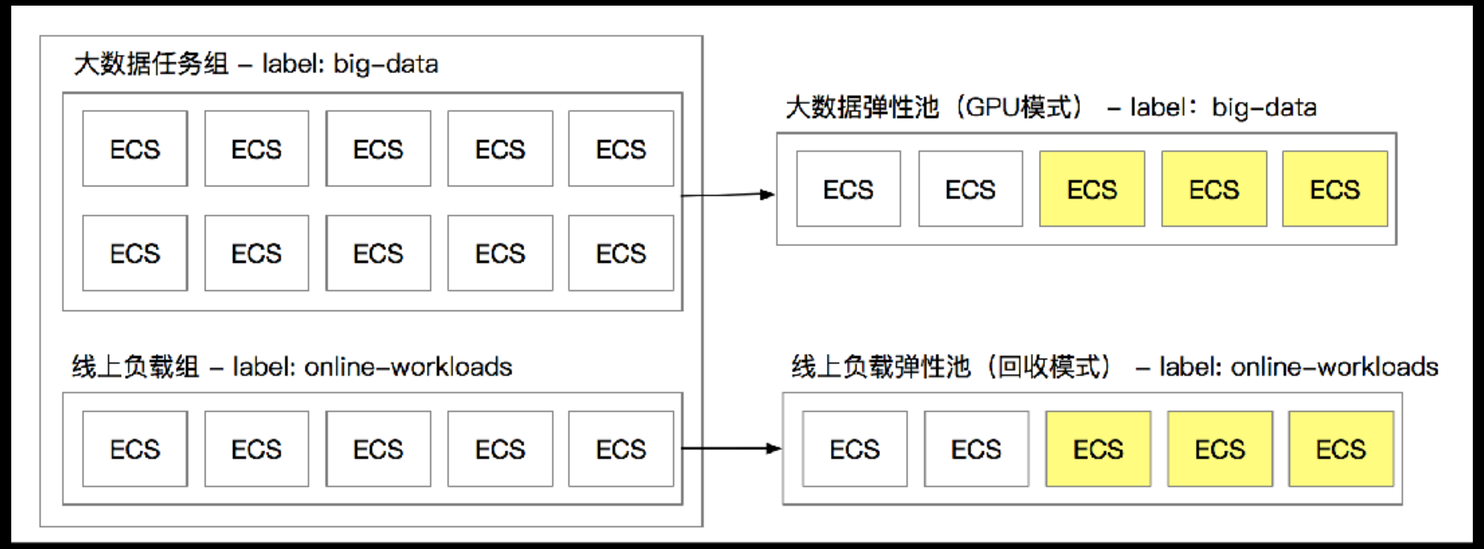
Cluster-Autoscaler - 多可用区、多实例规格保证伸缩效果

Cluster-Autoscaler感知机器规格的方式是通过NodeTemplate的模拟调度实现的，当配置多个不同实例族的时候，会默认选择第一个机器配置作为规格。具体的规格选择交由ESS进行判断。目前支持如下四种策略。

可用区均衡
价格最低
Spot优先
顺序优先

多实例规格、多可用区可以保证弹性的稳定性，保证资源分配的平衡性，保证价格的相对最优性

Cluster-Autoscaler - 指定分组调度与策略调度

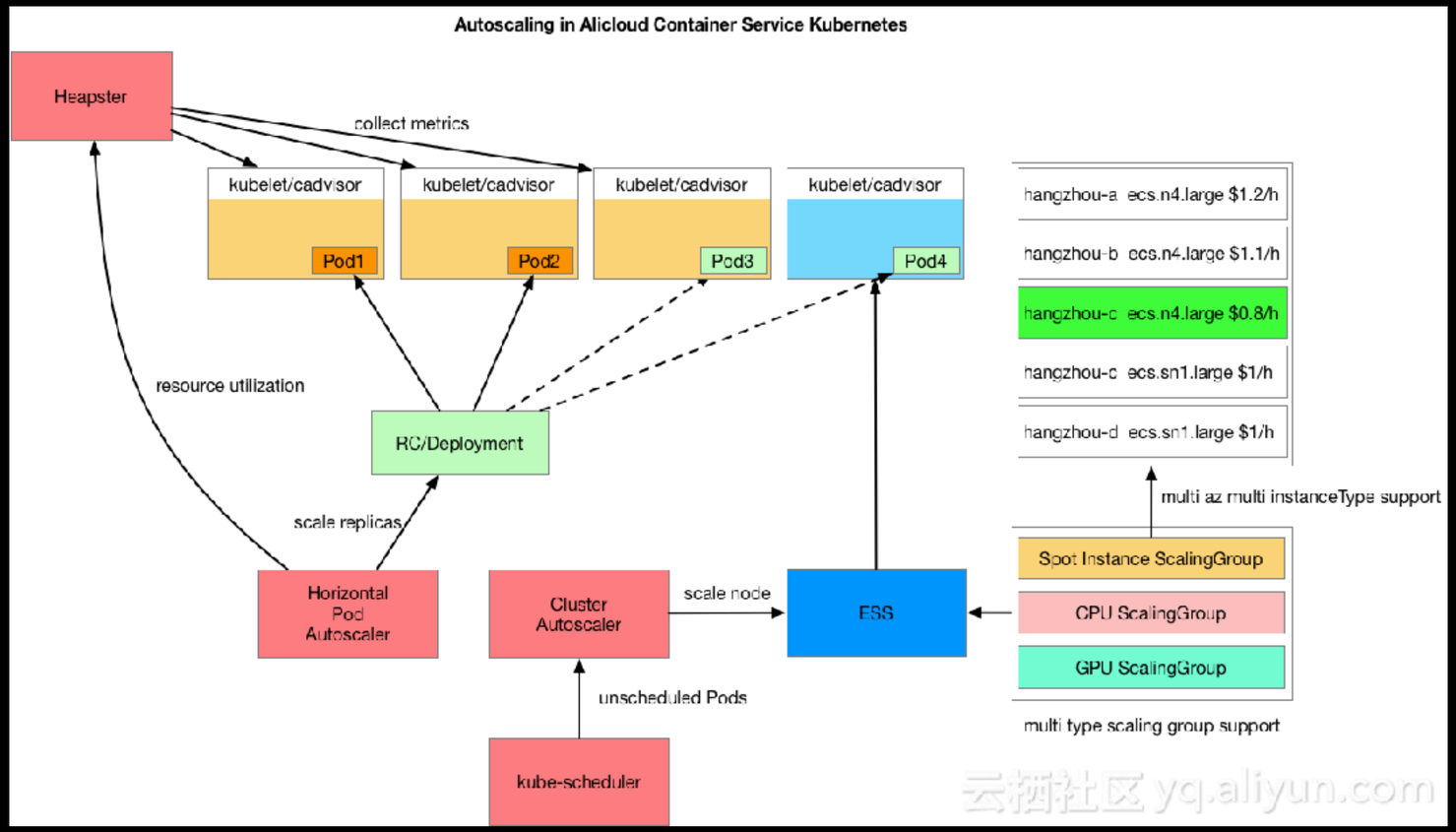


模板可以通过label-selector指定调度，每个弹性伸缩组可以有自己的伸缩特性。

大数据与机器学习可以考虑用GPU的Spot Instance伸缩组。

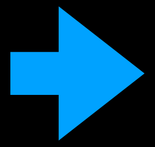
Cluster-Autoscaler - 实现非调度类型的弹性扩缩容

部分客户存在特殊的弹性伸缩要求，例如要基于metrics指标伸缩，要基于定时任务伸缩等非调度伸缩



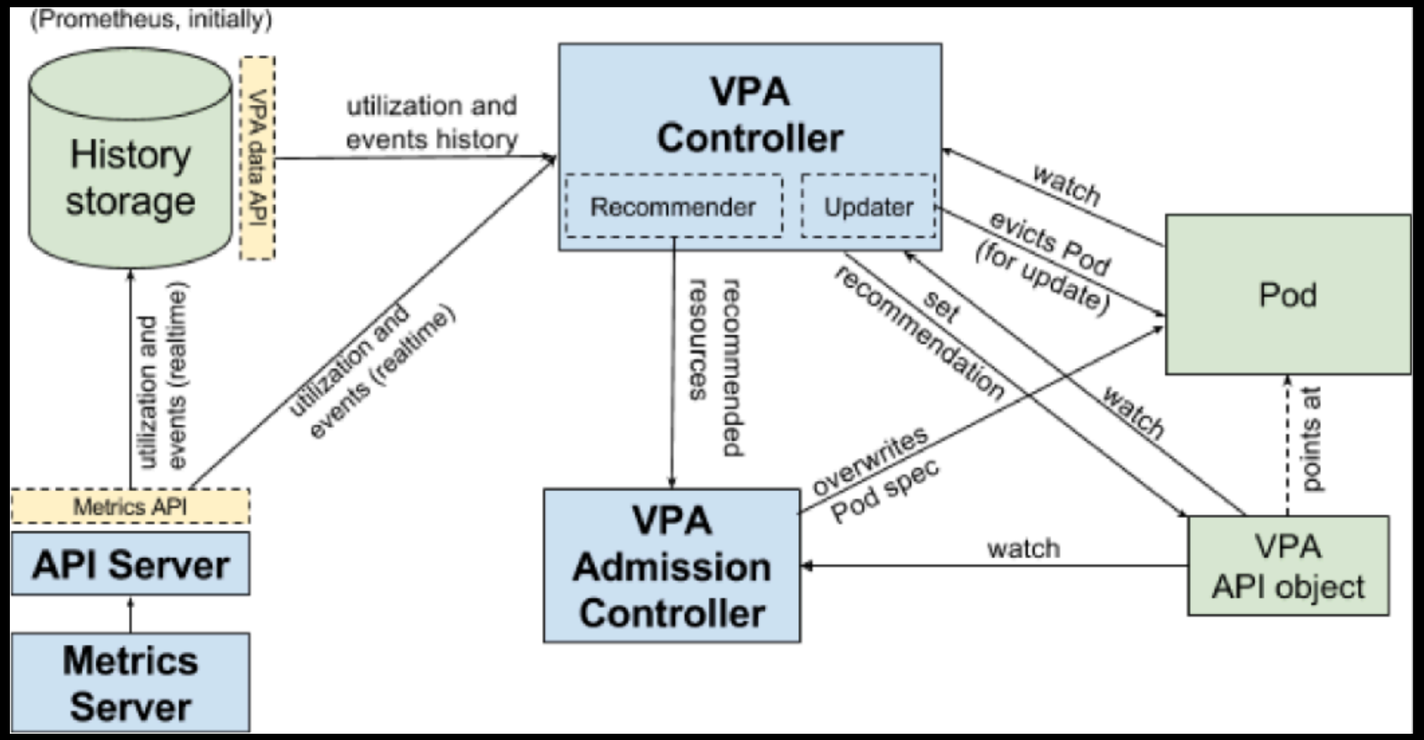
VPA - 要解决的问题

有状态服务进行扩缩容的问题
CPU竞争的问题
内存OOM的问题
资源利用率压缩节约成本的问题



Request的变化，重新调度或更新

VPA - 原理架构图



MetricsServer
实时监控数据提供

Prometheus
历史监控数据提供

Admission Controller
拦截VPA object与Pod Spec并动态更新Pod的request与limit。

Recommender
监视当前和过去的资源消耗，并提供推荐值容器的CPU和内存请求。

Updater
监控Pod Spec变化，触发驱逐进行变更

VPA - 四种不同的updateMode

```yaml
apiVersion: poc.autoscaling.k8s.io/v1alpha1
kind: VerticalPodAutoscaler
metadata:
  name: redis-vpa
spec:
  selector:
    matchLabels:
      app: redis
  updatePolicy:
      updateMode: "Auto"
---
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: redis-master
spec:
  selector:
    matchLabels:
      app: redis
  replicas: 3
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: master
        image: k8s.gcr.io/redis:e2e  # or just image: redis
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

四种不同的updateMode:

init:只在Pod创建的时候生效，后续不生效
auto/recreate:只在安全的时候或者不可不变化时更新，变更或者驱逐
off: 不真实触发变化，但可以看到VPA的建议

VPA - 获取推荐值的方式

安全边界：Pod正常运行峰值+15%的边界阈值
最小单元：25毫核 250M内存
默认算子：下限0.5 当前值0.9 上限0.95

1.初始250Mi内存，下限：138Mi 上限：263Mi
2.实时计算上下限，当触发上下限驱逐的时候会主动驱逐并设置新的request
3.如果连续触发，则会不断添加新的request到Pod，可能会触发资源无法满足，此时需要autoscaler协助。

上限驱逐

```
// No history      : *INF（do not force pod eviction）
// 12h history     : *3    (force pod eviction if the request is > 3 * upper bound)
// 24h history     : *2
// 1 week history : *1.14
```

下限驱逐

```
// No history    : *0   (do not force pod eviction)
// 5m history    : *0.6 (force pod eviction if the request is < 0.6 * lower bound)
// 30m history   : *0.9
// 60m history   : *0.95
```

cluster-proportional-autoscaler - 基于集群节点数目伸缩Pods

Demo

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-autoscaler
  namespace: default
  labels:
    app: autoscaler
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: autoscaler
    spec:
      containers:
        - image: gcr.io/google_containers/cluster-proportional-autoscaler-amd64:1.1.2
          name: autoscaler
          command:
            - /cluster-proportional-autoscaler
            - --namespace=default
            - --configmap=nginx-autoscaler
            - --target=deployment/nginx-autoscale-example
            - --default-params={"linear":{"coresPerReplica":2,"nodesPerReplica":1,"preventSinglePointFailure":true}}
            - --logtostderr=true
            - --v=2
```

梯度模型

```
data:
  ladder: |-
    {
      "coresToReplicas":
      [
        [ 1, 1 ],
        [ 64, 3 ],
        [ 512, 5 ],
        [ 1024, 7 ],
        [ 2048, 10 ],
        [ 4096, 15 ]
      ],
      "nodesToReplicas":
      [
        [ 1, 1 ],
        [ 2, 2 ]
      ]
    }
```

cpa是类似kube-dns-autoscaler的抽象，可以通过cpa配置实现其他组件的弹性伸缩。

cpa目前还没有基于CRD的机制来实现，目前必须一个组件一个autoscaler，可以有更多优化的空间。

线性模型

```
data:
  linear: |-
    {
      "coresPerReplica": 2,
      "nodesPerReplica": 1,
      "min": 1,
      "max": 100,
      "preventSinglePointFailure": true
    }
```

```
replicas = max( ceil( cores * 1/coresPerReplica ) , ceil( nodes * 1/nodesPerReplica ) )
replicas = min(replicas, max)
replicas = max(replicas, min)
```