# 70Kubernetes 系列（六十四）集群成本监控—Kubernetes Resource Report 和 Kubecost

监控集群的使用效率是非常重要的，特别是如果应用程序由不深入研究 requests 并设置膨胀值“保留”的开发人员部署。当然，需要储备——但简单地请求资源并不是一个好主意。

例如，您有一个具有 4 个 vCPU (4.000 milicpu) 和 16 GB RAM 的 WorkerNode，并且您创建了一个 Kubernetes Deployment，您在其中将 Pod 的 requests 设置为 2.5m 和 4 GB 内存。启动一个 Pod 后，它会请求超过一半的可用处理器时间，而要启动第二个 Pod，Kubernetes 会报告可用节点上资源不足，这将导致启动另一个 WorkerNode，当然，会影响集群的整体成本。

为了避免这种情况，有几个实用程序，例如 Kubernetes Resource Report(https://codeberg.org/hjacobs/kube-resource-report) 和 Kubecost(https://www.kubecost.com/)。

## Kube Resource Report

Kubernetes Resource Report 是最容易运行的，而且按功能分类：它简单地显示资源，按类型分组，并显示统计信息——请求了多少 CPU/MEM，以及实际使用了多少。

我喜欢它正是因为它的简单性——只需启动它，每两周查看一次集群中发生的情况，并在必要时询问开发人员“这个应用程序真的需要 100500 GB 的内存吗？”

有一个 Helm Chart，但很少更新，因此从清单安装更容易。

创建命名空间：

```
[root@~]kubectl create ns kube-resource-report
namespace/kube-resource-report created
```

下载 kube-resource-report 存储库：

```
[root@~]git clone https://codeberg.org/hjacobs/kube-resource-report
[root@~]cd kube-resource-report/
```

在 deploy 目录中已经有一个 Kustomize 文件，让我们将它安装添加到我们的命名空间中：

```
[root@~]echo "namespace: kube-resource-report" >> deploy/kustomization.yaml
```

检查：

```
# cat deploy/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- rbac.yaml
- service.yaml
- configmap.yaml
namespace: kube-resource-report
```

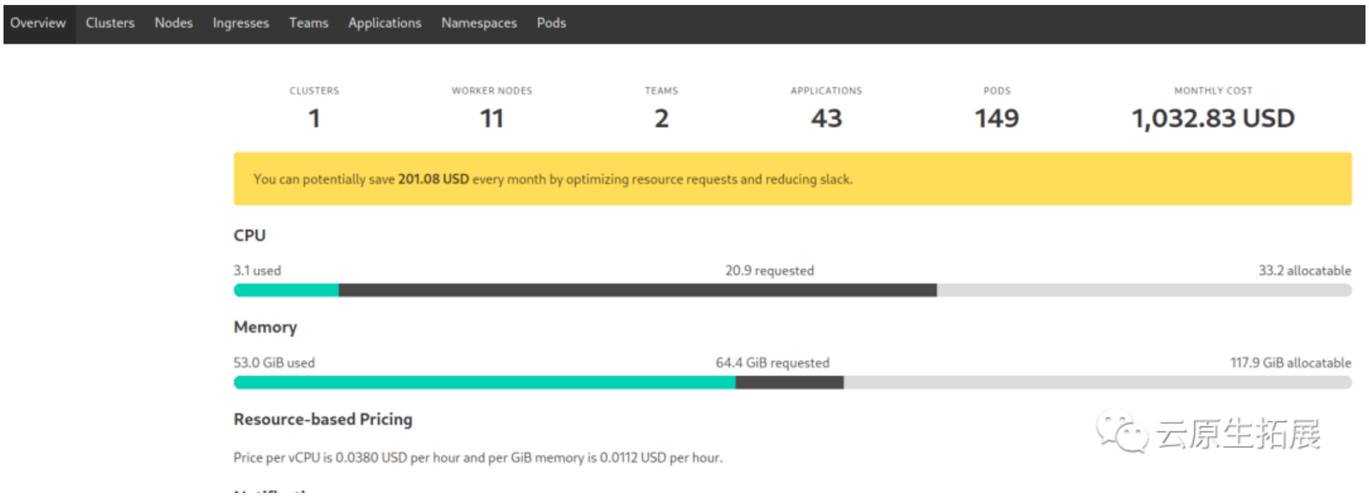
然后安装：

```
[root@~]kubectl apply -k deploy/
serviceaccount/kube-resource-report created
clusterrole.rbac.authorization.k8s.io/kube-resource-report created
clusterrolebinding.rbac.authorization.k8s.io/kube-resource-report created
configmap/kube-resource-report created
service/kube-resource-report created
deployment.apps/kube-resource-report created
```

开放访问其服务：

```
[root@~]kubectl -n kube-resource-report port-forward svc/kube-resource-report 8080:80
```

并在浏览器中打开报告：



| CLUSTERS | WORKER NODES | TEAMS | APPLICATIONS | PODS | MONTHLY COST |
|---|---|---|---|---|---|
| 1 | 11 | 2 | 43 | 149 | 1,032.83 USD |

You can potentially save **201.08 USD** every month by optimizing resource requests and reducing slack.

CPU

3.1 used 20.9 requested 33.2 allocatable

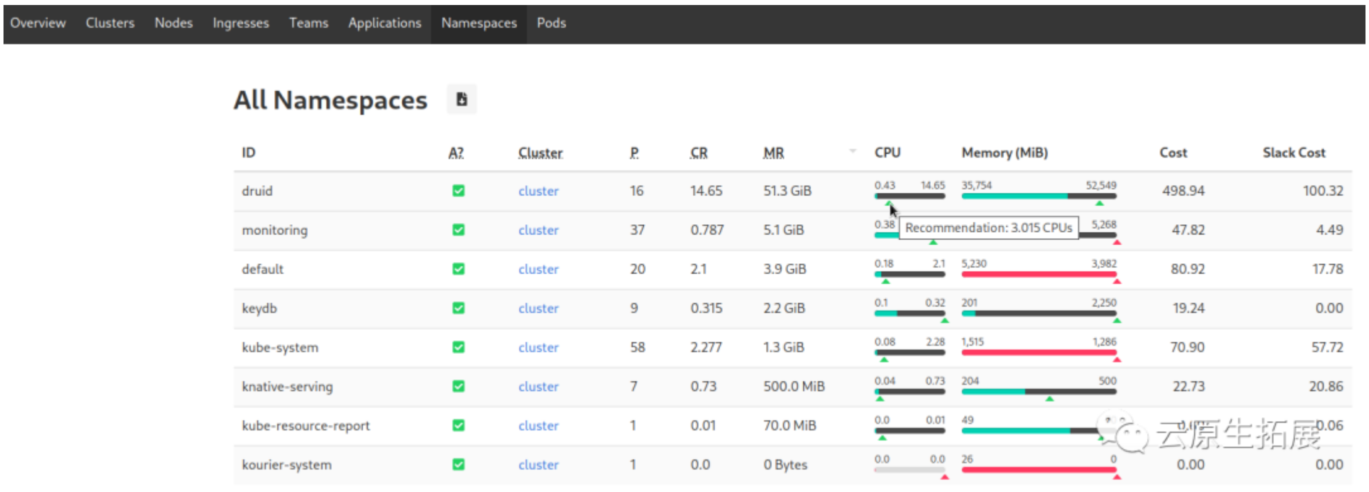
Memory

53.0 GiB used 64.4 GiB requested 117.9 GiB allocatable

Resource-based Pricing

Price per vCPU is 0.0380 USD per hour and per GiB memory is 0.0112 USD per hour.

接下来，例如，转到命名空间，并按 CR（请求的 CPU）对列进行排序：



All Namespaces

| ID | A? | Cluster | P | CR | MR | CPU | Memory (MiB) | Cost | Slack Cost |
|---|---|---|---|---|---|---|---|---|---|
| druid | ✓ | cluster | 16 | 14.65 | 51.3 GiB | 0.43 14.65 | 35,754 52,549 | 498.94 | 100.32 |
| monitoring | ✓ | cluster | 37 | 0.787 | 5.1 GiB | 0.38 Recommendation: 3.015 CPUs | 5,268 | 47.82 | 4.49 |
| default | ✓ | cluster | 20 | 2.1 | 3.9 GiB | 0.18 2.1 | 5,230 3,982 | 80.92 | 17.78 |
| keydb | ✓ | cluster | 9 | 0.315 | 2.2 GiB | 0.1 0.32 | 201 2,250 | 19.24 | 0.00 |
| kube-system | ✓ | cluster | 58 | 2.277 | 1.3 GiB | 0.08 2.28 | 1,515 1,286 | 70.90 | 57.72 |
| knative-serving | ✓ | cluster | 7 | 0.73 | 500.0 MiB | 0.04 0.73 | 204 500 | 22.73 | 20.86 |
| kube-resource-report | ✓ | cluster | 1 | 0.01 | 70.0 MiB | 0.0 0.01 | 49 [illegible] | [illegible] | 0.06 |
| kourier-system | ✓ | cluster | 1 | 0.0 | 0 Bytes | 0.0 0.0 | 26 0 | 0.00 | 0.00 |

当您将鼠标悬停在滑块上时，Kube 资源报告将从它的角度建议最佳值。

然后自己想一想，考虑到你的应用程序，请求的资源是否真的需要，或者可以减少。

在我的情况下，我们有带 16 个 pod 的 Apache Druid，每个 pod 都运行一个喜欢处理器和内存的 JVM，并且希望为每个 Java 执行线程分配 1 个处理器核心，所以好的 - 让它成为所请求处理器的 14.65 个。

## kubecost

Kubecost 是 Kube Resource Report 的强化版。它可以计算流量、发送警报、为 Prometheus 生成指标，拥有自己的 Grafana 仪表板，可以连接多个 Kubernetes 集群等等。

并非没有 bug，但总的来说，该工具令人愉悦且功能强大。

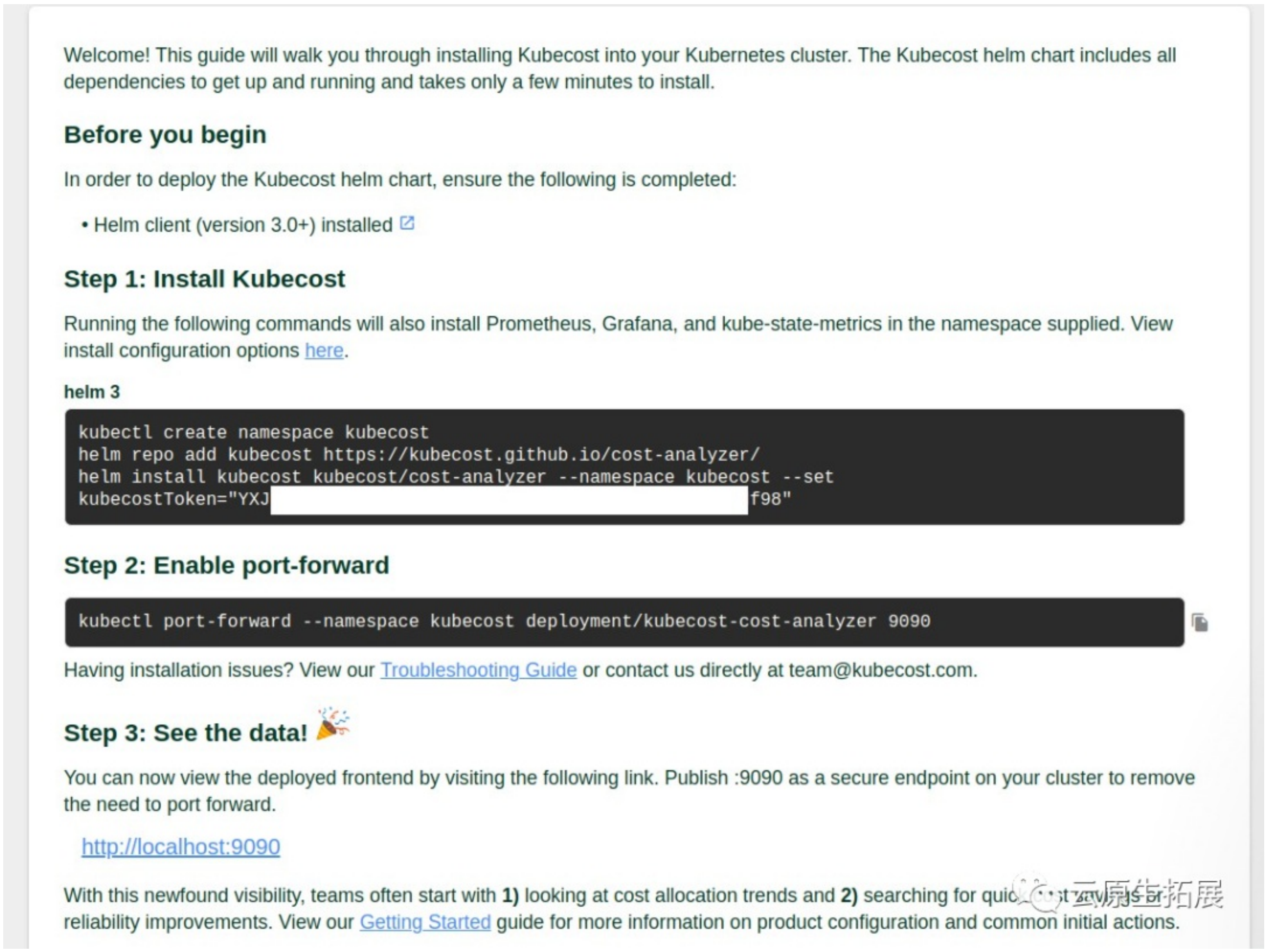
在我看来，商用的费用为 499 美元，价格有点高。但是，免费版足以满足基本需求。

"Under the hood" 使用自己的 Prometheus 实例存储数据。您可以禁用并使用外部实例，但不建议这样做（https://github.com/kubecost/docs/blob/main/custom-prom.md#bring-your-own-prometheus）。

## 安装

Github 文档 中描述了主要可用选项，另外您还可以获取其 Helm chart的默认值。

需要注册才能收到许可证密钥 - 转到 https://www.kubecost.com/install.html ，指定您的邮件，然后将立即重定向到包含您的密钥的安装说明：



Welcome! This guide will walk you through installing Kubecost into your Kubernetes cluster. The Kubecost helm chart includes all dependencies to get up and running and takes only a few minutes to install.

**Before you begin**

In order to deploy the Kubecost helm chart, ensure the following is completed:

- Helm client (version 3.0+) installed

**Step 1: Install Kubecost**

Running the following commands will also install Prometheus, Grafana, and kube-state-metrics in the namespace supplied. View install configuration options here.

**helm 3**

```
kubectl create namespace kubecost
helm repo add kubecost https://kubecost.github.io/cost-analyzer/
helm install kubecost kubecost/cost-analyzer --namespace kubecost --set
kubecostToken="YXJ[redacted]f98"
```

**Step 2: Enable port-forward**

```
kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 9090
```

Having installation issues? View our Troubleshooting Guide or contact us directly at team@kubecost.com.

**Step 3: See the data!**

You can now view the deployed frontend by visiting the following link. Publish :9090 as a secure endpoint on your cluster to remove the need to port forward.

http://localhost:9090

With this newfound visibility, teams often start with **1)** looking at cost allocation trends and **2)** searching for quick cost savings or reliability improvements. View our Getting Started guide for more information on product configuration and common initial actions.

### Helm chart values

首先，让我们创建自己的 values。

如果您已经部署了 Kube Prometheus Stack，并且拥有 Grafana 和 NodeExporter，那么在 Kubecost 中禁用它们是有意义的。另外关闭 kube-state-metrics ，这样监控中的数据就不会重复了。

为了让 Prometheus 开始从我们自己的堆栈中收集 Kubecost 指标，设置一个 ServiceMonitor 的创建并添加它 labels ——然后就可以生成我们自己的警报并使用 Grafana 仪表板。

但是如果禁用内置Grafana的启动， kubecost-cost-analyzer 就不会启动它的Pod，不知道是bug还是feature。但它有自己的仪表板，可能会有用，所以你可以不使用它。

此外，您可以打开 networkCosts ，但我仍然没有看到足够的流量成本——也许我没有正确处理它。

networkCosts 在资源方面可能相当贪婪，因此需要监控 CPU 的使用情况。

实际上，我最初的 values.yaml：

```
kubecostToken: "c2V***f98"

kubecostProductConfigs:
  clusterName: development-qa-data-services

prometheus:
  kube-state-metrics:
    disabled: true
  nodeExporter:
    enabled: false
  serviceAccounts:
    nodeExporter:
      create: false

serviceMonitor:
  enabled: true
  additionalLabels:
    release: prometheus

networkCosts:
  enabled: true
  podMonitor:
    enabled: true
  config:
    services:
      amazon-web-services: true
```

安装到 kubecost 命名空间：

```
[root@~]helm repo add kubecost https://kubecost.github.io/cost-analyzer/
[root@~]helm upgrade --install -n kubecost --create-namespace -f values.yaml kubecost kubecost/cost-analyzer
```

打开端口：

```
[root@~]kubectl -n kubecost port-forward svc/kubecost-cost-analyzer 9090:9090
```

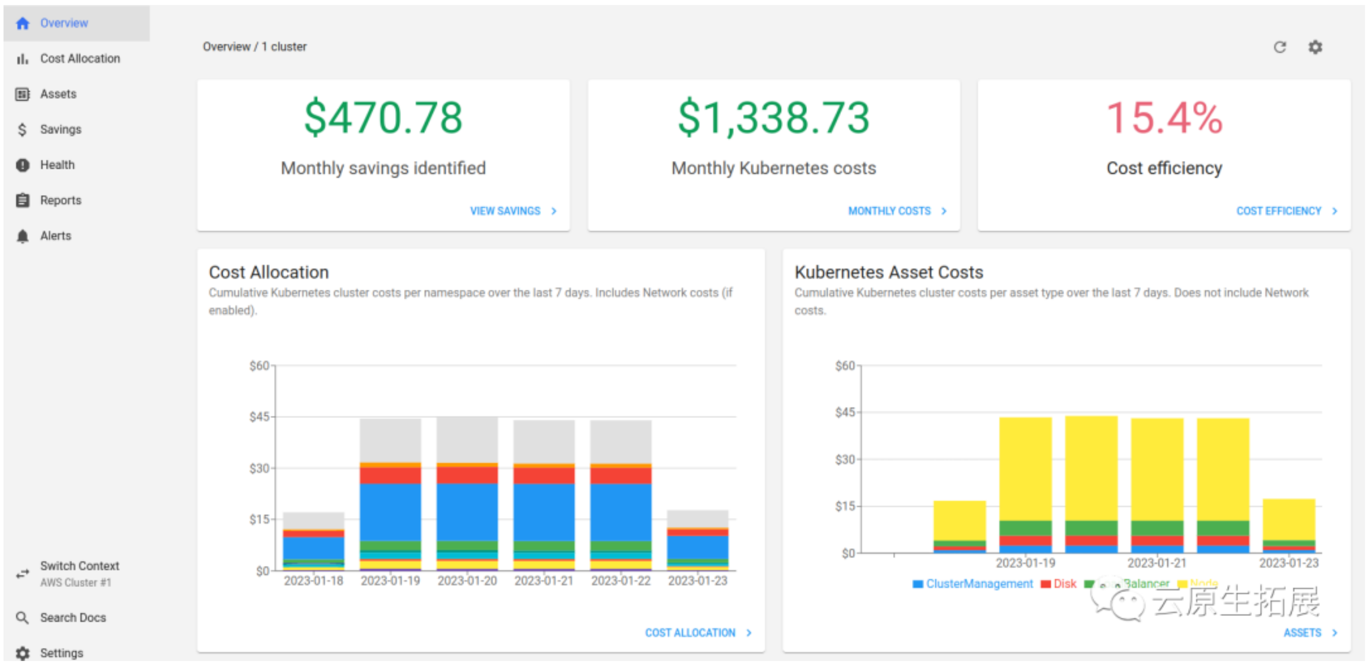
检查 Pod：

```
[root@~]kubectl -n kubecost get pod
NAME                                          READY   STATUS    RESTARTS   AGE
kubecost-cost-analyzer-5f5b85bf59-f22ld       2/2     Running   0          59s
kubecost-grafana-6bd995d6f9-kslh2             2/2     Running   0          63s
kubecost-network-costs-22dps                  1/1     Running   0          64s
kubecost-network-costs-m7rf5                  1/1     Running   0          64s
kubecost-network-costs-tcdvn                  1/1     Running   0          64s
kubecost-network-costs-xzvsz                  1/1     Running   0          64s
kubecost-prometheus-server-ddb597d5c-dvrgc    2/2     Running   0          6m49s
```

并打开对 kubecost-cost-analyzer 服务的访问：

```
[root@~]kubectl -n kubecost port-forward svc/kubecost-cost-analyzer 9090:9090
```

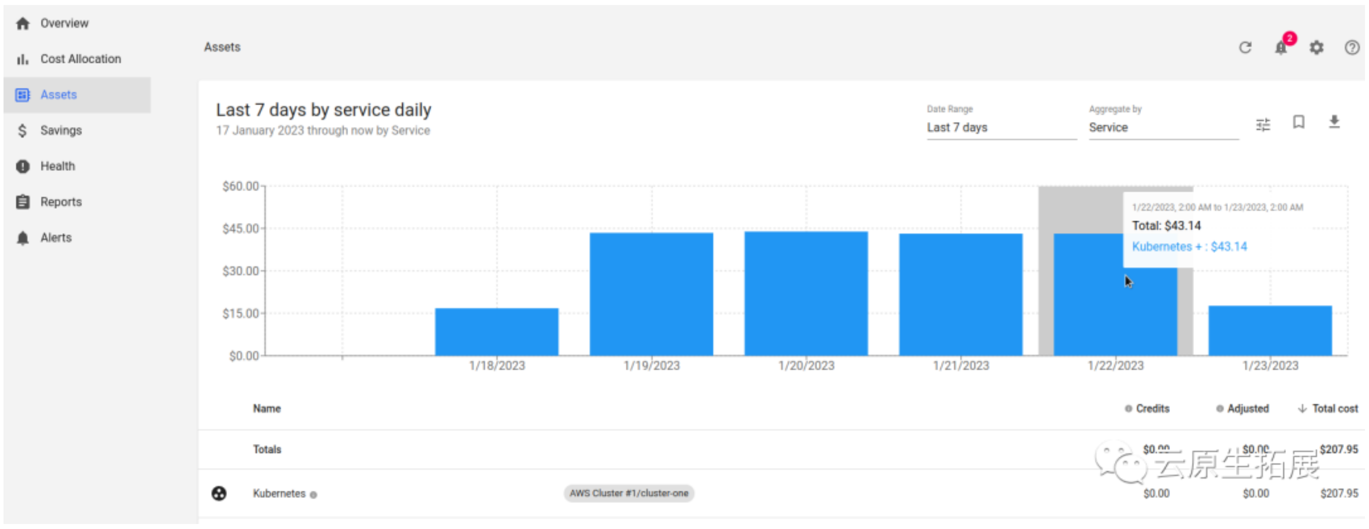
导航到 http://localhost:9090 。 这是 Kubecost 的截图，它已经在我们的一个集群上运行了将近一周：



让我们简要介绍一下主菜单项。

## Assets

要了解成本，最好从 Assets 项开始，它显示集群“硬件”的成本：

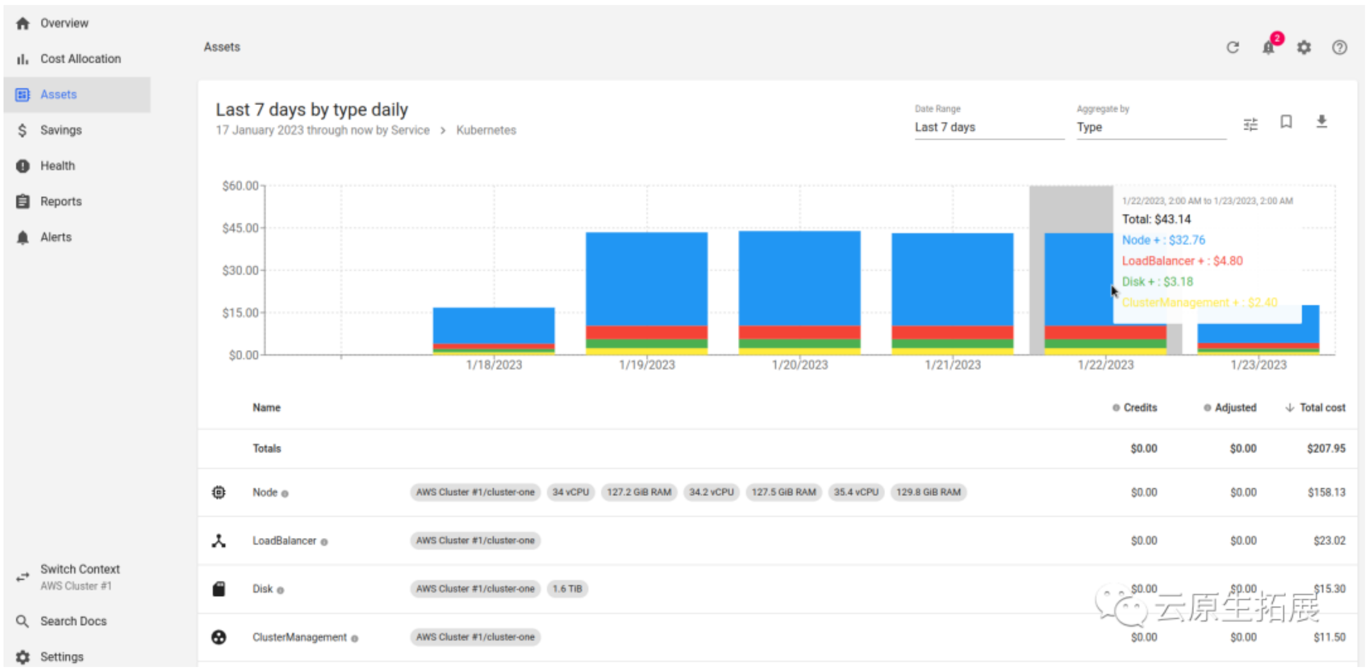

Overview
Cost Allocation
Assets
Savings
Health
Reports
Alerts
Assets
Last 7 days by service daily
17 January 2023 through now by Service
1/22/2023, 2:00 AM to 1/23/2023, 2:00 AM
Total: $43.14
Kubernetes + : $43.14

Date Range: Last 7 days
Aggregate by: Service

| Name | | Credits | Adjusted | Total cost |
|---|---|---|---|---|
| Totals | | $0.00 | $0.00 | $207.95 |
| Kubernetes | AWS Cluster #1/cluster-one | $0.00 | $0.00 | $207.95 |

我们看到我们的集群每天花费 43 美元。

您可以深入了解集群的详细信息，并查看按资源分类的细目——工作节点、负载均衡器、磁盘和 AWS Elastic Kubernetes 服务本身的成本：



Last 7 days by type daily

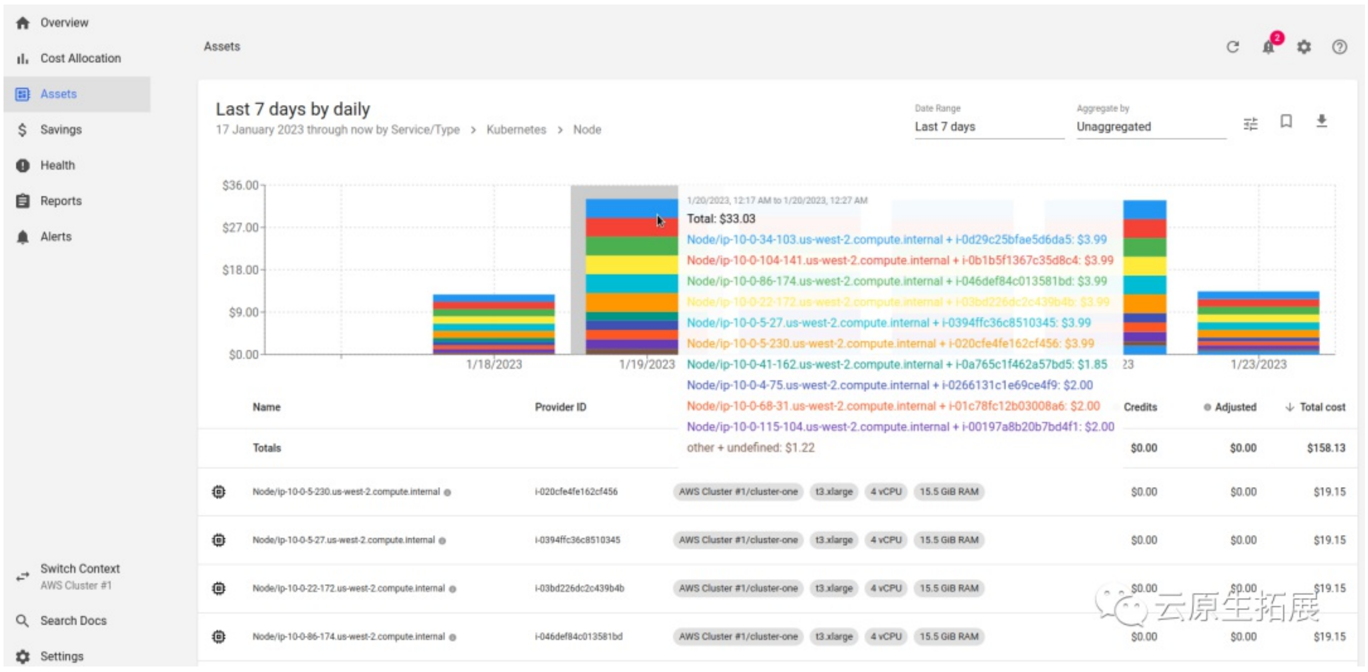
17 January 2023 through now by Service > Kubernetes

Date Range: Last 7 days
Aggregate by: Type

1/22/2023, 2:00 AM to 1/23/2023, 2:00 AM
Total: $43.14
Node + : $32.76
LoadBalancer + : $4.80
Disk + : $3.18
ClusterManagement + : $2.40

| | Name | | Credits | Adjusted | Total cost |
|---|---|---|---|---|---|
| | Totals | | $0.00 | $0.00 | $207.95 |
| | Node | AWS Cluster #1/cluster-one · 34 vCPU · 127.2 GiB RAM · 34.2 vCPU · 127.5 GiB RAM · 35.4 vCPU · 129.8 GiB RAM | $0.00 | $0.00 | $158.13 |
| | LoadBalancer | AWS Cluster #1/cluster-one | $0.00 | $0.00 | $23.02 |
| | Disk | AWS Cluster #1/cluster-one · 1.6 TiB | $0.00 | $0.00 | $15.30 |
| | ClusterManagement | AWS Cluster #1/cluster-one | $0.00 | $0.00 | $11.50 |

让我们更深入一点，在节点中：



Last 7 days by daily

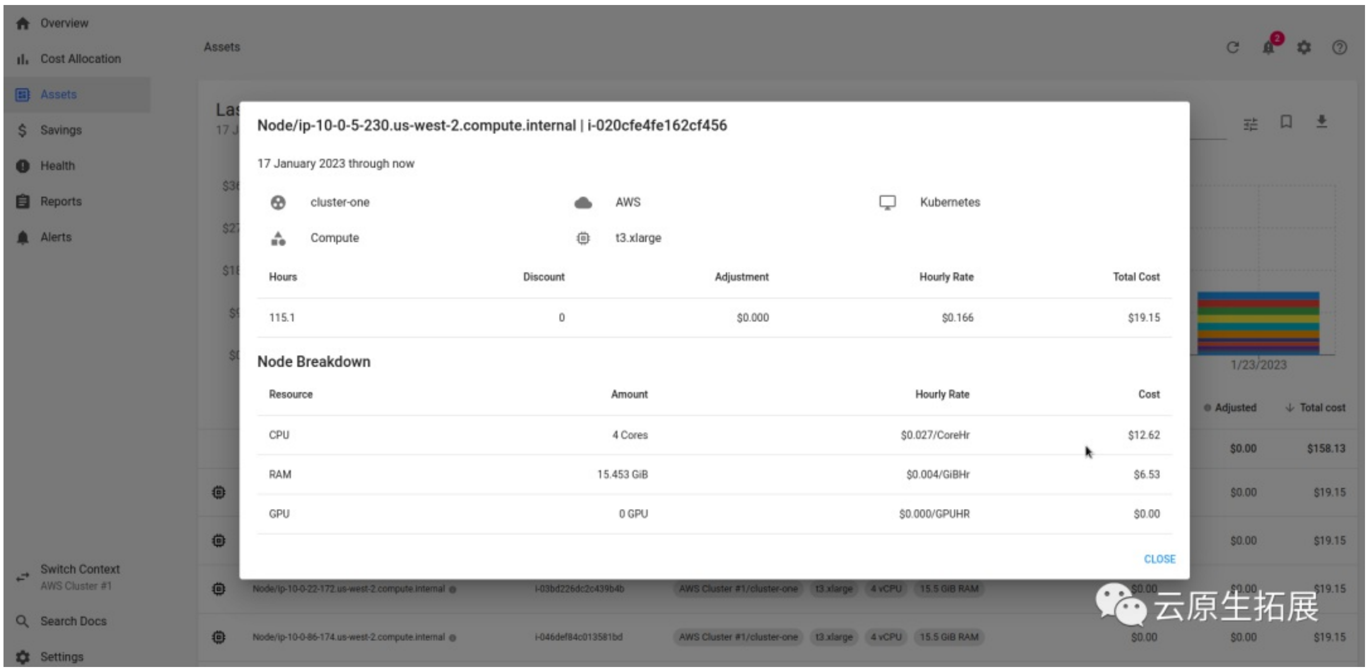
17 January 2023 through now by Service/Type > Kubernetes > Node

Date Range: Last 7 days
Aggregate by: Unaggregated

1/20/2023, 12:17 AM to 1/20/2023, 12:27 AM
Total: $33.03
Node/ip-10-0-34-103.us-west-2.compute.internal + i-0d29c25bfae5d6da5: $3.99
Node/ip-10-0-104-141.us-west-2.compute.internal + i-0b1b5f1367c35d8c4: $3.99
Node/ip-10-0-86-174.us-west-2.compute.internal + i-046def84c013581bd: $3.99
Node/ip-10-0-22-172.us-west-2.compute.internal + i-03bd226dc2c439b4b: $3.99
Node/ip-10-0-5-27.us-west-2.compute.internal + i-0394ffc36c8510345: $3.99
Node/ip-10-0-5-230.us-west-2.compute.internal + i-020cfe4fe162cf456: $3.99
Node/ip-10-0-41-162.us-west-2.compute.internal + i-0a765c1f462a57bd5: $1.85
Node/ip-10-0-4-75.us-west-2.compute.internal + i-0266131c1e69ce4f9: $2.00
Node/ip-10-0-68-31.us-west-2.compute.internal + i-01c78fc12b03008a6: $2.00
Node/ip-10-0-115-104.us-west-2.compute.internal + i-00197a8b20b7bd4f1: $2.00
other + undefined: $1.22

| | Name | Provider ID | | Credits | Adjusted | Total cost |
|---|---|---|---|---|---|---|
| | Totals | | | $0.00 | $0.00 | $158.13 |
| | Node/ip-10-0-5-230.us-west-2.compute.internal | i-020cfe4fe162cf456 | AWS Cluster #1/cluster-one · t3.xlarge · 4 vCPU · 15.5 GiB RAM | $0.00 | $0.00 | $19.15 |
| | Node/ip-10-0-5-27.us-west-2.compute.internal | i-0394ffc36c8510345 | AWS Cluster #1/cluster-one · t3.xlarge · 4 vCPU · 15.5 GiB RAM | $0.00 | $0.00 | $19.15 |
| | Node/ip-10-0-22-172.us-west-2.compute.internal | i-03bd226dc2c439b4b | AWS Cluster #1/cluster-one · t3.xlarge · 4 vCPU · 15.5 GiB RAM | $0.00 | $0.00 | $19.15 |
| | Node/ip-10-0-86-174.us-west-2.compute.internal | i-046def84c013581bd | AWS Cluster #1/cluster-one · t3.xlarge · 4 vCPU · 15.5 GiB RAM | $0.00 | $0.00 | $19.15 |

并可以查看特定节点的成本明细：



Node/ip-10-0-5-230.us-west-2.compute.internal | i-020cfe4fe162cf456

17 January 2023 through now

cluster-one AWS Kubernetes
Compute t3.xlarge

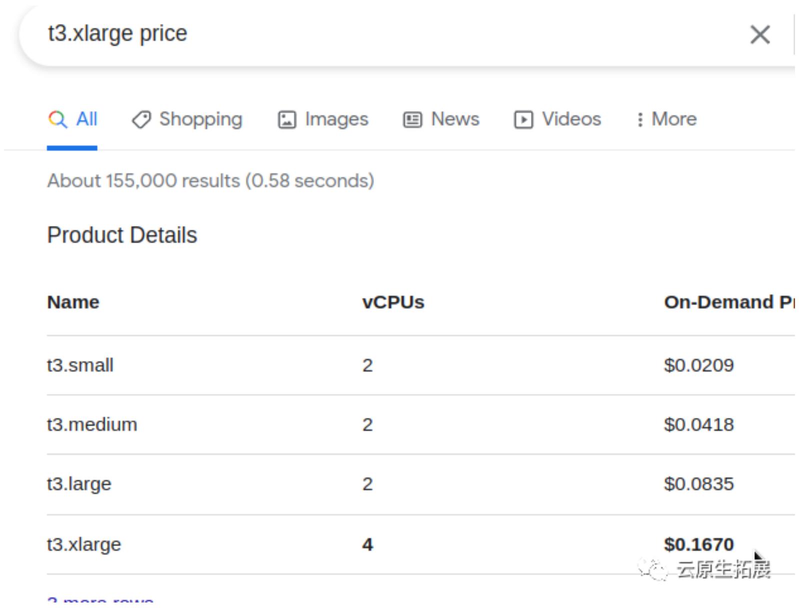
| Hours | Discount | Adjustment | Hourly Rate | Total Cost |
|---|---|---|---|---|
| 115.1 | 0 | $0.000 | $0.166 | $19.15 |

Node Breakdown

| Resource | Amount | Hourly Rate | Cost |
|---|---|---|---|
| CPU | 4 Cores | $0.027/CoreHr | $12.62 |
| RAM | 15.453 GiB | $0.004/GiBHr | $6.53 |
| GPU | 0 GPU | $0.000/GPUHR | $0.00 |

CLOSE

让我们检查：



About 155,000 results (0.58 seconds)

Product Details

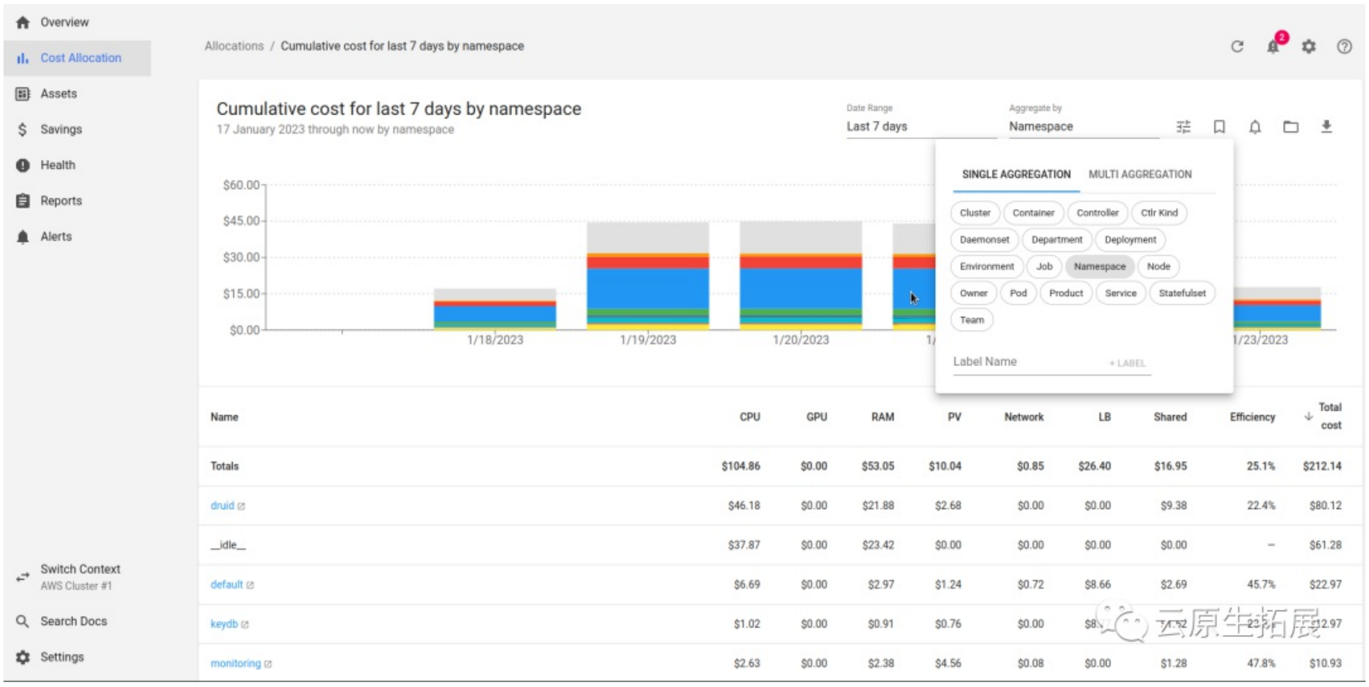
| Name | vCPUs | On-Demand Pr |
|---|---|---|
| t3.small | 2 | $0.0209 |
| t3.medium | 2 | $0.0418 |
| t3.large | 2 | $0.0835 |
| t3.xlarge | 4 | $0.1670 |

每小时 0.167，就像 Kubecost 在 Hourly Rate 中报告的那样。

要设置 AWS Spot 实例的成本，请参阅 Spot 实例数据源和 AWS Spot 实例。

## Cost Allocation

显示资源在 Kubernetes 自身的花费：



Cumulative cost for last 7 days by namespace

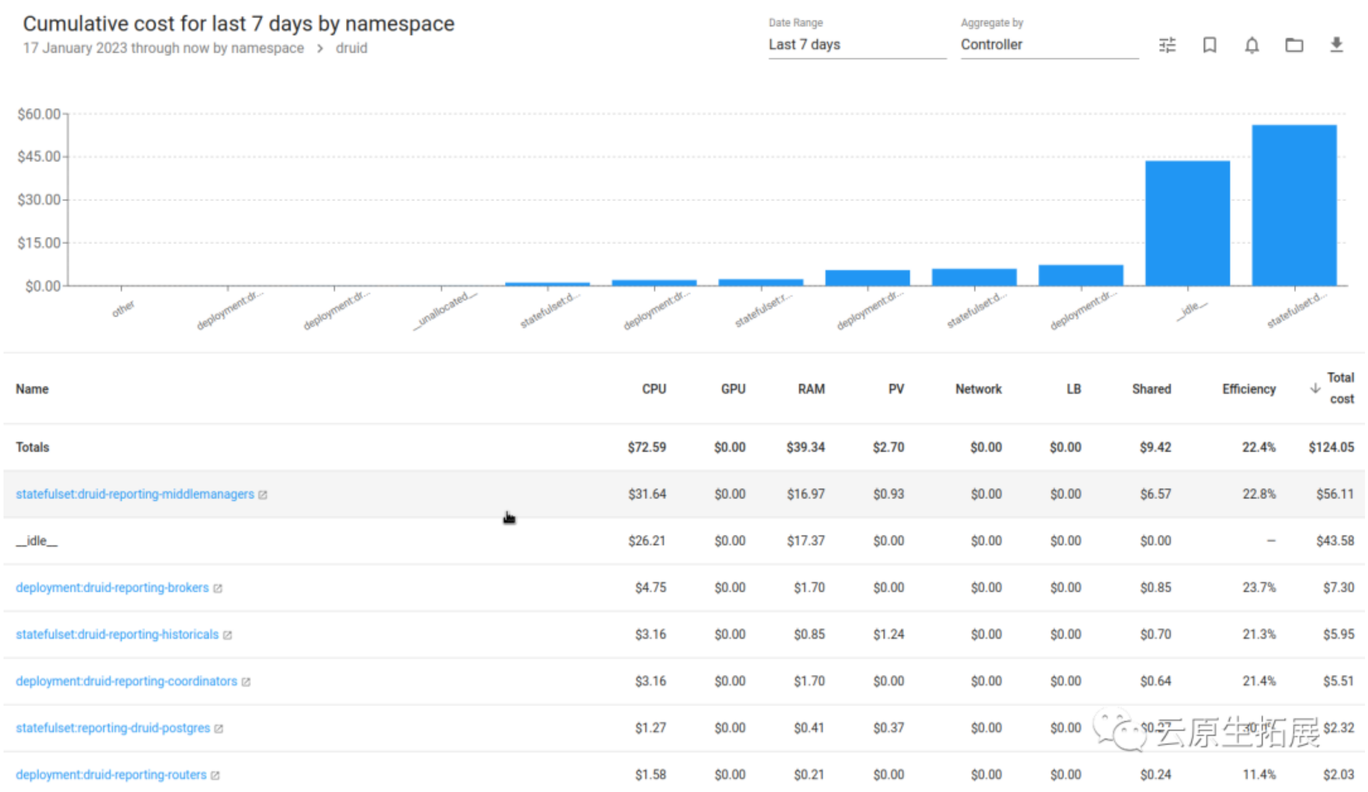
17 January 2023 through now by namespace

| Name | CPU | GPU | RAM | PV | Network | LB | Shared | Efficiency | Total cost |
|---|---|---|---|---|---|---|---|---|---|
| Totals | $104.86 | $0.00 | $53.05 | $10.04 | $0.85 | $26.40 | $16.95 | 25.1% | $212.14 |
| druid | $46.18 | $0.00 | $21.88 | $2.68 | $0.00 | $0.00 | $9.38 | 22.4% | $80.12 |
| __idle__ | $37.87 | $0.00 | $23.42 | $0.00 | $0.00 | $0.00 | $0.00 | – | $61.28 |
| default | $6.69 | $0.00 | $2.97 | $1.24 | $0.72 | $8.66 | $2.69 | 45.7% | $22.97 |
| keydb | $1.02 | $0.00 | $0.91 | $0.76 | $0.00 | [illegible] | [illegible] | [illegible] | [illegible] |
| monitoring | $2.63 | $0.00 | $2.38 | $4.56 | $0.08 | $0.00 | $1.28 | 47.8% | $10.93 |

Kubecost 考虑每个 WorkerNode 的 CPU 和 RAM 成本，并根据其请求和使用情况相应地显示每个命名空间的成本。

这是我们的 Apache Druid 拥有 CPU requests ，每周 48 美元或每天 4.08 美元。

再深入一点，按具体的控制器——StatefulSet、Deployment 进行细分：



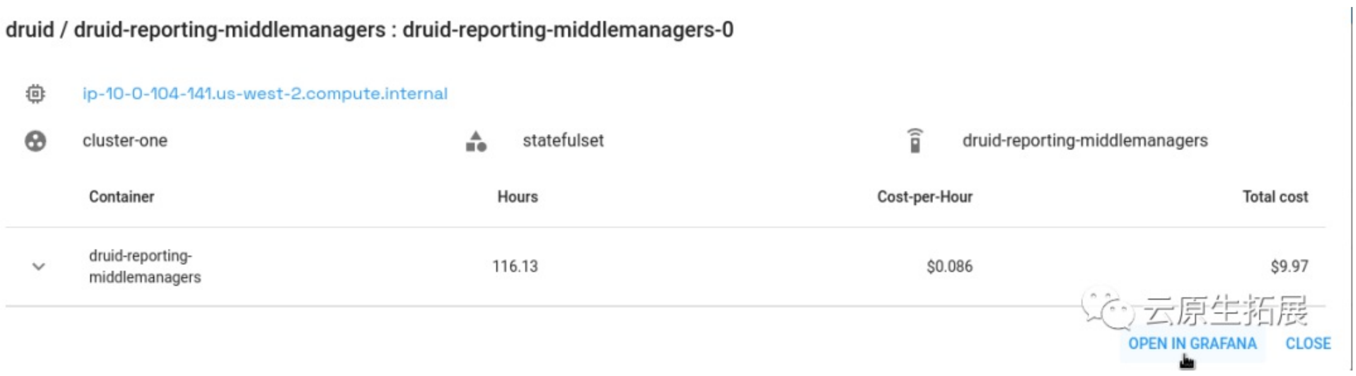
Cumulative cost for last 7 days by namespace

17 January 2023 through now by namespace > druid

| Name | CPU | GPU | RAM | PV | Network | LB | Shared | Efficiency | Total cost |
|---|---|---|---|---|---|---|---|---|---|
| Totals | $72.59 | $0.00 | $39.34 | $2.70 | $0.00 | $0.00 | $9.42 | 22.4% | $124.05 |
| statefulset:druid-reporting-middlemanagers | $31.64 | $0.00 | $16.97 | $0.93 | $0.00 | $0.00 | $6.57 | 22.8% | $56.11 |
| __idle__ | $26.21 | $0.00 | $17.37 | $0.00 | $0.00 | $0.00 | $0.00 | – | $43.58 |
| deployment:druid-reporting-brokers | $4.75 | $0.00 | $1.70 | $0.00 | $0.00 | $0.00 | $0.85 | 23.7% | $7.30 |
| statefulset:druid-reporting-historicals | $3.16 | $0.00 | $0.85 | $1.24 | $0.00 | $0.00 | $0.70 | 21.3% | $5.95 |
| deployment:druid-reporting-coordinators | $3.16 | $0.00 | $1.70 | $0.00 | $0.00 | $0.00 | $0.64 | 21.4% | $5.51 |
| statefulset:reporting-druid-postgres | $1.27 | $0.00 | $0.41 | $0.37 | $0.00 | $0.00 | [illegible] | [illegible] | $2.32 |
| deployment:druid-reporting-routers | $1.58 | $0.00 | $0.21 | $0.00 | $0.00 | $0.00 | $0.24 | 11.4% | $2.03 |

- CPU、RAM：使用资源的成本取决于WorkerNode资源的成本
- PV：所选控制器中的 PersistentVolume 成本，即对于 MiddleManagers 的 StaefulSet，我们有 PV，即 AWS EBS，这会花费我们金钱
- 网络：需要检查，值太低，对我来说
- LB：AWS 中按成本划分的负载均衡器
- Shared：不会单独统计的共享资源，比如 kube-system 命名空间，配置在 http://localhost:9090/settings > Shared Cost
- 效率：利用率与公式的请求：((CPU Usage / CPU Requested) * CPU Cost) + (RAM Usage / RAM Requested) * RAM Cost) / (RAM Cost + CPU Cost))

如果再深入一点，会有一个指向内置 Grafana 的链接，您可以在其中查看特定 Pod 的资源使用情况：

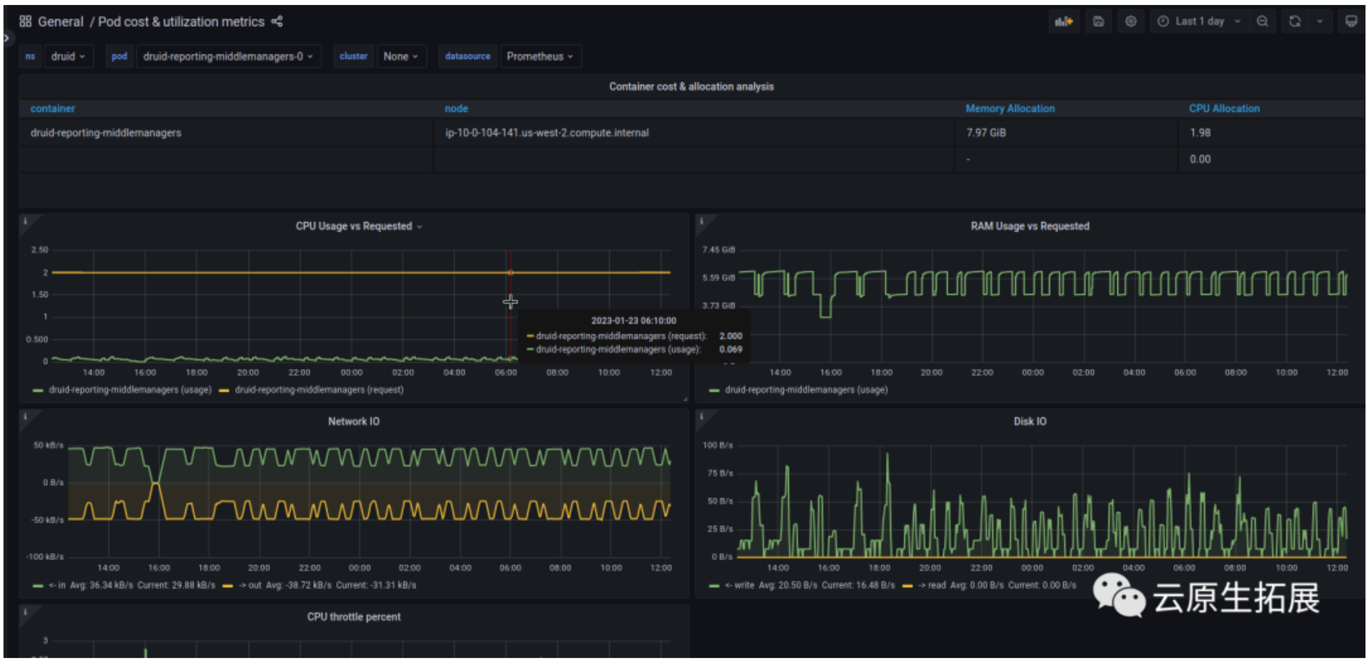


druid / druid-reporting-middlemanagers : druid-reporting-middlemanagers-0

ip-10-0-104-141.us-west-2.compute.internal

cluster-one | statefulset | druid-reporting-middlemanagers

| Container | Hours | Cost-per-Hour | Total cost |
|---|---|---|---|
| druid-reporting-middlemanagers | 116.13 | $0.086 | $9.97 |



General / Pod cost & utilization metrics

ns druid · pod druid-reporting-middlemanagers-0 · cluster None · datasource Prometheus

Container cost & allocation analysis

| container | node | Memory Allocation | CPU Allocation |
|---|---|---|---|
| druid-reporting-middlemanagers | ip-10-0-104-141.us-west-2.compute.internal | 7.97 GiB | 1.98 |
| | | - | 0.00 |

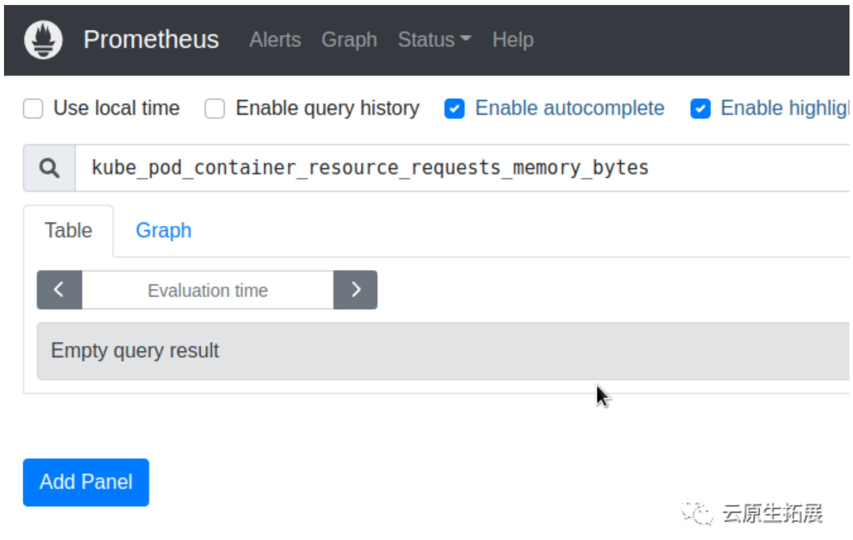
Last 1 day
CPU Usage vs Requested
2023-01-23 06:10:00
druid-reporting-middlemanagers (request): 2.000
druid-reporting-middlemanagers (usage): 0.069
druid-reporting-middlemanagers (usage)
druid-reporting-middlemanagers (request)
RAM Usage vs Requested
druid-reporting-middlemanagers (usage)
Network IO
<- in Avg: 36.34 kB/s Current: 29.88 kB/s
-> out Avg: -38.72 kB/s Current: -31.31 kB/s
Disk IO
<- write Avg: 20.50 B/s Current: 16.48 B/s
-> read Avg: 0.00 B/s Current: 0.00 B/s
CPU throttle percent

尽管 RAM Requested 的指标不是开箱即用的。

要检查指标，您可以转到内置的 Prometheus：

```
[root@~]kubectl -n kubecost port-forward svc/kubecost-prometheus-server 9091:80
```

事实上， kube_pod_container_resource_requests_memory_bytes 是空的：

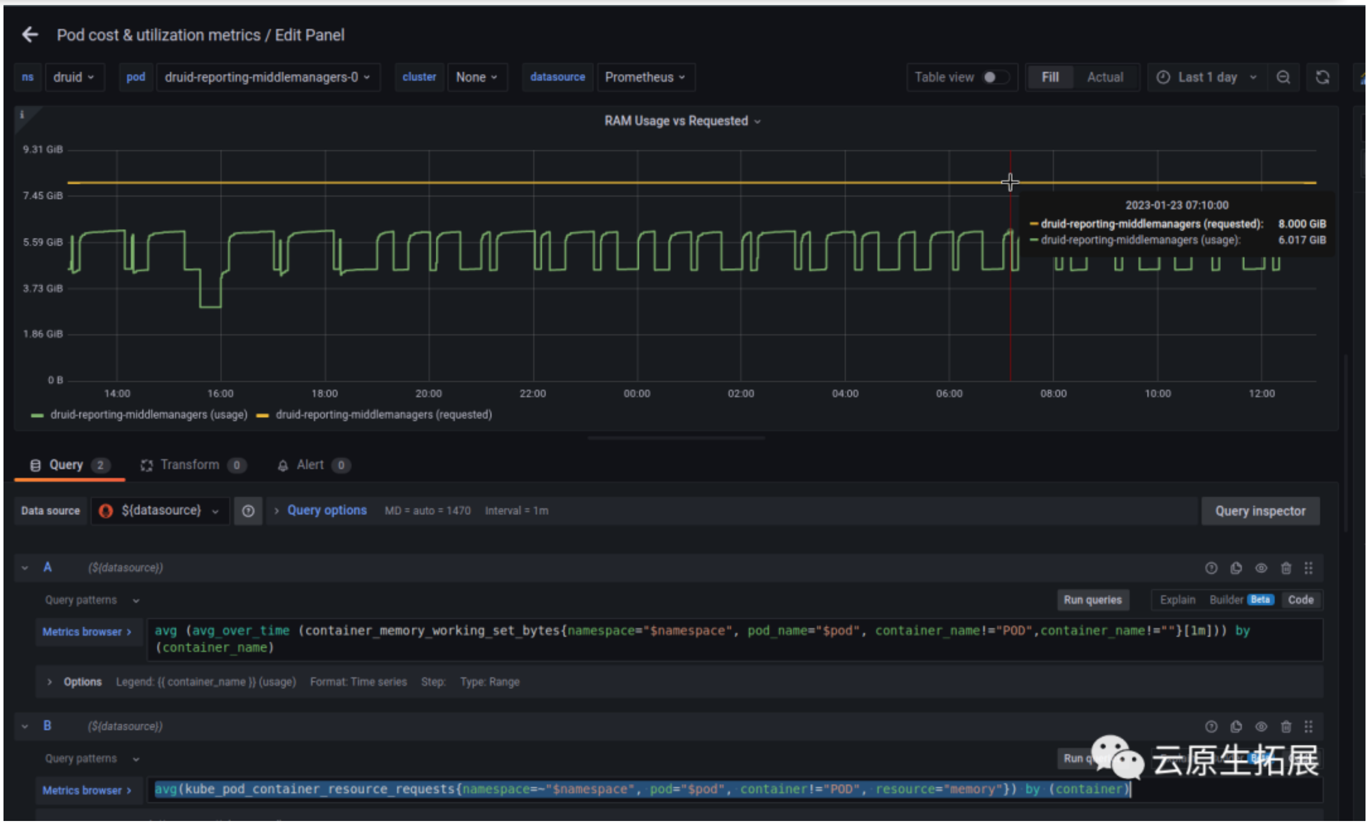


```
kube_pod_container_resource_requests_memory_bytes
```

Empty query result

因为指标现在被称为 kube_pod_container_resource_requests 和 resource="memory"，所以需要更新这个 Grafana 中的查询：

```
avg(kube_pod_container_resource_requests{namespace=~"$namespace", pod="$pod", container!="POD", resource="memory"}) by (container)
```


Pod cost & utilization metrics / Edit Panel

ns: druid · pod: druid-reporting-middlemanagers-0 · cluster: None · datasource: Prometheus · Last 1 day

RAM Usage vs Requested

2023-01-23 07:10:00
— druid-reporting-middlemanagers (requested): 8.000 GiB
— druid-reporting-middlemanagers (usage): 6.017 GiB

Data source: ${datasource} · Query options · MD = auto = 1470 · Interval = 1m

A (${datasource})

```
avg (avg_over_time (container_memory_working_set_bytes{namespace="$namespace", pod_name="$pod", container_name!="POD",container_name!=""}[1m])) by
(container_name)
```

Options · Legend: {{ container_name }} (usage) · Format: Time series · Step: · Type: Range

B (${datasource})

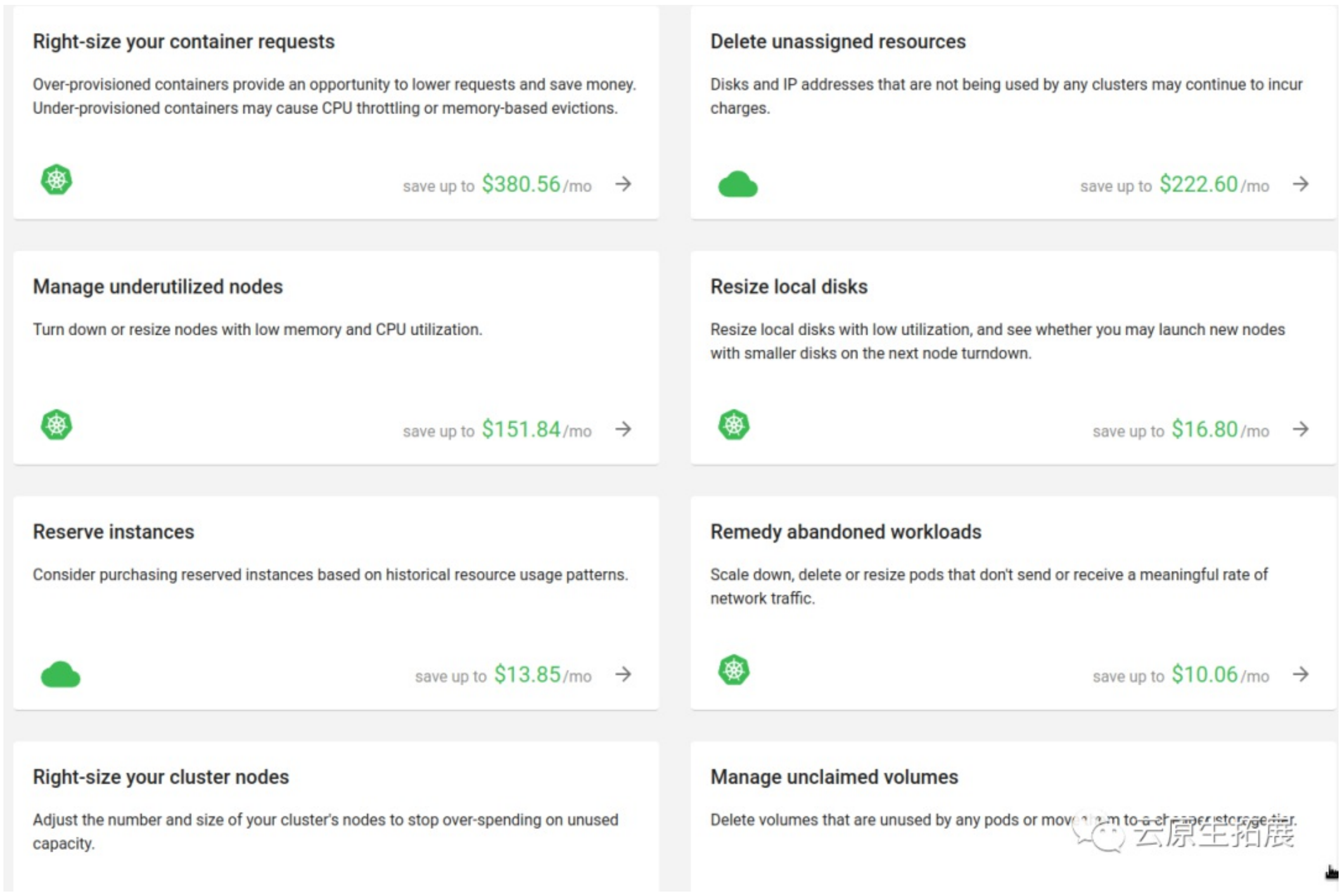
```
avg(kube_pod_container_resource_requests{namespace=~"$namespace", pod="$pod", container!="POD", resource="memory"}) by (container)
```

## idle

**idle** 费用是为现有对象（Pod、Deployments）分配的资源成本之间的差异——它们的请求和实际使用，以及它们工作的“空闲硬件”，即可用于推出新资源。

## Savings

以下是成本优化的一些技巧：



**Right-size your container requests**

Over-provisioned containers provide an opportunity to lower requests and save money. Under-provisioned containers may cause CPU throttling or memory-based evictions.

save up to $380.56/mo →

**Delete unassigned resources**

Disks and IP addresses that are not being used by any clusters may continue to incur charges.

save up to $222.60/mo →

**Manage underutilized nodes**

Turn down or resize nodes with low memory and CPU utilization.

save up to $151.84/mo →

**Resize local disks**

Resize local disks with low utilization, and see whether you may launch new nodes with smaller disks on the next node turndown.

save up to $16.80/mo →

**Reserve instances**

Consider purchasing reserved instances based on historical resource usage patterns.

save up to $13.85/mo →

**Remedy abandoned workloads**

Scale down, delete or resize pods that don't send or receive a meaningful rate of network traffic.

save up to $10.06/mo →

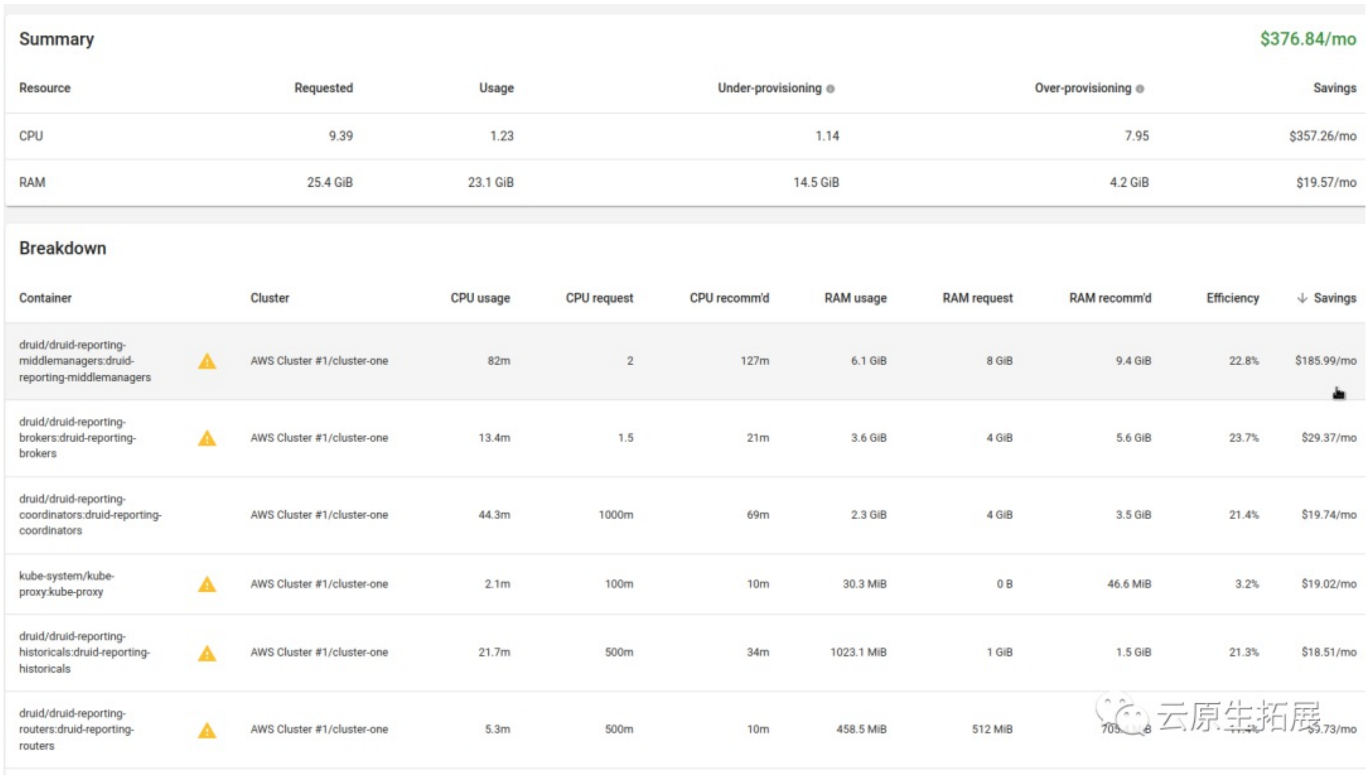
**Right-size your cluster nodes**

Adjust the number and size of your cluster's nodes to stop over-spending on unused capacity.

**Manage unclaimed volumes**

Delete volumes that are unused by any pods or move them to a cheaper storage tier.

例如，“Right-size your container requests”包含为资源配置 requests 的建议——类似于 Kubernetes 资源报告中的报告：



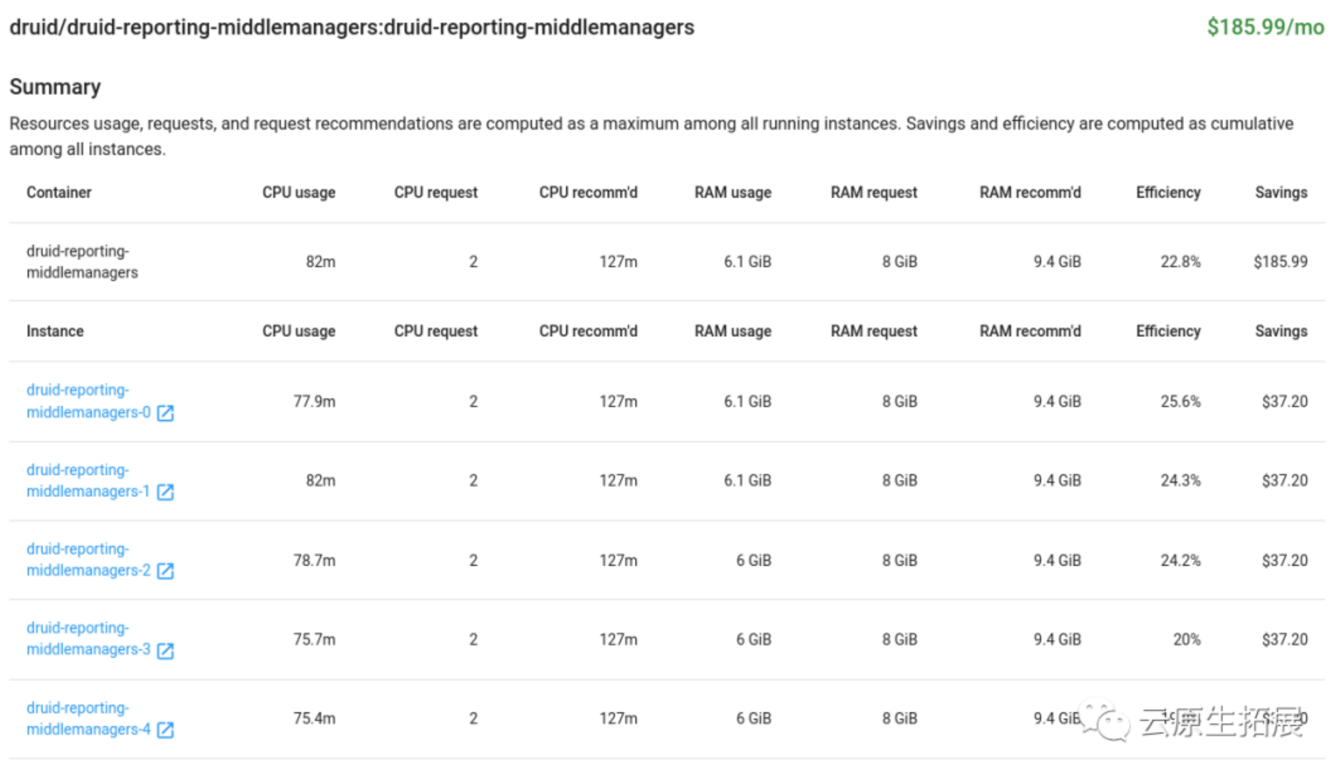
**Summary** $376.84/mo

| Resource | Requested | Usage | Under-provisioning | Over-provisioning | Savings |
|---|---|---|---|---|---|
| CPU | 9.39 | 1.23 | 1.14 | 7.95 | $357.26/mo |
| RAM | 25.4 GiB | 23.1 GiB | 14.5 GiB | 4.2 GiB | $19.57/mo |

**Breakdown**

| Container | | Cluster | CPU usage | CPU request | CPU recomm'd | RAM usage | RAM request | RAM recomm'd | Efficiency | ↓ Savings |
|---|---|---|---|---|---|---|---|---|---|---|
| druid/druid-reporting-middlemanagers:druid-reporting-middlemanagers | ⚠ | AWS Cluster #1/cluster-one | 82m | 2 | 127m | 6.1 GiB | 8 GiB | 9.4 GiB | 22.8% | $185.99/mo |
| druid/druid-reporting-brokers:druid-reporting-brokers | ⚠ | AWS Cluster #1/cluster-one | 13.4m | 1.5 | 21m | 3.6 GiB | 4 GiB | 5.6 GiB | 23.7% | $29.37/mo |
| druid/druid-reporting-coordinators:druid-reporting-coordinators | | AWS Cluster #1/cluster-one | 44.3m | 1000m | 69m | 2.3 GiB | 4 GiB | 3.5 GiB | 21.4% | $19.74/mo |
| kube-system/kube-proxy:kube-proxy | ⚠ | AWS Cluster #1/cluster-one | 2.1m | 100m | 10m | 30.3 MiB | 0 B | 46.6 MiB | 3.2% | $19.02/mo |
| druid/druid-reporting-historicals:druid-reporting-historicals | ⚠ | AWS Cluster #1/cluster-one | 21.7m | 500m | 34m | 1023.1 MiB | 1 GiB | 1.5 GiB | 21.3% | $18.51/mo |
| druid/druid-reporting-routers:druid-reporting-routers | ⚠ | AWS Cluster #1/cluster-one | 5.3m | 500m | 10m | 458.5 MiB | 512 MiB | 705[illegible]B | [illegible] | $9.73/mo |

我们再看看 Apache Druid：



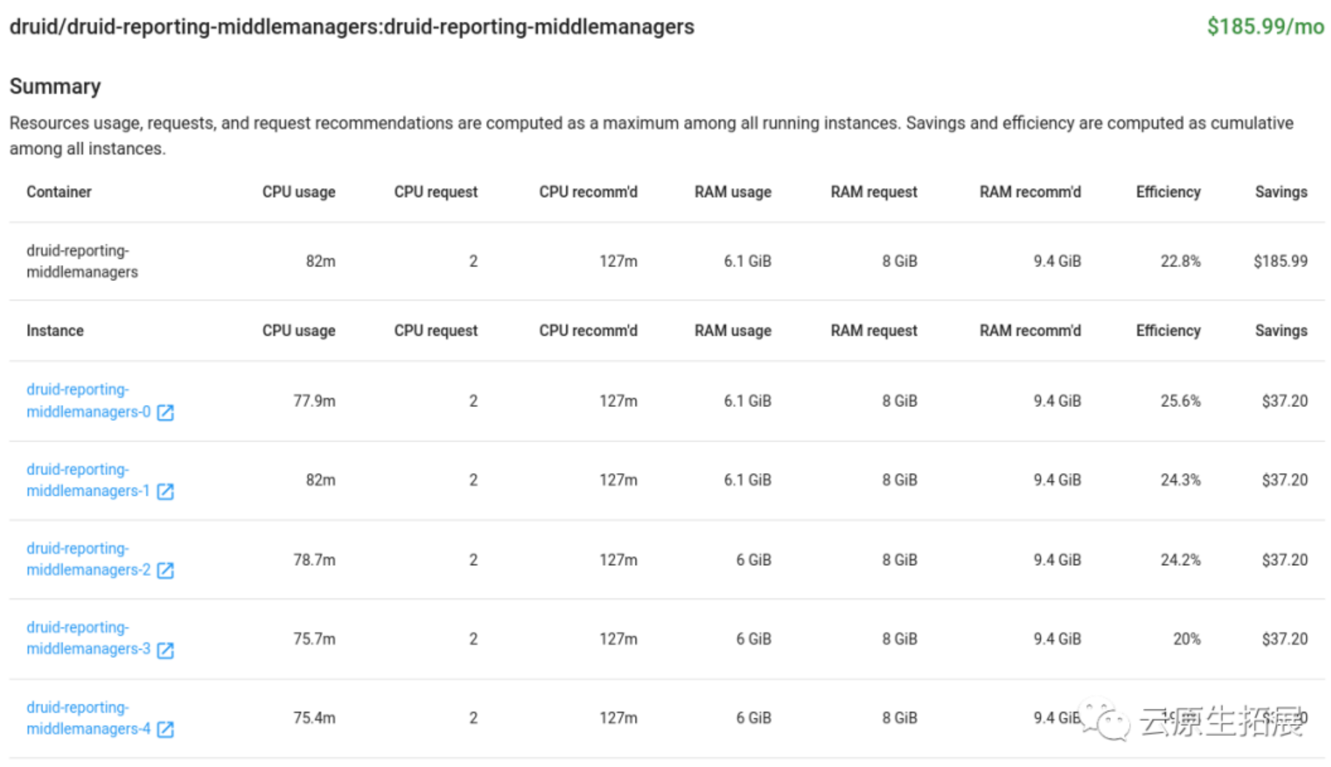
druid/druid-reporting-middlemanagers:druid-reporting-middlemanagers $185.99/mo

Summary

Resources usage, requests, and request recommendations are computed as a maximum among all running instances. Savings and efficiency are computed as cumulative among all instances.

| Container | CPU usage | CPU request | CPU recomm'd | RAM usage | RAM request | RAM recomm'd | Efficiency | Savings |
|---|---|---|---|---|---|---|---|---|
| druid-reporting-middlemanagers | 82m | 2 | 127m | 6.1 GiB | 8 GiB | 9.4 GiB | 22.8% | $185.99 |
| **Instance** | **CPU usage** | **CPU request** | **CPU recomm'd** | **RAM usage** | **RAM request** | **RAM recomm'd** | **Efficiency** | **Savings** |
| druid-reporting-middlemanagers-0 | 77.9m | 2 | 127m | 6.1 GiB | 8 GiB | 9.4 GiB | 25.6% | $37.20 |
| druid-reporting-middlemanagers-1 | 82m | 2 | 127m | 6.1 GiB | 8 GiB | 9.4 GiB | 24.3% | $37.20 |
| druid-reporting-middlemanagers-2 | 78.7m | 2 | 127m | 6 GiB | 8 GiB | 9.4 GiB | 24.2% | $37.20 |
| druid-reporting-middlemanagers-3 | 75.7m | 2 | 127m | 6 GiB | 8 GiB | 9.4 GiB | 20% | $37.20 |
| druid-reporting-middlemanagers-4 | 75.4m | 2 | 127m | 6 GiB | 8 GiB | 9.4 GiB | [illegible] | $37.20 |

这是对 CPU 的过度请求，Kubecost 建议减少这些请求：



druid/druid-reporting-middlemanagers:druid-reporting-middlemanagers $185.99/mo

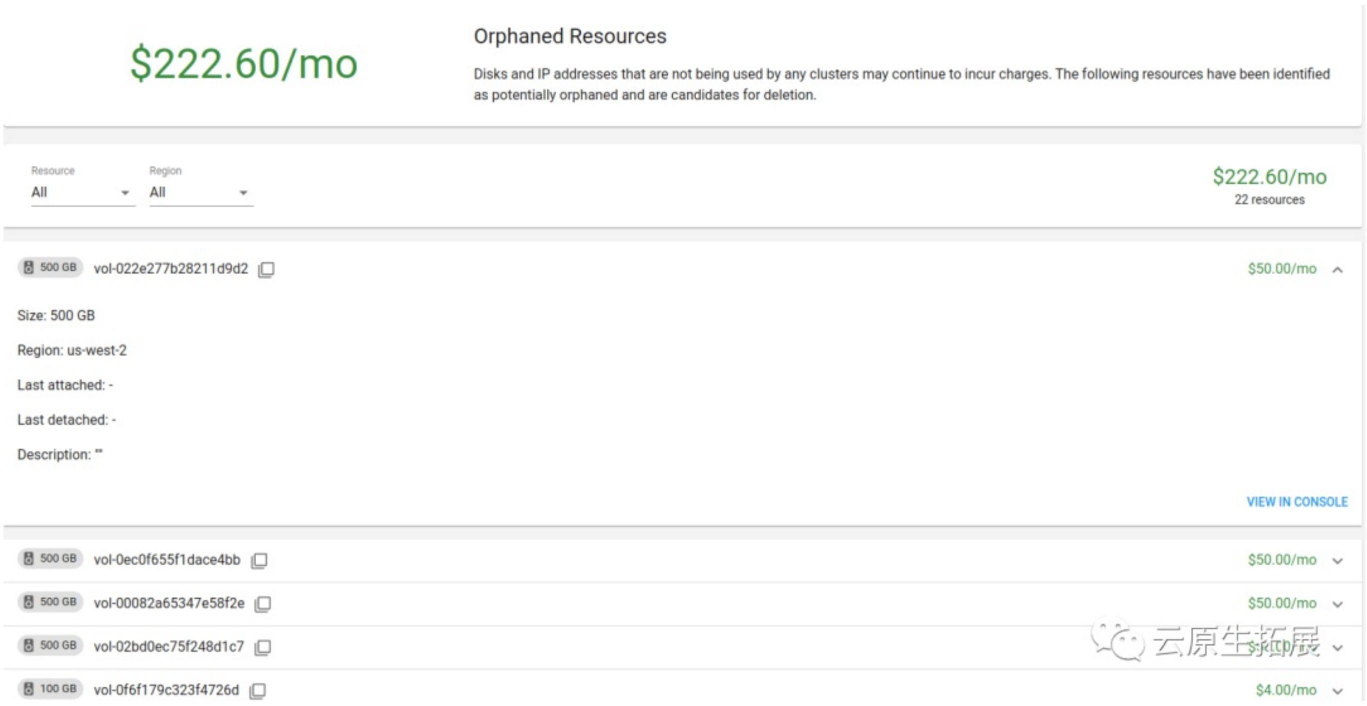
Summary

Resources usage, requests, and request recommendations are computed as a maximum among all running instances. Savings and efficiency are computed as cumulative among all instances.

| Container | CPU usage | CPU request | CPU recomm'd | RAM usage | RAM request | RAM recomm'd | Efficiency | Savings |
|---|---|---|---|---|---|---|---|---|
| druid-reporting-middlemanagers | 82m | 2 | 127m | 6.1 GiB | 8 GiB | 9.4 GiB | 22.8% | $185.99 |
| **Instance** | **CPU usage** | **CPU request** | **CPU recomm'd** | **RAM usage** | **RAM request** | **RAM recomm'd** | **Efficiency** | **Savings** |
| druid-reporting-middlemanagers-0 | 77.9m | 2 | 127m | 6.1 GiB | 8 GiB | 9.4 GiB | 25.6% | $37.20 |
| druid-reporting-middlemanagers-1 | 82m | 2 | 127m | 6.1 GiB | 8 GiB | 9.4 GiB | 24.3% | $37.20 |
| druid-reporting-middlemanagers-2 | 78.7m | 2 | 127m | 6 GiB | 8 GiB | 9.4 GiB | 24.2% | $37.20 |
| druid-reporting-middlemanagers-3 | 75.7m | 2 | 127m | 6 GiB | 8 GiB | 9.4 GiB | 20% | $37.20 |
| druid-reporting-middlemanagers-4 | 75.4m | 2 | 127m | 6 GiB | 8 GiB | 9.4 GiB | [illegible] | $37.20 |

但是我们已经在上面谈到了 Druid——有 JVM，对于每个 MiddleManager，我们运行一个 Supervisor 和两个 Task，并且最好为每个 Task 分配一个完整的核心。所以，让它保持原样。

有用的部分是“删除未分配的资源”——例如，我们发现了一堆未使用的 EBS：



$222.60/mo

Orphaned Resources

Disks and IP addresses that are not being used by any clusters may continue to incur charges. The following resources have been identified as potentially orphaned and are candidates for deletion.

Resource All | Region All | $222.60/mo 22 resources

500 GB vol-022e277b28211d9d2 $50.00/mo

Size: 500 GB

Region: us-west-2

Last attached: -

Last detached: -

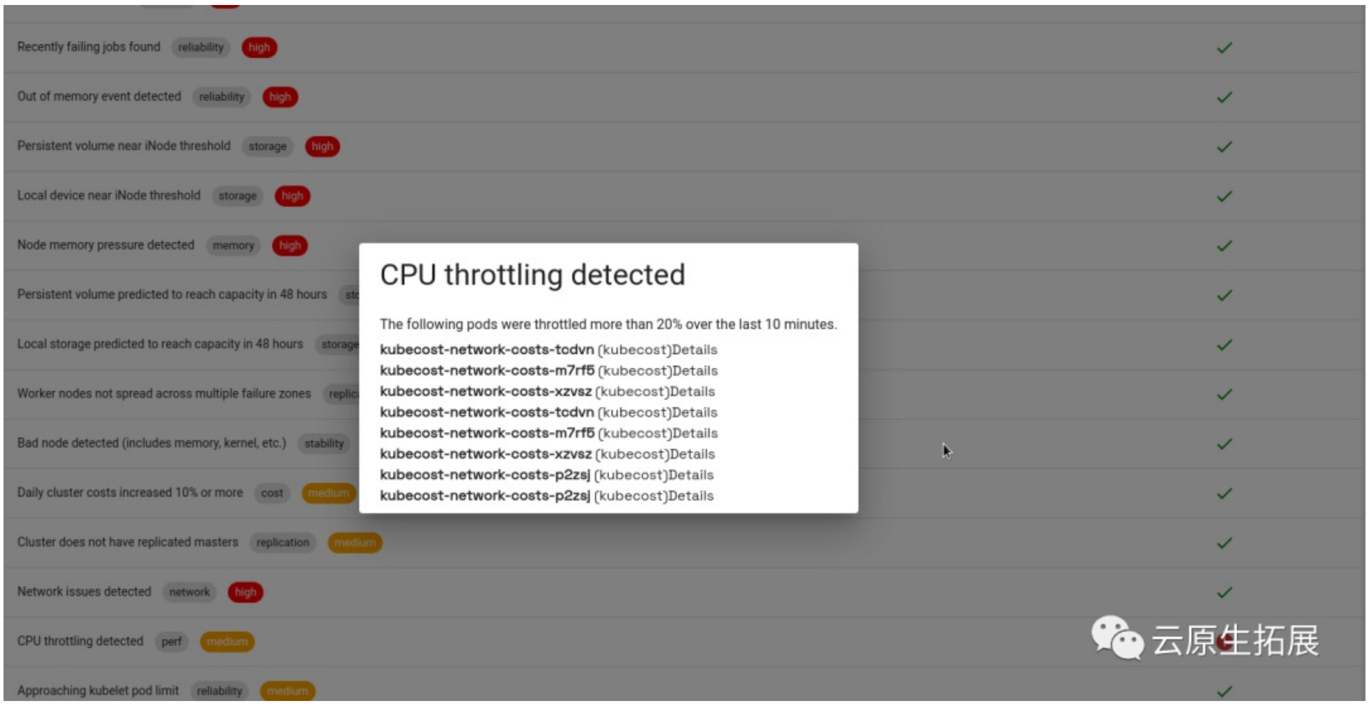
Description: ""

VIEW IN CONSOLE

| | | |
|---|---|---|
| 500 GB | vol-0ec0f655f1dace4bb | $50.00/mo |
| 500 GB | vol-00082a65347e58f2e | $50.00/mo |
| 500 GB | vol-02bd0ec75f248d1c7 | $50.00/mo |
| 100 GB | vol-0f6f179c323f4726d | $4.00/mo |

## Health

此外，还有一个有用的东西可以显示集群的主要问题：



| | | | |
|---|---|---|---|
| Recently failing jobs found | reliability | high | ✓ |
| Out of memory event detected | reliability | high | ✓ |
| Persistent volume near iNode threshold | storage | high | ✓ |
| Local device near iNode threshold | storage | high | ✓ |
| Node memory pressure detected | memory | high | ✓ |
| Persistent volume predicted to reach capacity in 48 hours | | | ✓ |
| Local storage predicted to reach capacity in 48 hours | storage | | ✓ |
| Worker nodes not spread across multiple failure zones | | | ✓ |
| Bad node detected (includes memory, kernel, etc.) | stability | | ✓ |
| Daily cluster costs increased 10% or more | cost | medium | ✓ |
| Cluster does not have replicated masters | replication | medium | ✓ |
| Network issues detected | network | high | ✓ |
| CPU throttling detected | perf | medium | |
| Approaching kubelet pod limit | reliability | medium | ✓ |

CPU throttling detected

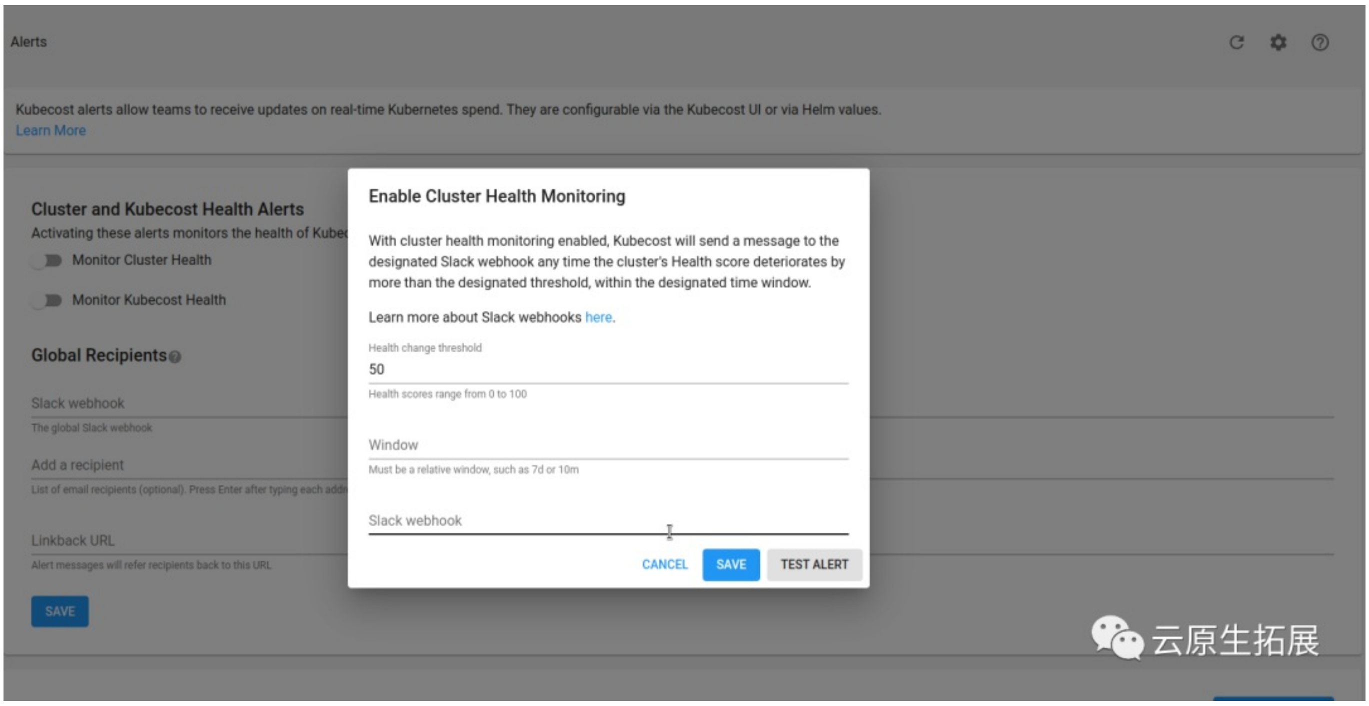
The following pods were throttled more than 20% over the last 10 minutes.

kubecost-network-costs-tcdvn (kubecost)Details
kubecost-network-costs-m7rf5 (kubecost)Details
kubecost-network-costs-xzvsz (kubecost)Details
kubecost-network-costs-tcdvn (kubecost)Details
kubecost-network-costs-m7rf5 (kubecost)Details
kubecost-network-costs-xzvsz (kubecost)Details
kubecost-network-costs-p2zsj (kubecost)Details
kubecost-network-costs-p2zsj (kubecost)Details

kubecost-network-costs 主动使用 CPU，超出其请求，而 Kubernetes 对其进行了限制。

## Alerts

在这里我们可以设置警报，但我只能通过 Slack Webhook 设置发送：



Alerts

Kubecost alerts allow teams to receive updates on real-time Kubernetes spend. They are configurable via the Kubecost UI or via Helm values.
Learn More

Cluster and Kubecost Health Alerts
Activating these alerts monitors the health of Kubec

Monitor Cluster Health

Monitor Kubecost Health

Global Recipients

Slack webhook
The global Slack webhook

Add a recipient
List of email recipients (optional). Press Enter after typing each addr

Linkback URL
Alert messages will refer recipients back to this URL

SAVE

Enable Cluster Health Monitoring

With cluster health monitoring enabled, Kubecost will send a message to the designated Slack webhook any time the cluster's Health score deteriorates by more than the designated threshold, within the designated time window.

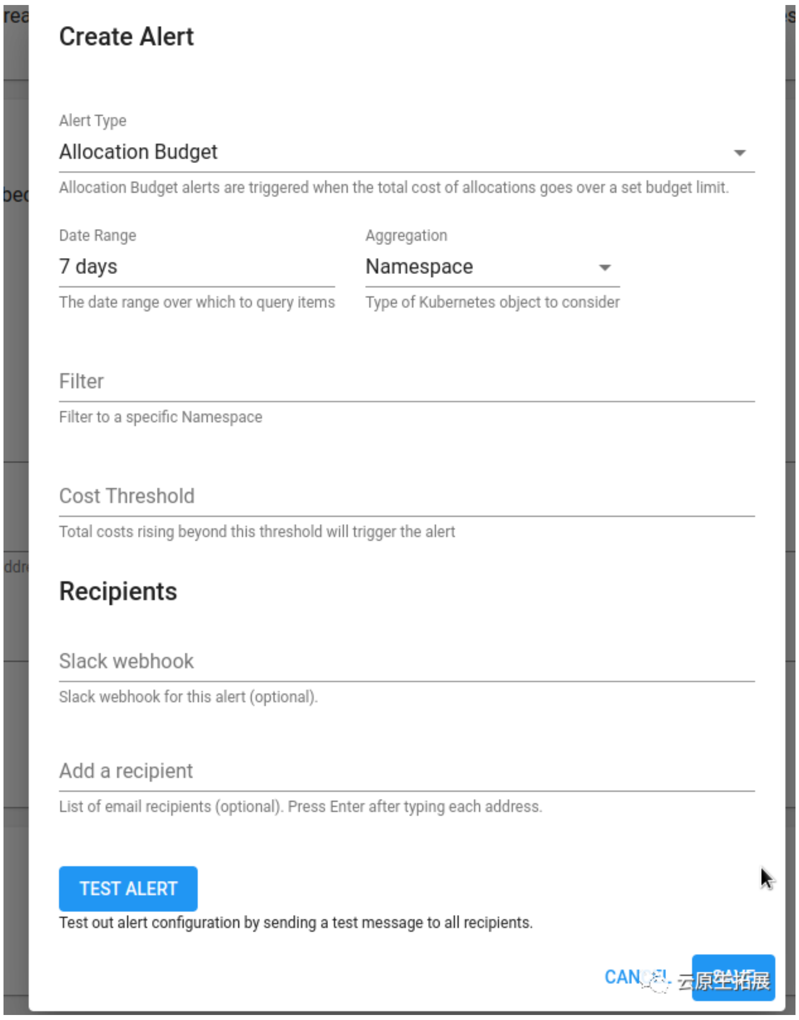
Learn more about Slack webhooks here.

Health change threshold
50
Health scores range from 0 to 100

Window
Must be a relative window, such as 7d or 10m

Slack webhook

CANCEL SAVE TEST ALERT



Create Alert

Alert Type
Allocation Budget
Allocation Budget alerts are triggered when the total cost of allocations goes over a set budget limit.

Date Range
7 days
The date range over which to query items

Aggregation
Namespace
Type of Kubernetes object to consider

Filter
Filter to a specific Namespace

Cost Threshold
Total costs rising beyond this threshold will trigger the alert

Recipients

Slack webhook
Slack webhook for this alert (optional).

Add a recipient
List of email recipients (optional). Press Enter after typing each address.

TEST ALERT
Test out alert configuration by sending a test message to all recipients.

CAN

文档 - 警报(https://docs.kubecost.com/using-kubecost/getting-started/alerts)。

Prometheus Alertmanager 可以通过 values 配置，但它使用自己的内置的，它与 Prometheus 一起运行，但我没有找到如何为它设置路由。

可以在 Kubecost 中配置的警报示例：

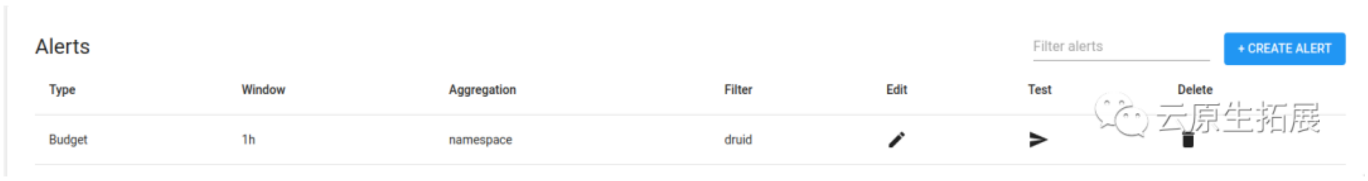
```
global:
  notifications:
    alertConfigs:
      alerts:
        - type: budget
          threshold: 1
          window: 1d
          aggregation: namespace
          filter: druid
    alertmanager:
      enabled: true
      fqdn: http://prometheus-kube-prometheus-alertmanager.monitoring.svc
```

在这里，我们添加了一个类型为 budget 的警报，我们在其中检查了过去 1 天命名空间 druid 的成本，并在它变得比 1 美元更贵时发出警报。

更新设置：

```
[root@~]helm upgrade --install -n kubecost -f values.yaml kubecost kubecost/cost-analyzer
```

警报出现在列表中：



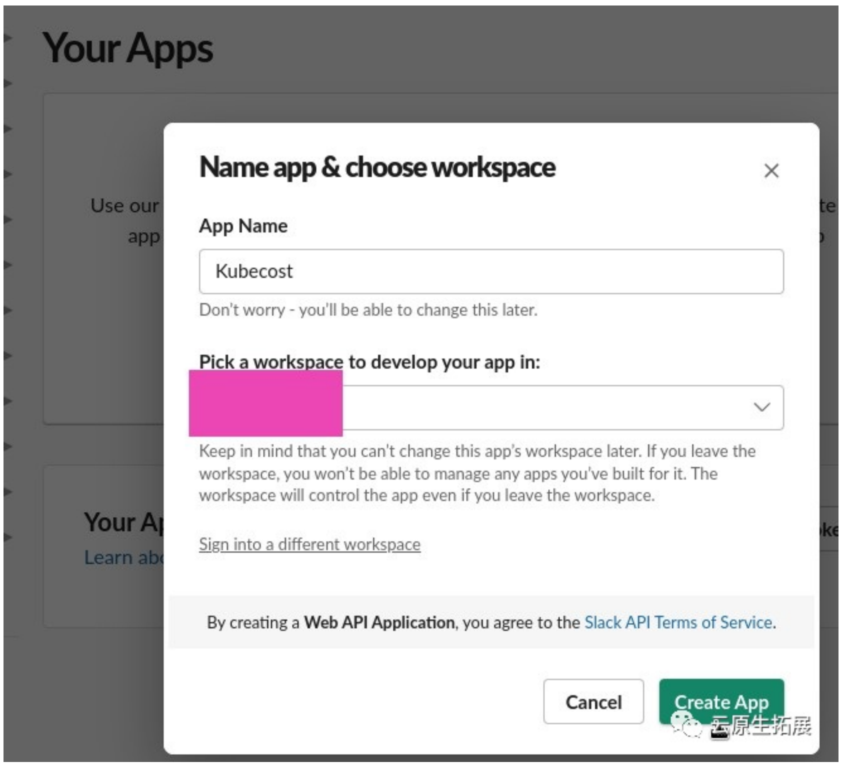
| Type | Window | Aggregation | Filter | Edit | Test | Delete |
|---|---|---|---|---|---|---|
| Budget | 1h | namespace | druid | | | |

但不响应按下测试按钮，并且警报不会出现在本地 Alertmanager 中。

## Slack webhook

让我们用 Slack webhook 试试看，文档在这里>>>https://api.slack.com/messaging/webhooks。

创建一个应用程序：



Name app & choose workspace

App Name

Kubecost

Don't worry - you'll be able to change this later.

Pick a workspace to develop your app in:

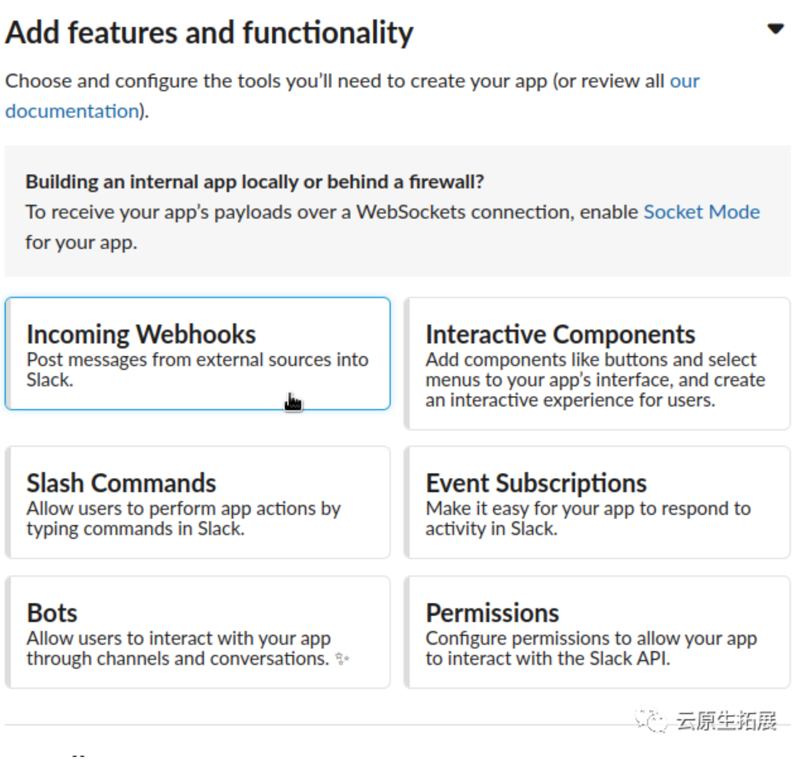
[redacted]

Keep in mind that you can't change this app's workspace later. If you leave the workspace, you won't be able to manage any apps you've built for it. The workspace will control the app even if you leave the workspace.

Sign into a different workspace

By creating a **Web API Application**, you agree to the Slack API Terms of Service.

转到 Webhook：


## Add features and functionality

Choose and configure the tools you'll need to create your app (or review all our documentation).

**Building an internal app locally or behind a firewall?**
To receive your app's payloads over a WebSockets connection, enable Socket Mode for your app.

**Incoming Webhooks**
Post messages from external sources into Slack.

**Interactive Components**
Add components like buttons and select menus to your app's interface, and create an interactive experience for users.

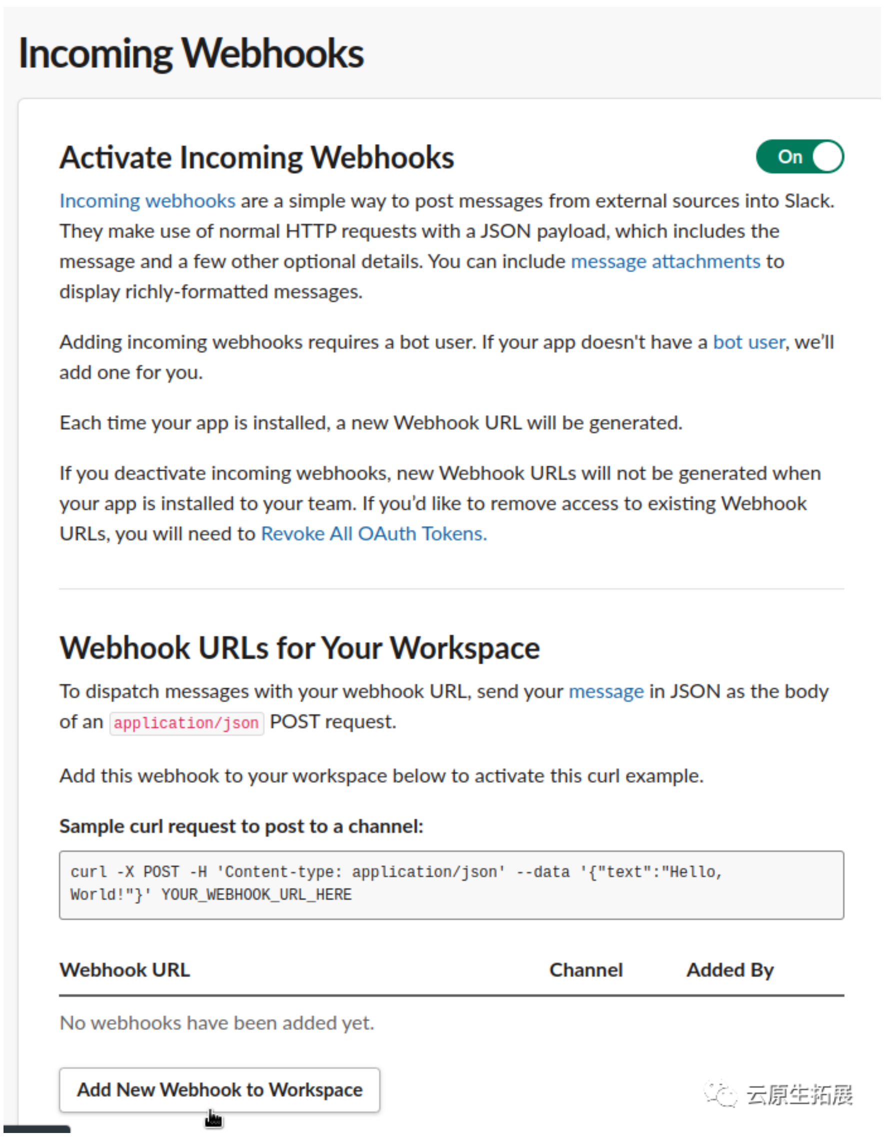
**Slash Commands**
Allow users to perform app actions by typing commands in Slack.

**Event Subscriptions**
Make it easy for your app to respond to activity in Slack.

**Bots**
Allow users to interact with your app through channels and conversations.

**Permissions**
Configure permissions to allow your app to interact with the Slack API.

激活它并单击“添加新 Webhook”：


## Incoming Webhooks

### Activate Incoming Webhooks

On

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include message attachments to display richly-formatted messages.

Adding incoming webhooks requires a bot user. If your app doesn't have a bot user, we'll add one for you.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to Revoke All OAuth Tokens.

### Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your message in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

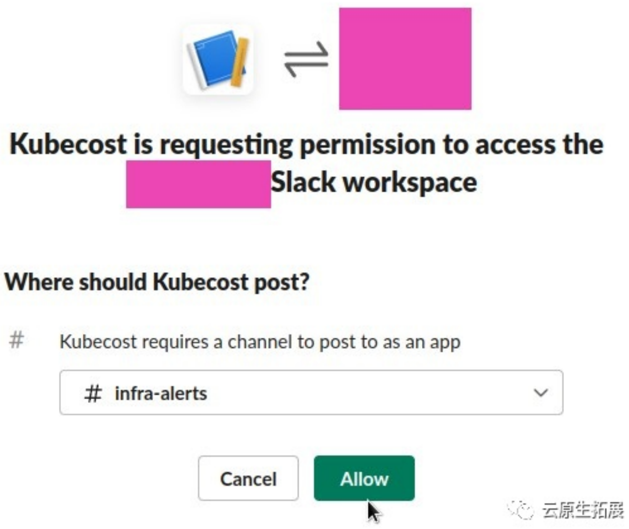
**Sample curl request to post to a channel:**

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello,
World!"}' YOUR_WEBHOOK_URL_HERE
```

| Webhook URL | Channel | Added By |
|---|---|---|
| No webhooks have been added yet. | | |

Add New Webhook to Workspace

选择一个渠道来发送警报：



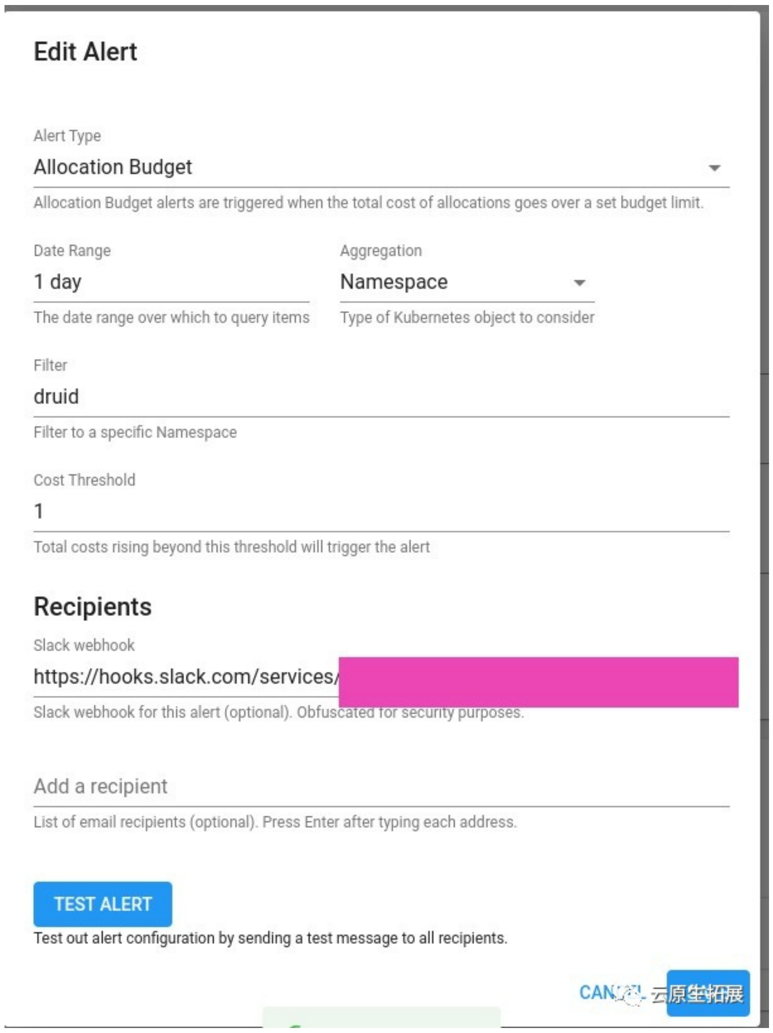
Kubecost is requesting permission to access the [redacted] Slack workspace

Where should Kubecost post?

Kubecost requires a channel to post to as an app

# infra-alerts

将 URL 添加到 Kubecost，并测试：



Edit Alert

Alert Type
Allocation Budget
Allocation Budget alerts are triggered when the total cost of allocations goes over a set budget limit.

Date Range
1 day
The date range over which to query items

Aggregation
Namespace
Type of Kubernetes object to consider

Filter
druid
Filter to a specific Namespace

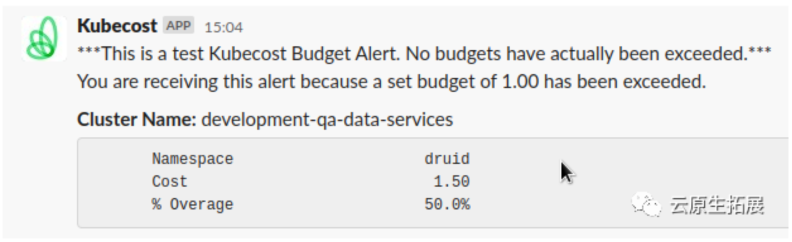
Cost Threshold
1
Total costs rising beyond this threshold will trigger the alert

Recipients

Slack webhook
https://hooks.slack.com/services/[redacted]
Slack webhook for this alert (optional). Obfuscated for security purposes.

Add a recipient
List of email recipients (optional). Press Enter after typing each address.

Test out alert configuration by sending a test message to all recipients.



Kubecost APP 15:04
***This is a test Kubecost Budget Alert. No budgets have actually been exceeded.***
You are receiving this alert because a set budget of 1.00 has been exceeded.

**Cluster Name:** development-qa-data-services

```
Namespace            druid
Cost                  1.50
% Overage            50.0%
```

## 最终 Values.yaml

最后，为了测试，我得到了以下 values：

```yaml
kubecostToken: "c2V***f98"
kubecostProductConfigs:
  clusterName: development-qa-data-services
global:
  notifications:
    alertConfigs:
      globalSlackWebhookUrl: https://hooks.slack.com/services/T03***c1f
      alerts:
        - type: assetBudget
          threshold: 30
          window: 1d
          aggregation: type
          filter: 'Node'

        - type: assetBudget
          threshold: 4
          window: 1d
          aggregation: type
          filter: 'LoadBalancer'

        - type: assetBudget
          threshold: 3
          window: 1d
          aggregation: type
          filter: 'Disk'

        - type: assetBudget
          threshold: 40
          window: 3d
          aggregation: cluster
          filter: 'development-qa-data-services'

        - type: spendChange
          relativeThreshold: 0.01  # change relative to baseline average cost. Must be greater than -1 (can be negative).
          window: 1d
          baselineWindow: 7d       # previous window, offset by window
          aggregation: namespace
          filter: default, druid

        - type: spendChange
          relativeThreshold: 0.01
          window: 1d
          baselineWindow: 7d
          aggregation: cluster
          filter: 'development-qa-data-services'

        - type: health              # Alerts when health score changes by a threshold
          window: 10m
          threshold: 1
prometheus:
  kube-state-metrics:
    disabled: true
  nodeExporter:
    enabled: false
  serviceAccounts:
    nodeExporter:
      create: false

#serviceMonitor:
#  enabled: true
#  additionalLabels:
#    release: prometheus

networkCosts:
  enabled: true
```

```
podMonitor:
  enabled: true
config:
  destinations:
    direct-classification:
    - region: "us-west-2"
      zone: "us-west-2c"
      ips:
        - "10.0.64.0/19"
        - "10.0.160.0/20"
        - "10.0.208.0/21"
    - region: "us-west-2"
      zone: "us-west-2d"
      ips:
        - "10.0.216.0/21"
        - "10.0.96.0/19"
        - "10.0.176.0/20"
  services:
    amazon-web-services: true
```

以下是原则上可以投入生产的测试警报。

我禁用了 ServiceMonitor 以接收外部 Prometheus 中的指标，因为我还没有看到这一点——Kubecost 将通过 Slack Webhook 发出警报并带有自己的警报，内置 Grafana 中的仪表板已经足够好了。

另外，我为 networkCosts 添加了 direct-classification - 让我们看看，也许它会显示更正确的流量数据。

## TODO

我还没有解决的问题：

- 通过 Alertmanager 发出警报
- Kubecost 没有看到 Node Exporter（在 http://localhost:9090/diagnostics 处检查），但这似乎没有影响任何东西——它从 cAdvisor 接收主要指标
- 网络成本太低

未测试：

- 没有为 AWS 设置成本使用报告，请参阅 AWS Cloud Integration
- 没有配置 AWS Spot 实例定价
- 没有添加 Ingress，因为我们有 AWS ALB Controller，需要做授权，Kubecost 中的 SAML 只有 Premium 才有

该系统有趣且有用，但存在一些需要处理的错误和困难。