其中，

- kube-controller-manager 需要使用 hostNetwork 模式，即直接使用宿主机网络。
- 镜像的tag来源于kubernetes发布包中的kube-controller-manager.docker_tag文件：kubernetes/server/kubernetes-server-linux-amd64/server/bin/kube-controller-manager.docker_tag。
- --master：指定 kube-apiserver 服务的 URL 地址。
- --leader-elect=true：使用 leader 选举机制。

```
kube-scheduler.yaml:
apiVersion: v1
kind: Pod
metadata:
  name: kube-scheduler
spec:
  hostNetwork: true
  containers:
  - name: kube-scheduler
    image:
gcr.io/google_containers/kube-scheduler:34d0b8f8b31e27937327961528739bc9
    command:
    - /bin/sh
    - -c
    - /usr/local/bin/kube-scheduler --master=127.0.0.1:8080 --v=2
--leader-elect=true 1>>/var/log/kube-scheduler.log 2>&1
    livenessProbe:
      httpGet:
        path: /healthz
        port: 10251
      initialDelaySeconds: 15
      timeoutSeconds: 1
    volumeMounts:
    - mountPath: /var/log/kube-scheduler.log
      name: logfile
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-s8ejd
      readOnly: true
  volumes:
  - hostPath:
      path: /var/log/kube-scheduler.log
    name: logfile
```

其中，

- kube-scheduler 需要使用 hostNetwork 模式，即直接使用宿主机网络。

- 镜像的 tag 来源于 kubernetes 发布包中的 kube-scheduler.docker_tag 文件：kubernetes/server/kubernetes-server-linux-amd64/server/bin/kube-scheduler.docker_tag。
- --master：指定 kube-apiserver 服务的 URL 地址。
- --leader-elect=true：使用 leader 选举机制。

将这两个 yaml 文件复制到 kubelet 监控的/etc/kubernetes/manifests 目录下，kubelet 将会自动创建 yaml 文件中定义的 kube-controller-manager 和 kube-scheduler 的 Pod。

至此，我们完成了 Kubernetes Master 组件高可用的完整配置，配合 etcd 存储的高可用，整个 Kubernetes 集群的高可用已经全部完成。最后，只需要确认集群中所有访问 API Server 的地方都已经将访问地址修改为负载均衡的地址，就可以保证集群高可用的正常工作了。

3. Master 高可用架构的演进

在当前的版本中，kubelet 可以设置“--api-servers”启动参数来指定多个 kube-apiserver，但是当第 1 个 kube-apiserver 不可用之后，kubelet 无法连接到后面的 kube-apiserver，也就是说只有第 1 个 kube-apiserver 起作用。如果这个问题得到解决，则 kubelet 无须通过额外的负载均衡器就能连接到多个 API Server 了。

另外，除了 kubelet，其他核心组件 kube-controller-manager、kube-scheduler 和 kube-proxy 都需要配置 kube-apiserver，目前它们的启动参数“--master”仅支持配置一个 kube-apiserver，还无法支持多个 kube-apiserver 的配置。

Kubernetes 计划在后续的版本中支持多个 Master 的配置，实现不需要负载均衡器的 Master 高可用架构。

5.1.6　Kubernetes 集群监控

1. 通过 cAdvisor 页面查看容器的运行状态

开源软件 cAdvisor（Container Advisor）是用于监控容器运行状态的利器之一（cAdvisor 项目的主页为 https://github.com/google/cadvisor），它被用于多个与 Docker 相关的开源项目中。

在 Kubernetes 系统中，cAdvisor 已被默认集成到了 kubelet 组件内，当 kubelet 服务启动时，它会自动启动 cAdvisor 服务，然后 cAdvisor 会实时采集所在节点的性能指标及在节点上运行的容器的性能指标。kubelet 的启动参数--cadvisor-port 可自定义 cAdvisor 对外提供服务的端口号，默认为 4194。

cAdvisor 提供了 Web 页面可供浏览器访问。例如 Kubernetes 集群中的一个 Node 的 IP 地址

是 192.168.18.3，则在浏览器中输入网址 http://192.168.18.3:4194 来访问 cAdvisor 的监控页面。cAdvisor 的主页显示了主机的实时运行状态，包括 CPU 使用情况、内存使用情况、网络吞吐量及文件系统使用情况等信息。

图 5.7 展示了 cAdvisor 的几个性能监控页面。

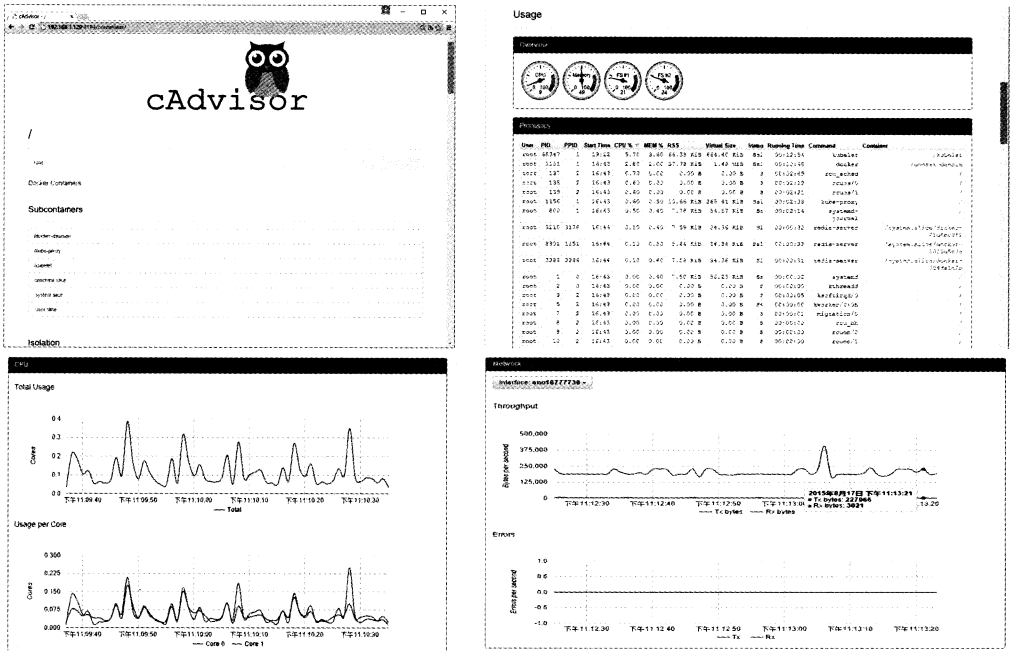


图 5.7　主机的性能监控页面

通过 Docker Containers 链接可以查看容器列表及每个容器的性能数据，如图 5.8 所示。

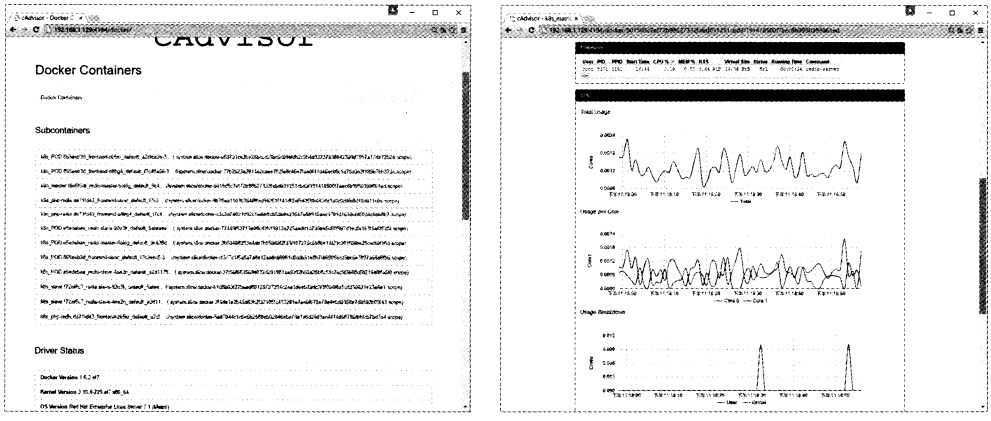


图 5.8　容器的性能监控页面

此外，cAdvisor 也提供了 REST API 供客户端远程调用，主要是为了定制开发，API 返回的数据格式为 JSON，可以采用如下 URL 来访问：

```
http://<hostname>:<port>/api/<version>/<request>
```

例如，通过 URL http://192.168.18.3:4194/api/v1.3/machine 可以获取主机的相关信息：

```
{
    "num_cores":2,
    "cpu_frequency_khz":2793544,
    "memory_capacity":1915408384,
    "machine_id":"0f6233d8256a4ec1a673640e04b8344a",
    "system_uuid":"564D188F-8E82-21C0-6E89-176E2C51EBB5",
    "boot_id":"a03d00d8-ca9c-4d74-a674-ebf5dfbc69d9",
    "filesystems":[
        {
            "device":"/dev/mapper/rhel-root",
            "capacity":18746441728
        },
        {
            "device":"/dev/sda1",
            "capacity":520794112
        }
    ],
    "disk_map":{
        "253:0":{
            "name":"dm-0",
            "major":253,
            "minor":0,
            "size":2147483648,
            "scheduler":"none"
        },
        ......
    },
    "network_devices":[
        {
            "name":"eno16777736",
            "mac_address":"00:0c:29:51:eb:b5",
            "speed":1000,
            "mtu":1500
        }
    ],
    "topology":[
        {
            "node_id":0,
            "memory":2146947072,
            "cores":[
                {
```

```
                "core_id":0,
                "thread_ids":[
                    0
                ],
                "caches":null
            },
            ......
        ],
        "caches":[
            {
                "size":6291456,
                "type":"Unified",
                "level":3
            }
        ]
    }
  ]
}
```


通过下面的 URL 则可以获取节点上最新（1 分钟内）的容器的性能数据：http://192.168.1.129:4194/api/v1.3/subcontainers/system.slice/docker-5015d5c7ef72b98627332fabd031251cbd3f191418500f7aec6b9950399661ed.scope。

结果为：


```
[
  {
"name":"/system.slice/docker-5015d5c7ef72b98627332fabd031251cbd3f191418500f7aec6b9950399661ed.scope",
      "aliases":[
"k8s_master.f8a6f6df_Redis-master-6okig_default_9c428d4f-4167-11e5-afe7-000c2921ba71_5dce2f85",
"5015d5c7ef72b98627332fabd031251cbd3f191418500f7aec6b9950399661ed"
      ],
      "namespace":"docker",
      "spec":{
        "creation_time":"2015-08-17T08:44:27.401122502Z",
        "labels":{
          "io.kubernetes.pod.name":"default/Redis-master-6okig"
        },
        "has_cpu":true,
        "cpu":{
          "limit":2,
          "max_limit":0,
          "mask":"0-1"
        },
        "has_memory":true,
```

```
            "memory":{
                "limit":18446744073709552000,
                "swap_limit":18446744073709552000
            },
            "has_network":true,
            "has_filesystem":false,
            "has_diskio":true
        },
        "stats":[
            {
                "timestamp":"2015-08-18T00:54:26.167988505+08:00",
                "cpu":{
                    "usage":{
                        "total":43121463207,
                        "per_cpu_usage":[
                            21578091763,
                            21543371444
                        ],
                        "user":410000000,
                        "system":13620000000
                    },
                    "load_average":0
                },
                "diskio":{
                    "io_service_bytes":[
                        {
                            "major":253,"minor":14,
                            "stats":{
"Async":8036352,"Read":8036352,"Sync":0,"Total":8036352,"Write":0
                            }
                        }
                    ],
                    "io_serviced":[
                        {
                            "major":8,
                            "minor":0,
                            "stats":{
                                "Async":0,
                                ......
                    ]
                },
                "memory":{
                    "usage":16748544,
                    "working_set":9297920,
                    "container_data":{
                        "pgfault":882,
                        "pgmajfault":8
```

```
                    },
                    "hierarchical_data":{
                        "pgfault":882,
                        "pgmajfault":8
                    }
                },
                "network":{
                    "name":"",
    "rx_bytes":0,"rx_packets":0,"rx_errors":0,"rx_dropped":0,"tx_bytes":0,"tx_pa
ckets":0,"tx_errors":0,"tx_dropped":0
                },
                "task_stats":{
    "nr_sleeping":0,"nr_running":0,"nr_stopped":0,"nr_uninterruptible":0,"nr_io_
wait":0
                }
            },
    ......
        ]
    }
]
```


容器的性能数据对于集群监控非常有用，系统管理员可以根据 cAdvisor 提供的数据进行分析和告警。不过，由于 cAdvisor 是在每台 Node 上运行的，只能采集本机的性能指标数据，所以系统管理员需要对每台 Node 主机单独监控。

针对大型集群，Kubernetes 建议使用几个开源软件组成的集成解决方案来实现对整个集群的监控。这些开源软件包括 Heapster、InfluxDB 及 Grafana 等。

2. Heapster+Influxdb+Grafana 集群性能监控平台搭建

根据前面的说明，cAdvisor 集成在 kubelet 中，运行在每个 Node 上，所以一个 cAdvisor 仅能对一台 Node 进行监控。在大规模容器集群中，需要对所有 Node 和全部容器进行性能监控，Kubernetes 建议使用一套工具来实现集群性能数据的采集、存储和展示：Heapster、InfluxDB 和 Grafana。

- Heapster：对集群中各 Node 上 cAdvisor 的数据采集汇聚的系统，通过访问每个 Node 上 kubelet 的 API，再通过 kubelet 调用 cAdvisor 的 API 来采集该节点上所有容器的性能数据。Heapster 对性能数据进行聚合，并将结果保存到后端存储系统中。Heaspter 支持多种后端存储系统，包括 memory（保存在内存中）、InfluxDB、BigQuery、谷歌云平台提供的 Google Cloud Monitoring（https://cloud.google.com/monitoring/）和 Google Cloud Logging（https://cloud.google.com/logging/）等。Heapster 项目的主页为 https://github.com/kubernetes/heapster。

- InfluxDB：是分布式时序数据库（每条记录都带有时间戳属性），主要用于实时数据采集、事件跟踪记录、存储时间图表、原始数据等。InfluxDB 提供了 REST API 用于数据的存储和查询。InfluxDB 的主页为 http://influxdb.com。
- Grafana：通过 Dashboard 将 InfluxDB 中的时序数据展现成图表或曲线等形式，便于运维人员查看集群的运行状态。Grafana 的主页为 http://grafana.org。

基于 heapster+influxdb+grafana 的集群监控系统总体架构如图 5.9 所示。

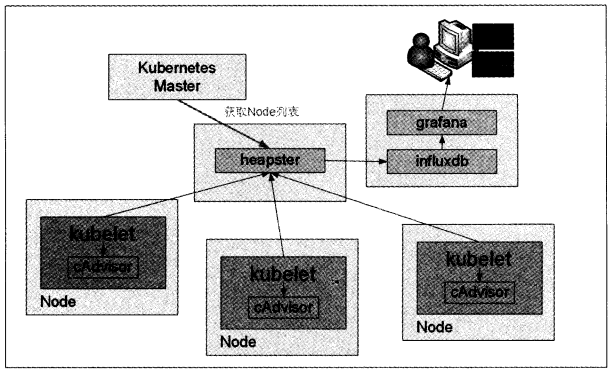


图 5.9　Heapster 集群监控系统架构图

Heapster、InfluxDB 和 Grafana 均以 Pod 的形式启动和运行。由于 Heapster 需要与 Kubernetes Master 进行安全连接，所以需要设置 Master 的 CA 证书安全策略（参见第 2 章的说明）。

1）部署 Heapster、InfluxDB、Grafana 容器应用

先创建它们的 Service：

heapster-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: Heapster
  name: heapster
  namespace: kube-system
spec:
  ports:
    - port: 80
      targetPort: 8082
  selector:
    k8s-app: heapster
```

influxdb-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels: null
  name: monitoring-InfluxDB
  namespace: kube-system
spec:
  type: NodePort
  ports:
    - name: http
      port: 8083
      targetPort: 8083
      nodePort: 8083
    - name: api
      port: 8086
      targetPort: 8086
      nodePort: 8086
  selector:
    name: influxGrafana
```

注意，这里使用 type=NodePort 将 InfluxDB 暴露在宿主机 Node 的端口上，以便我们使用浏览器对其进行访问。

grafana-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  labels:
    kubernetes.io/name: monitoring-Grafana
    kubernetes.io/cluster-service: "true"
  name: monitoring-Grafana
  namespace: kube-system
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 8085
  selector:
    name: influxGrafana
```

同样，使用 type=NodePort 将 Grafana 暴露在 Node 的端口上，以便客户端的浏览器对其进行访问。

使用 kubectl create 命令创建 Services：

```
$ kubectl create -f heapster-service.yaml
$ kubectl create -f InfluxDB-service.yaml
$ kubectl create -f Grafana-service.yaml
```

在创建 heapster 容器之前，先创建 InfluxDB 和 Grafana 的 RC，这两个容器将运行在同一个 Pod 中：

influxdb-grafana-controller-v3.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: monitoring-influxdb-grafana-v3
  namespace: kube-system
  labels:
    k8s-app: influxGrafana
    version: v3
    kubernetes.io/cluster-service: "true"
spec:
  replicas: 1
  selector:
    k8s-app: influxGrafana
    version: v3
  template:
    metadata:
      labels:
        k8s-app: influxGrafana
        version: v3
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
        - image: gcr.io/google_containers/heapster_influxdb:v0.5
          name: influxdb
          resources:
            # keep request = limit to keep this container in guaranteed class
            limits:
              cpu: 100m
              memory: 500Mi
            requests:
              cpu: 100m
              memory: 500Mi
          ports:
            - containerPort: 8083
            - containerPort: 8086
          volumeMounts:
          - name: influxdb-persistent-storage
            mountPath: /data
```

```
        - image: gcr.io/google_containers/heapster_grafana:v2.6.0-2
          name: grafana
          resources:
            limits:
              cpu: 100m
              memory: 100Mi
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            # This variable is required to setup templates in Grafana.
            - name: INFLUXDB_SERVICE_URL
              value: http://monitoring-influxdb:8086
              # The following env variables are required to make Grafana accessible
via
              # the kubernetes api-server proxy. On production clusters, we
recommend
              # removing these env variables, setup auth for grafana, and expose
the grafana
              # service using a LoadBalancer or a public IP.
            - name: GF_AUTH_BASIC_ENABLED
              value: "false"
            - name: GF_AUTH_ANONYMOUS_ENABLED
              value: "true"
            - name: GF_AUTH_ANONYMOUS_ORG_ROLE
              value: Admin
            - name: GF_SERVER_ROOT_URL
              value:
/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/
          volumeMounts:
          - name: grafana-persistent-storage
            mountPath: /var
      volumes:
      - name: influxdb-persistent-storage
        emptyDir: {}
      - name: grafana-persistent-storage
        emptyDir: {}
```

注意，Grafana 容器环境变量 INFLUXDB_SERVICE_URL 设置为 InfluxDB 服务的所在地址。由于 Grafana 与 InfluxDB 处于同一个 Pod 中，所以 Grafana 使用 127.0.0.1 或 localhost 也可以访问到 InfluxDB 服务。

使用 kubectl create 命令创建该 RC：

```
$ kubectl create -f influxdb-grafana-controller-v3.yaml
```

通过 kubectl get pods --namespace=kube-system 确认 Pod 成功启动：

```
# kubectl get pods --namespace=kube-system
NAME                                    READY     STATUS    RESTARTS   AGE
monitoring-influxdb-grafana-v3-uu730    2/2       Running   0          4m
```

创建 heapster 容器，v1.1.0 版本的 heapster 由 4 个容器组合为一个 Pod：

heapster-controller-v1.1.0.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heapster-v1.1.0
  namespace: kube-system
  labels:
    k8s-app: heapster
    kubernetes.io/cluster-service: "true"
    version: v1.1.0
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: heapster
      version: v1.1.0
  template:
    metadata:
      labels:
        k8s-app: heapster
        version: v1.1.0
    spec:
      # 4 containers, 2 heapsters, 2 resizer
      containers:
        - image: gcr.io/google_containers/heapster:v1.1.0
          name: heapster
          resources:
            # keep request = limit to keep this container in guaranteed class
            limits:
              cpu: 100m
              memory: 200Mi
            requests:
              cpu: 100m
              memory: 200Mi
          command:
            - /heapster
            - --source=kubernetes.summary_api:'192.168.18.3:8080'
            - --sink=influxdb:http://monitoring-influxdb:8086
            - --metric_resolution=60s
        - image: gcr.io/google_containers/heapster:v1.1.0
          name: eventer
```

```
      resources:
        # keep request = limit to keep this container in guaranteed class
        limits:
          cpu: 100m
          memory: 200Mi
        requests:
          cpu: 100m
          memory: 200Mi
      command:
        - /eventer
        - --source=kubernetes:'192.168.18.3:8080'
        - --sink=influxdb:http://monitoring-influxdb:8086
    - image: gcr.io/google_containers/addon-resizer:1.3
      name: heapster-nanny
      resources:
        limits:
          cpu: 50m
          memory: 100Mi
        requests:
          cpu: 50m
          memory: 100Mi
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      command:
        - /pod_nanny
        - --cpu=100m
        - --extra-cpu=0m
        - --memory=200Mi
        - --extra-memory=4Mi
        - --threshold=5
        - --deployment=heapster-v1.1.0
        - --container=heapster
        - --poll-period=300000
        - --estimator=exponential
    - image: gcr.io/google_containers/addon-resizer:1.3
      name: eventer-nanny
      resources:
        limits:
          cpu: 50m
          memory: 100Mi
```

```
        requests:
          cpu: 50m
          memory: 100Mi
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      command:
        - /pod_nanny
        - --cpu=100m
        - --extra-cpu=0m
        - --memory=200Mi
        - --extra-memory=500Ki
        - --threshold=5
        - --deployment=heapster-v1.1.0
        - --container=eventer
        - --poll-period=300000
        - --estimator=exponential
```

Heapster 需要设置的启动参数如下。

（1）-source

配置采集来源，为 Master URL 地址：

```
--source=kubernetes.summary_api:'192.168.18.3:8080'
```

（2）-sink

配置后端存储系统，使用 InfluxDB 系统：

```
--sink=InfluxDB:http://monitoring-InfluxDB:8086
```

（3）--metric_resolution

性能指标的精度，60s 表示将过去 60 秒的数据进行汇聚再进行存储。

其他参数可以通过进入 heapster 容器执行 # heapster --help 命令查看和设置。

注意，URL 中的主机名地址使用的是 InfluxDB 的 Service 名字，这需要 DNS 服务正常工作，如果没有配置 DNS 服务，则也可以使用 Service 的 ClusterIP 地址。

值得说明的是，InfluxDB 服务的名称没有加上命名空间，是因为 Heapster 服务与 InfluxDB 服务属于相同的命名空间 kube-system。当然，使用带上命名空间的全服务名也是可以的，例如 http://monitoring-influxdb.kube-system:8086。

使用 kubectl create 命令完成创建该 RC：

```
$ kubectl create -f heapster-controller-v1.1.0.yaml
```

通过 kubectl get pods --namespace=kube-system 确认 Pod 成功启动：

```
# kubectl get deployment --namespace=kube-system
NAME                                READY     STATUS    RESTARTS   AGE
heapster-v1.1.0-1895667918-guisl      4/4       Running   0          3m
```

查看 heapster 的日志，确保 heapster 成功在 influxdb 数据库中创建名为 k8s 的数据库：

```
    # kubectl logs heapster-v1.1.0-1895667918-guisl -c heapster
--namespace=kube-system
    I0706 09:36:15.313587       1 heapster.go:65] /heapster
--source=kubernetes.summary_api:'192.168.18.3:8080'
--sink=influxdb:http://monitoring-influxdb:8086 --metric_resolution=60s
    I0706 09:36:15.313849       1 heapster.go:66] Heapster version 1.1.0
    I0706 09:36:15.314347       1 configs.go:60] Using Kubernetes client with master
"https://169.169.0.1:443" and version "v1"
    I0706 09:36:15.314371       1 configs.go:61] Using kubelet port 10255
    I0706 09:36:15.512107       1 influxdb.go:223] created influxdb sink with options:
host:monitoring-influxdb:8086 user:root db:k8s
    I0706 09:36:15.512154       1 heapster.go:92] Starting with InfluxDB Sink
    I0706 09:36:15.512163       1 heapster.go:92] Starting with Metric Sink
    I0706 09:36:16.414060       1 heapster.go:171] Starting heapster on port 8082
```

2）查询 InfluxDB 数据库中的数据

让我们先通过 InfluxDB 的管理页面查看数据。

由于设置 InfluxDB 服务会暴露到物理 Node 节点上，所以我们可以通过任一 Node 的 8083 端口访问 InfluxDB 数据库提供的管理页面，如图 5.10 所示。通过右上角齿轮按钮可以修改连接属性（用于 influxdb service 设置为非默认端口号的时候）。单击右上角的 Database 下拉列表可以选择数据库，heapster 创建的数据库名为 k8s。

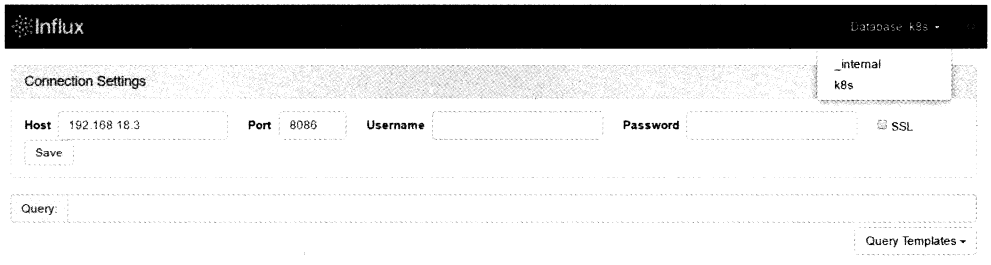


图 5.10　InfluxDB 管理页面

在 Query 输入框中输入“SHOW MEASUREMENTS”，即可查看所有的 measurements（序列表）。图 5.11 显示了部分 measurements。

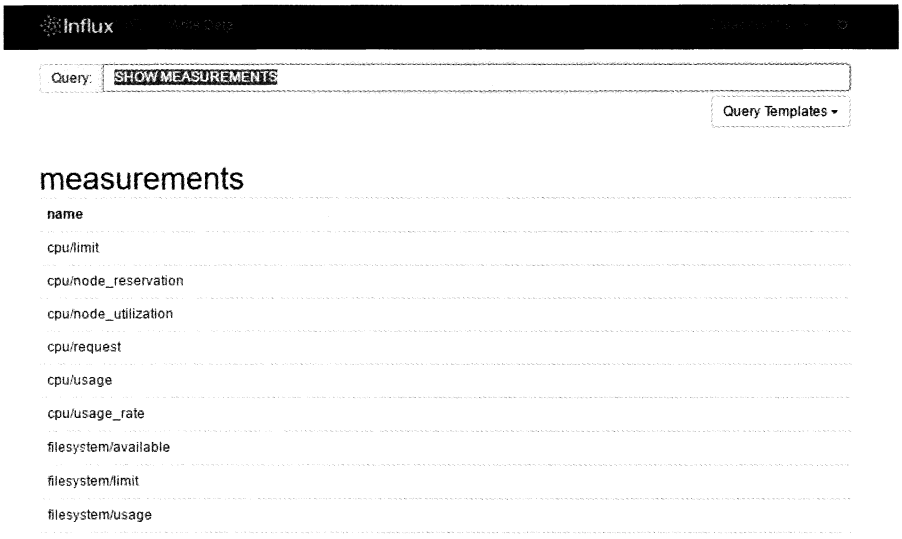


图 5.11　show measurements 结果页面

heapster 采集的全部 metric（性能指标）如表 5.6 所示。

表 5.6　heapster 采集的 metric

metric 名称	说　明
cpu/limit	CPU hard limit，单位为毫秒
cpu/node_reservation	Node 保留的 CPU Share
cpu/node_utilization	Node 的 CPU 使用时间
cpu/request	CPU request，单位为毫秒
cpu/usage	全部 Core 的 CPU 累计使用时间
cpu/usage_rate	全部 Core 的 CPU 累计使用率，单位为毫秒
filesystem/usage	文件系统已用的空间，单位为字节
filesystem/limit	文件系统总空间限制，单位为字节
filesystem/available	文件系统可用的空间，单位为字节
memory/limit	Memory hard limit，单位为字节
memory/major_page_faults	major page faults 数量
memory/major_page_faults_rate	每秒的 major page faults 数量
memory/node_reservation	Node 保留的内存 Share
memory/node_utilization	Node 的内存使用值
memory/page_faults	page faults 数量
memory/page_faults_rate	每秒的 page faults 数量

续表

metric 名称	说　明
memory/request	Memory request，单位为字节
memory/usage	总内存使用量
memory/working_set	总的 Working set usage，Working set 是指不会被 kernel 移除的内存
network/rx	累计接收的网络流量字节数
network/rx_errors	累计接收的网络流量错误数
network/rx_errors_rate	每秒接收的网络流量错误数
network/rx_rate	每秒接收的网络流量字节数
network/tx	累计发送的网络流量字节数
network/tx_errors	累计发送的网络流量错误数
network/tx_errors_rate	每秒发送的网络流量错误数
network/tx_rate	每秒发送的网络流量字节数
uptime	容器启动总时长

每个 metric 可以看作一张数据库表，表中每条记录由一组 label 组成，可以看作字段，如表 5.7 所示。

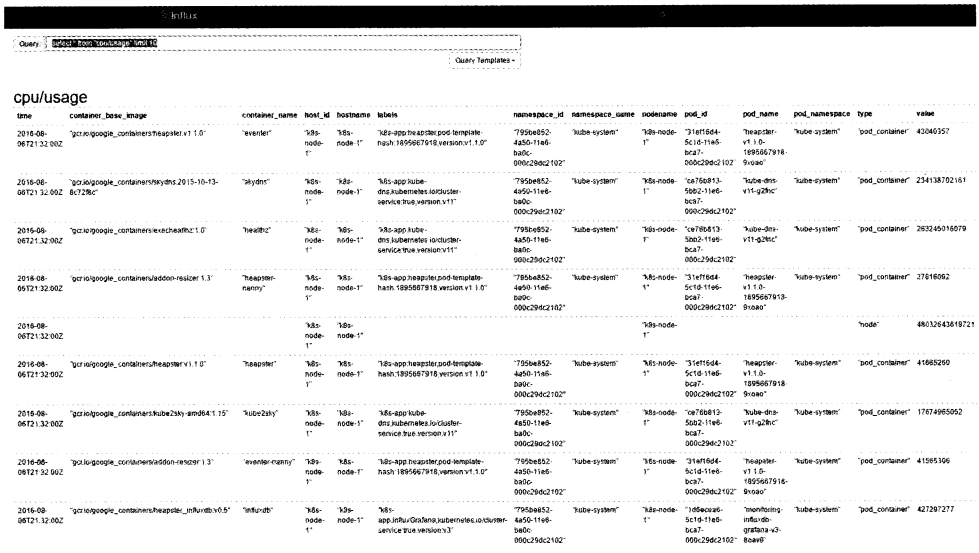
表 5.7　metric 的各 label

Label 名称	说　明
pod_id	系统生成的 Pod 唯一名称
pod_name	用户指定的 Pod 名称
pod_namespace	Pod 所属的 namespace
container_base_image	容器的镜像名称
container_name	用户指定的容器名称
host_id	用户指定的 Node 主机名
hostname	容器运行所在主机名
labels	逗号分隔的 Label 列表
namespace_id	Pod 所属的 namespace 的 UID
resource_id	资源 ID

可以使用标准 SQL SELECT 语句对每个 metric 进行查询，例如查询 CPU 的使用时间：

```
select * from "cpu/usage" limit 10
```

结果如图 5.12 所示。



Query: select * from "cpu/usage_rate" limit 10

Query Templates

cpu/usage

time	container_base_image	container_name	host_id	hostname	labels	namespace_id	namespace_name	nodename	pod_id	pod_name	pod_namespace	type	value
2016-08-06T21:32:00Z	"gcr.io/google_containers/heapster:v1.1.0"	"eventer"	"k8s-node-1"	"k8s-node-1"	"k8s-app:heapster,pod-template-hash:1895667918,version:v1.1.0"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"31ef16d4-5c1d-11e6-bca7-000c29dc2102"	"heapster-v1.1.0-1895667918-9vcao"	"kube-system"	"pod_container"	43046357
2016-08-06T21:32:00Z	"gcr.io/google_containers/skydns:2015-10-13-8c72f8c"	"skydns"	"k8s-node-1"	"k8s-node-1"	"k8s-app:kube-dns,kubernetes.io/cluster-service:true,version:v11"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"ce76b813-5bb2-11e6-bca7-000c29dc2102"	"kube-dns-v11-g2hic"	"kube-system"	"pod_container"	234138702161
2016-08-06T21:32:00Z	"gcr.io/google_containers/exechealthz:1.0"	"healthz"	"k8s-node-1"	"k8s-node-1"	"k8s-app:kube-dns,kubernetes.io/cluster-service:true,version:v11"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"ce76b813-5bb2-11e6-bca7-000c29dc2102"	"kube-dns-v11-g2hic"	"kube-system"	"pod_container"	263245016079
2016-08-06T21:32:00Z	"gcr.io/google_containers/addon-resizer:1.3"	"heapster-nanny"	"k8s-node-1"	"k8s-node-1"	"k8s-app:heapster,pod-template-hash:1895667918,version:v1.1.0"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"31ef16d4-5c1d-11e6-bca7-000c29dc2102"	"heapster-v1.1.0-1895667918-9vcao"	"kube-system"	"pod_container"	27616092
2016-08-06T21:32:00Z			"k8s-node-1"	"k8s-node-1"				"k8s-node-1"				"node"	48032643619721
2016-08-06T21:32:00Z	"gcr.io/google_containers/heapster:v1.1.0"	"heapster"	"k8s-node-1"	"k8s-node-1"	"k8s-app:heapster,pod-template-hash:1895667918,version:v1.1.0"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"31ef16d4-5c1d-11e6-bca7-000c29dc2102"	"heapster-v1.1.0-1895667918-9vcao"	"kube-system"	"pod_container"	41665260
2016-08-06T21:32:00Z	"gcr.io/google_containers/kube2sky-amd64:1.15"	"kube2sky"	"k8s-node-1"	"k8s-node-1"	"k8s-app:kube-dns,kubernetes.io/cluster-service:true,version:v11"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"ce76b813-5bb2-11e6-bca7-000c29dc2102"	"kube-dns-v11-g2hic"	"kube-system"	"pod_container"	17674965052
2016-08-06T21:32:00Z	"gcr.io/google_containers/addon-resizer:1.3"	"eventer-nanny"	"k8s-node-1"	"k8s-node-1"	"k8s-app:heapster,pod-template-hash:1895667918,version:v1.1.0"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"31ef16d4-5c1d-11e6-bca7-000c29dc2102"	"heapster-v1.1.0-1895667918-9vcao"	"kube-system"	"pod_container"	41565306
2016-08-06T21:32:00Z	"gcr.io/google_containers/heapster_influxdb:v0.5"	"influxdb"	"k8s-node-1"	"k8s-node-1"	"k8s-app:influxGrafana,kubernetes.io/cluster-service:true,version:v3"	"795be852-4a50-11e6-ba0c-000c29dc2102"	"kube-system"	"k8s-node-1"	"1d6ecae6-5c19-11e6-bca7-000c29dc2102"	"monitoring-influxdb-grafana-v3-8eav9"	"kube-system"	"pod_container"	427297277

图 5.12　查询 cpu/usage 结果页面

3）Grafana 页面查看和操作

访问 Grafana 服务需要通过 Master 代理模式进行访问，URL 地址为 http://192.168.18.3:8080/api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/。

在 grafana 主页可以查看监控数据的图表展示画面。如图 5.13 所示为 Cluster 集群的整体信息，以折线图的形式展示了集群范围内各 Node 的 CPU 使用率、内存使用情况等信息。

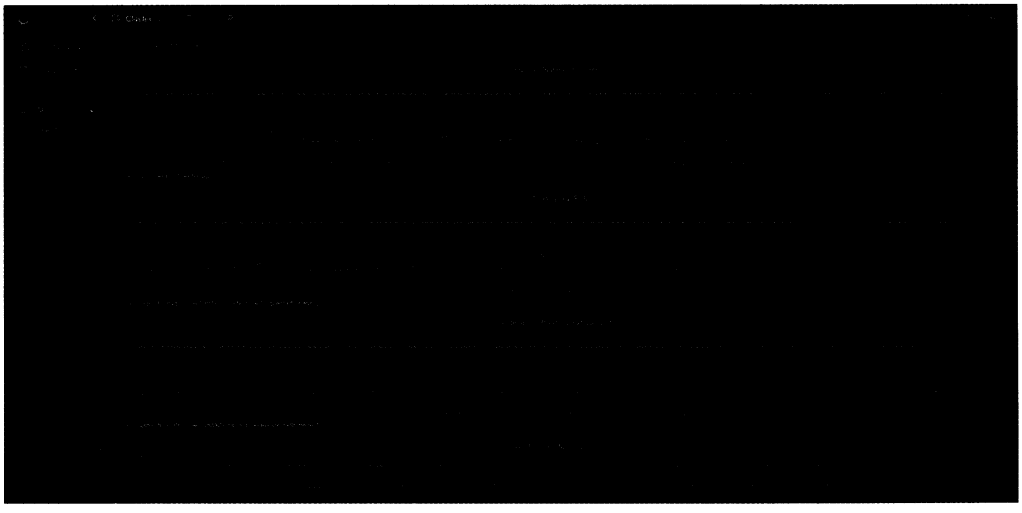


图 5.13　Grafana Cluster 监控页面

图 5.14 显示的是所有 Pod 的信息，以折线图的形式展示了集群范围内各 Pod 的 CPU 使用率、内存使用情况、网络流量、文件系统使用情况等信息。

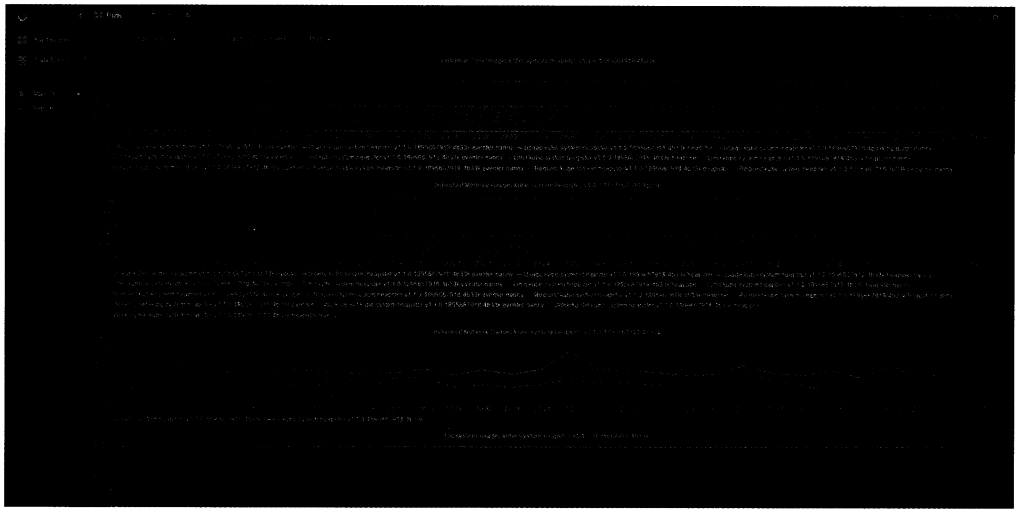


图 5.14　Grafana Pod 监控页面

Grafana 页面上的每个图表都可以进行编辑，在标题上单击鼠标，点击“Edit”进入编辑页面，可以对每个 metric 进行个性化设置，例如查询的表名、字段名、汇总计算等，如图 5.15 所示。

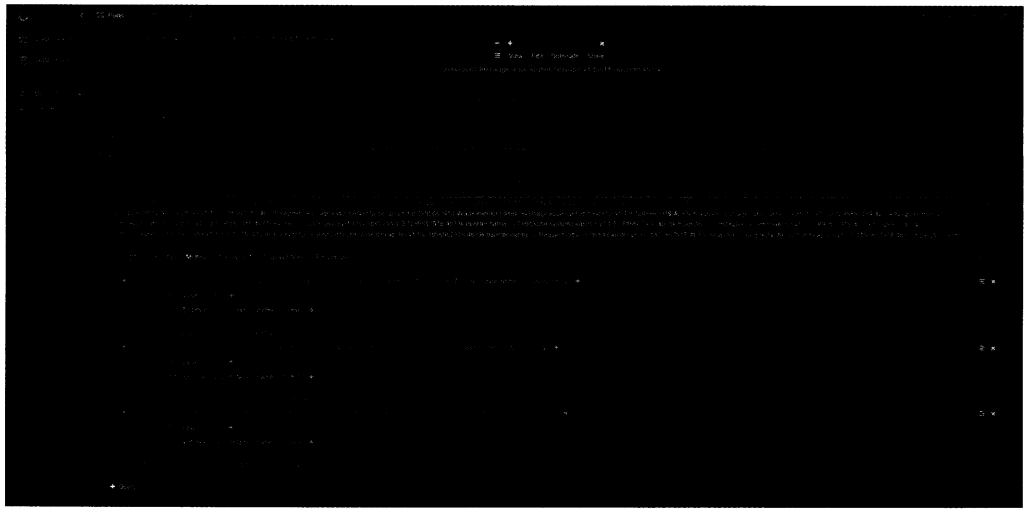


图 5.15　编辑折线图

到此，基于 heapster+influxdb+grafana 的 Kubernetes 集群监控系统就搭建完成了。

5.1.7　kubelet 的垃圾回收（GC）机制

Kubernetes 集群中的垃圾回收（Garbage Collection，简称 GC）机制由 kubelet 完成。kubelet 定期清理不再使用的容器和镜像，每分钟进行一次容器 GC 操作，每 5 分钟进行一次镜像 GC 操作。

1. 容器（Container）的 GC 设置

能够被 GC 清理的容器只能是仅由 kubelet 管理的容器。在 kubelet 所在的 Node 上直接通过 docker run 创建的容器将不会被 kubelet 进行 GC 清理操作。

kubelet 的以下 3 个启动参数用于设置容器 GC 的条件。

- **--minimum-container-ttl-duration**：已停止的容器在被清理之前的最小存活时间，例如“300ms”“10s”或“2h45m”，超过此存活时间的容器将被标记为可被 GC 清理，默认值为 1 分钟。
- **--maximum-dead-containers-per-container**：以 Pod 为单位的可以保留的已停止的（属于同一 Pod 的）容器集的最大数量。有时，Pod 中容器运行失败或者健康检查失败后，会被 kubelet 自动重启，这将产生一些停止的容器。默认值为 2。
- **--maximum-dead-containers**：在本 Node 上保留的已停止容器的最大数量，由于停止的容器也会消耗磁盘空间，所以超过该上限以后，kubelet 会自动清理已停止的容器以释放磁盘空间，默认值为 240。

如果需要关闭针对容器的 GC 操作，则可以将--minimum-container-ttl-duration 设置为 0，将--maximum-dead-containers-per-container 和--maximum-dead-containers 设置为负数。

2. 镜像（Image）的 GC 设置

Kubernetes 系统中通过 imageController 和 kublet 中集成的 cAdvisor 共同管理镜像的生命周期，主要根据本 Node 的磁盘使用率来触发镜像的 GC 操作。

kubelet 的以下 3 个启动参数用于设置镜像 GC 的条件。

- **--minimum-image-ttl-duration**：不再使用的镜像在被清理之前的最小存活时间，例如“300ms”“10s”或“2h45m”，超过此存活时间的镜像被标记为可被 GC 清理，默认值为两分钟。
- **--image-gc-high-threshold**：当磁盘使用率达到该值时，触发镜像的 GC 操作，默认值为 90%。
- **--image-gc-low-threshold**：当磁盘使用率降到该值时，GC 操作结束，默认值为 80%。

删除镜像的机制为：当磁盘使用率达到 image-gc-high-threshold（例如 90%）时触发，GC 操作从最久未使用（Least Recently Used）的镜像开始删除，直到磁盘使用率降为 image-gc-low-threshold（例如 80%）或没有镜像可删为止。

5.2 Kubernetes 高级案例

本节将对 ElasticSearch 日志管理平台的部署、Cassandra 集群的部署及 Kubernetes 中容器的高级应用进行说明。

5.2.1 ElasticSearch 日志搜集查询和展现案例

在 Kubernetes 集群环境中，一个完整的应用或服务都会涉及为数众多的组件运行，各组件所在的 Node 及实例数量都是可变的。日志子系统如果不做集中化管理，则会给系统的运维支撑造成很大的困难，因此有必要在集群层面对日志进行统一的收集和检索等工作。

容器中输出到控制台的日志，都会以*-json.log 的命名方式保存在/var/lib/docker/containers/目录之下，这样就给了我们进行日志采集和后续处理的基础。

Kubernetes 推荐采用 Fluentd+ElasticSearch+Kibana 完成对日志的采集、查询和展现工作。

在部署系统之前，需要以下两个前提条件。

- API Server 正确配置了 CA 证书。
- DNS 服务启动运行。

1. 系统部署架构

系统的逻辑架构如图 5.16 所示。

在各 Node 上运行一个 Fluentd 容器，对本节点/var/log 和/var/lib/docker/containers 两个目录下的日志进程采集，然后汇总到 ElasticSearch 集群，最终通过 Kibana 完成和用户的交互工作。

这里有一个特殊的需求，Fluentd 必须在每个 Node 上运行一份，为了满足这一需要，我们有以下几种不同的方式来部署 Fluentd。

- 直接在 Node 主机上部署 Fluentd。
- 利用 kubelet 的--config 参数，为每个 Node 加载 Fluentd Pod。
- 利用 DaemonSet 来让 Fluentd Pod 在每个 Node 上运行。

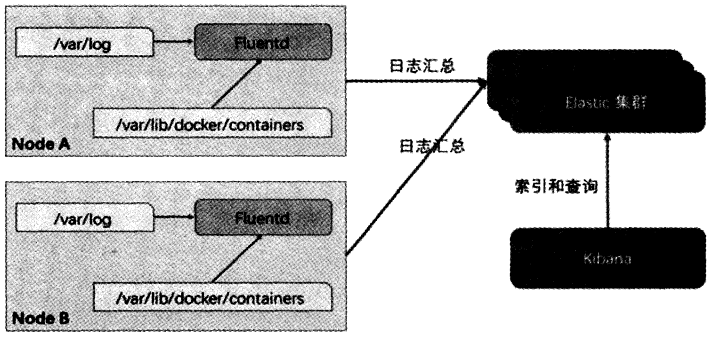


图 5.16　Fluentd+ElasticSearch+Kibana 系统逻辑架构图

目前官方推荐的包括Fluentd、Logstash等日志或者监控类的Pod的运行方式就是DaemonSet方式，因此本节我们也以这一方式进行配置。

2. 创建 ElasticSearch RC 和 Service

ElasticSearch 的 RC 和 Service 定义：

elasticsearch-rc-svc.yml

```
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: elasticsearch-logging-v1
  namespace: kube-system
  labels:
    k8s-app: elasticsearch-logging
    version: v1
    kubernetes.io/cluster-service: "true"
spec:
  replicas: 2
  selector:
    k8s-app: elasticsearch-logging
    version: v1
  template:
    metadata:
      labels:
        k8s-app: elasticsearch-logging
        version: v1
        kubernetes.io/cluster-service: "true"
    spec:
      containers:
      - image: gcr.io/google_containers/elasticsearch:1.8
        name: elasticsearch-logging
```

```
        resources:
          # keep request = limit to keep this container in guaranteed class
          limits:
            cpu: 100m
          requests:
            cpu: 100m
        ports:
        - containerPort: 9200
          name: db
          protocol: TCP
        - containerPort: 9300
          name: transport
          protocol: TCP
        volumeMounts:
        - name: es-persistent-storage
          mountPath: /data
      volumes:
      - name: es-persistent-storage
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: elasticsearch-logging
  namespace: kube-system
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: "Elasticsearch"
spec:
  ports:
  - port: 9200
    protocol: TCP
    targetPort: db
  selector:
    k8s-app: elasticsearch-logging
```

执行 kubectl create -f elastic-search.yml 命令完成创建。

命令成功执行后，首先验证 Pod 的运行情况。通过 kubectl get pods --namespaces=kube-system 获取运行中的 Pod：

```
# kubectl get pods --namespaces=kube-system
NAMESPACE     NAME                                READY     STATUS    RESTARTS  AGE
kube-system   elasticsearch-logging-v1-59qvp      1/1       Running   0         18h
kube-system   elasticsearch-logging-v1-xnv14      1/1       Running   0         18h
```

接下来通过 ElasticSearch 的页面验证其功能。

执行# kubectl cluster-info 命令获取 ElasticSearch 服务的地址：

```
# kubectl cluster-info
Elasticsearch is running at
http://192.168.18.3:8080/api/v1/proxy/namespaces/kube-system/services/elasticsea
rch-logging
```

接下来使用 # kubectl proxy 命令对 apiserver 进行代理，成功执行后输出如下：

```
# kubectl proxy
Starting to serve on 127.0.0.1:8001
```

这样我们就可以在浏览器上访问 URL 地址 http://192.168.18.3:8001/api/v1/proxy/namespaces/kube-system/services/elasticsearch-logging，来验证 ElasticSearch 的运行情况了，返回的内容是一个 JSON 文档：

```
{
"status": 200,
"name": "Emplate",
"cluster_name": "kubernetes-logging",
"version": {
"number": "1.5.2",
"build_hash": "62ff9868b4c8a0c45860bebb259e21980778ab1c",
"build_timestamp": "2015-04-27T09:21:06Z",
"build_snapshot": false,
"lucene_version": "4.10.4"
},
"tagline": "You Know, for Search"
}
```

3. 在每个 Node 上启动 Fluentd

Fluentd 的 DaemonSet 定义如下：

fluentd-ds.yml

```
---
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd-cloud-logging
  namespace: kube-system
  labels:
    k8s-app: fluentd-cloud-logging
spec:
  template:
    metadata:
      namespace: kube-system
      labels:
```

```
      k8s-app: fluentd-cloud-logging
  spec:
    containers:
    - name: fluentd-cloud-logging
      image: gcr.io/google_containers/fluentd-elasticsearch:1.17
      resources:
        limits:
          cpu: 100m
          memory: 200Mi
      env:
      - name: FLUENTD_ARGS
        value: -q
      volumeMounts:
      - name: varlog
        mountPath: /var/log
        readOnly: false
      - name: containers
        mountPath: /var/lib/docker/containers
        readOnly: false
    volumes:
    - name: containers
      hostPath:
        path: /var/lib/docker/containers
    - name: varlog
      hostPath:
        path: /var/log
```

通过 kubectl create 命令创建 Fluentd 容器：

```
# kubectl create -f fluentd-ds.yml
```

查看创建的结果：

```
# kubectl get daemonset
NAME                    DESIRED  CURRENT   NODE-SELECTOR   AGE
fluentd-cloud-logging   3        3         <none>          1h

# kubectl get pods
NAMESPACE     NAME                      READY     STATUS    RESTARTS   AGE
fluentd-cloud-logging-7tw9z             1/1       Running   0          18h
fluentd-cloud-logging-aqdn1             1/1       Running   0          18h
fluentd-cloud-logging-o4usx             1/1       Running   0          18h
```

结果显示 Fluentd DaemonSet 正常运行，启动 3 个 Pod，与集群中的 Node 数量一致。

接下来，使用 # kubectl logs fluentd-cloud-logging-7tw9z 命令查看 Pod 的日志，在 ElasticSearch 正常工作的情况下，我们会看到类似下面这样的日志内容：

```
# kubectl logs fluentd-cloud-logging-7tw9z
Connection opened to Elasticsearch cluster =>
```