Kubernetes bootcamp

邓德源 才云科技

deyuan@caicloud.io

主要内容

- 基本概念
- 架构设计
- 工作流程

Kubernetes 基本概念

什么是 kubernetes

Kubernetes (k8s) 是 Google 开源的容器集群管理系统，前身是 Borg，用于容器的部署、自动化调度和扩展机制等。

主要特性：

- 容器的自动化部署
- 自动化扩展或者缩容
- 容器可成组，对外提供服务，支持负载均衡
- 轻松实现应用镜像升级
- 容器的健康检查，自动重启
- 基于插件机制保证扩展性

API definition

```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

kubernetes 大版本号

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

API definition

```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

资源类型

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

API definition

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
spec:
  containers:
  dnsPolicy: ClusterFirst
  nodeName: i-2zea47skez7ye2xr438v
  restartPolicy: Always
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  volumes: xxx
status:
  conditions: xxx
  hostIP: 10.44.164.150
  phase: Running
  podIP: 192.168.79.9
  startTime: 2016-11-22T14:54:57Z
```

元数据

```
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: minikube
  name: minikube
  resourceVersion: "1027609"
  selfLink: /api/v1/nodesminikube
  uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
  externalID: minikube
status:
  addresses:
  - address: 192.168.99.101
    type: LegacyHostIP
  - address: 192.168.99.101
    type: InternalIP
  - address: minikube
    type: Hostname
```

API definition

```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

资源说明书，配置

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

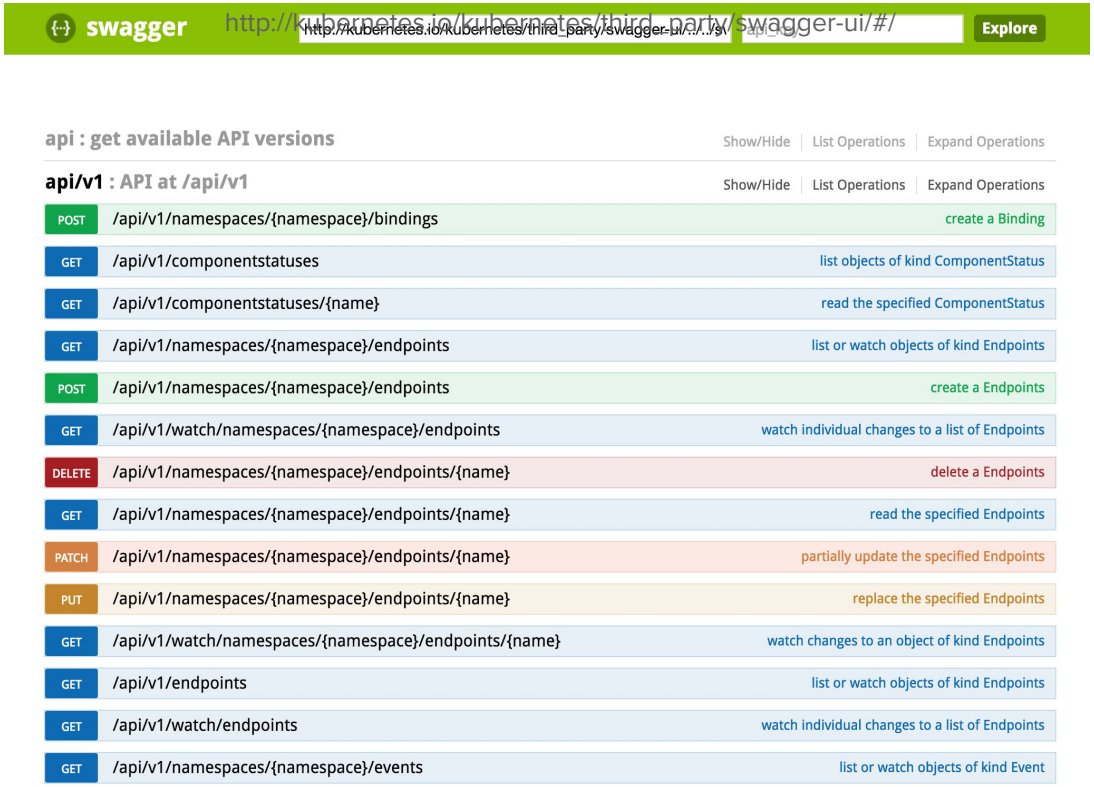
API definition

```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

资源状态

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

API endpoints



API group

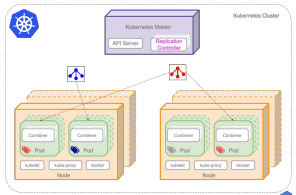
- 每个 API group 中的资源可以单独的 enable/disable
- 每个 API group 有不同的版本，单独开发维护


```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/apps",
    "/apis/apps/v1alpha1",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1beta1",
    "/apis/autoscaling",
    "/apis/autoscaling/v1",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v2alpha1",
    "/apis/certificates.k8s.io",
    "/apis/certificates.k8s.io/v1alpha1",
    "/apis/extensions",
    "/apis/extensions/v1beta1",
    "/apis/k8s.io",
    "/apis/k8s.io/v1",
    "/apis/policy",
    "/apis/policy/v1alpha1",
    "/apis/rbac.authorization.k8s.io",
    "/apis/rbac.authorization.k8s.io/v1alpha1",
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1beta1",
    "/healthz",
    "/healthz/ping",
    "/logs",
    "/metrics",
    "/swaggerapi/",
    "/ui/",
    "/version"
  ]
}
```

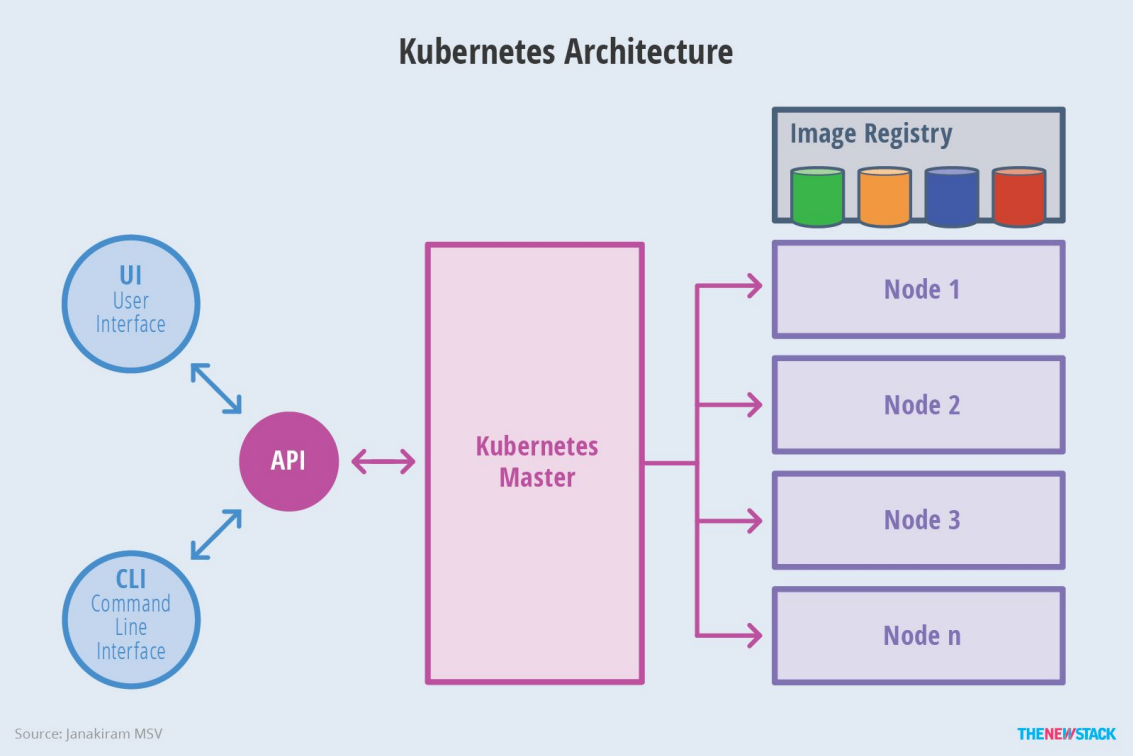
API example: Pod Health Check

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        livenessProbe:
          httpGet:
            # Path to probe; should be cheap, but representative of typical behavior
            path: /index.html
            port: 80
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

POST to “/api/v1/namespaces/default/pods”



Architecture



Source: TheNewStack

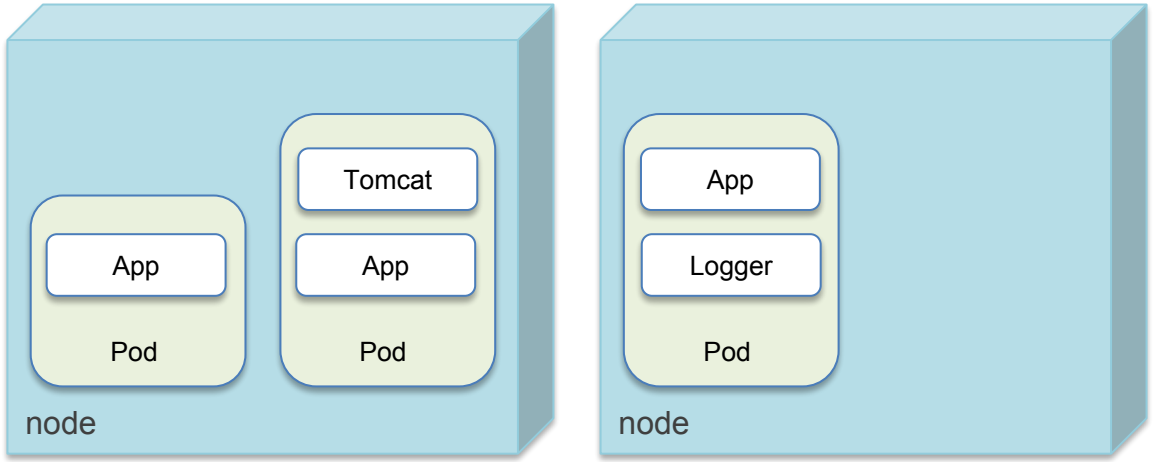
Master/Node

- 物理机/虚拟机
- Master（管理节点）
  - 资源管理
  - Pod 调度
  - 弹性伸缩
  - 安全控制
  - 系统监控
  - 纠错
- Node（工作节点）
  - 运行 Pod 化的服务

```
root@caicloud:/home/vagrant# kubectl get nodes
NAME            STATUS                        AGE
kube-master-1   Ready,SchedulingDisabled   3d
kube-node-1     Ready                         3d
root@caicloud:/home/vagrant#
```

Pod

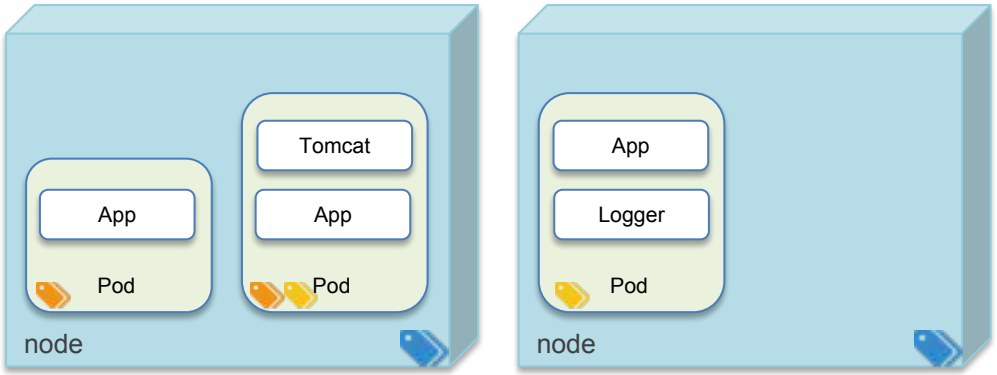
- 最小调度单位
- 多个容器集合
  - 共享生命周期
  - 共享命名空间
  - 共享数据卷
- 其他特性
  - 资源配额
  - 独立 IP
  - 安全策略
  - 等等



Pod demo

Label 和 Selector

- label
  - 用于区分资源的 key/value 键值对
  - 资源可以拥有多个 label
- selector
  - 根据 label 选择对应的资源
  - Equality-based 选择器
  - Set-based 选择器




```
"labels": {
    "environment": "production",
    "tier": "frontend",
    "component": "redis"
}
```

Label 的用途

- 资源分组和过滤
  - kubectl get pods -lapp=guestbook,role=slave
- 批量操作
  - kubectl delete deployment,services -l app=nginx
- 灰度发布

```
name: frontend
replicas: 3
...
labels:
   app: guestbook
   tier: frontend
   track: stable
...
image: gb-frontend:v3
```

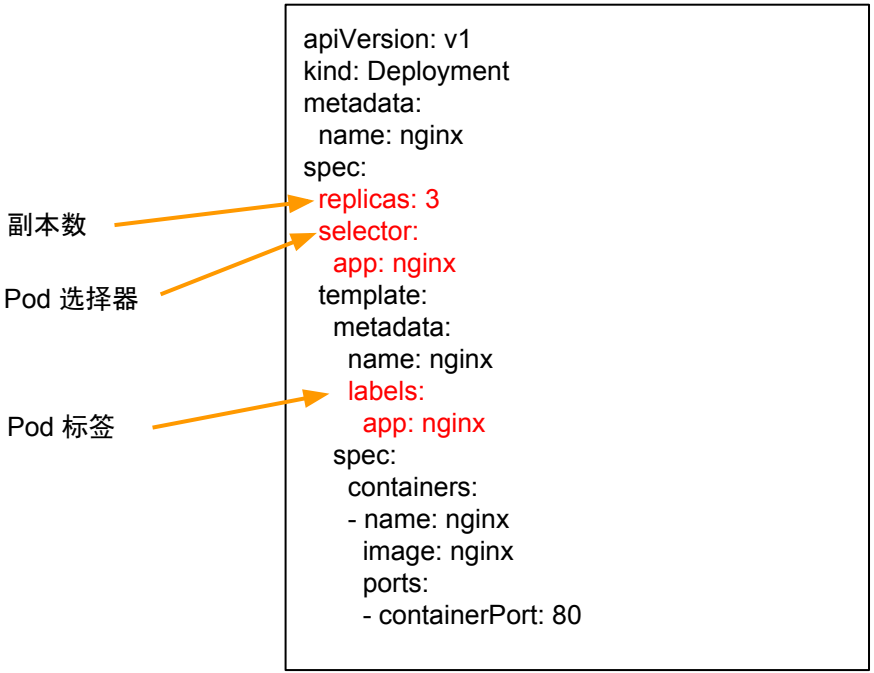
```
name: frontend-canary
replicas: 1
...
labels:
   app: guestbook
   tier: frontend
   track: canary
...
image: gb-frontend:v4
```

```
selector:
   app: guestbook
   tier: frontend
```

Label demo

Deployment

- 确保任意时间都有指定数量的 Pod “副本”在运行
- 通过 Pod 选择器筛选出对应的 Pod 实例并实时监控其状态和数量



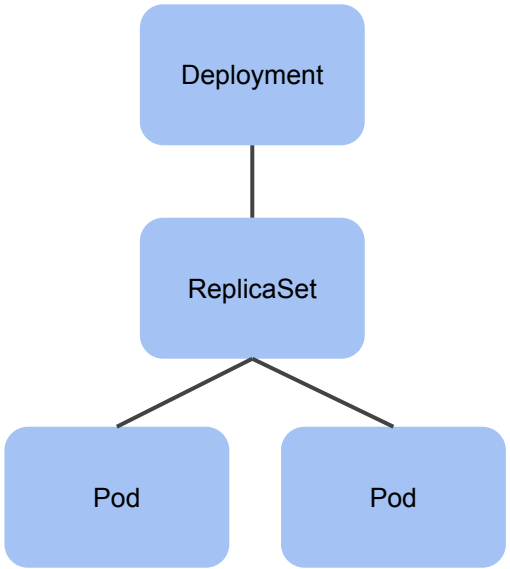
Deployment

- 提供 Pod 和 Replica Set 的声明式更新
- 把部署，滚动更新和回滚进行了自动化
- 创建 Deployment
  - kubectl create -f docs/user-guide/nginx-deployment.yaml --record
- 更新 Deployment
  - kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
- 查看 Deployment 的修订版本
  - kubectl rollout history deployment/nginx-deployment
- 回滚到上一个版本
  - kubectl rollout undo deployment/nginx-deployment
- 回滚到指定的版本
  - kubectl rollout undo deployment/nginx-deployment --to-revision=2

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

ReplicaSet

- ReplicaSet 控制实例数量
  - Deployment = Replicaset + Management (e.g. rollback)
- 如何使用
  - 一般不单独使用
  - Deployment
  - HPA（Horizontal Pod Autoscaler）



Deployment and ReplicaSet Demo

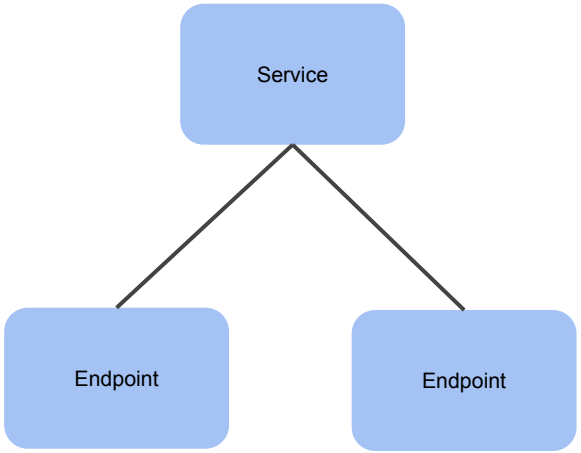
Service

- 一个唯一指定的名字
- 一个虚拟 IP 和端口号
- 服务类型
  - ClusterIP（default）
  - Nodeport
  - LoadBalancer
- 服务发现
  - 环境变量
  - DNS
- 负载均衡

```
{
   "kind": "Service",
   "apiVersion": "v1",
   "metadata": {
      "name": "my-service"
   },
   "spec": {
      "selector": {
         "app": "MyApp"
      },
      "ports": [
         {
            "protocol": "TCP",
            "port": 1234,
            "targetPort": 9376,
            "nodePort": 30029
         }
      ],
      "type":  "NodePort"
   }
}
```

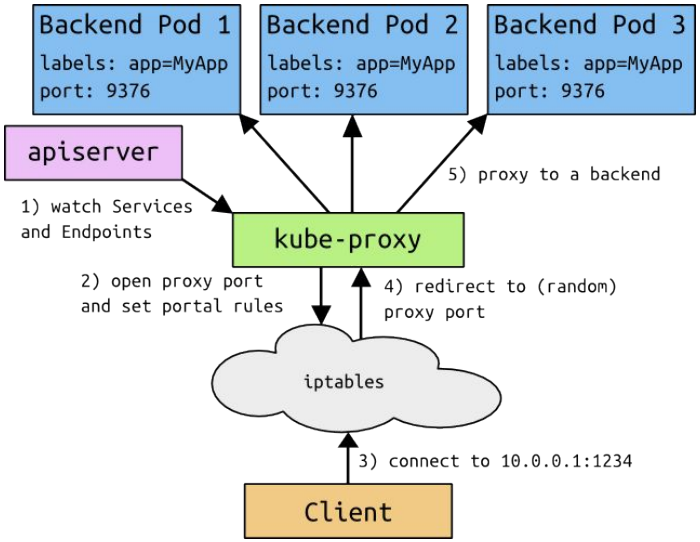
ClusterIP:port

NodeIP:nodePort

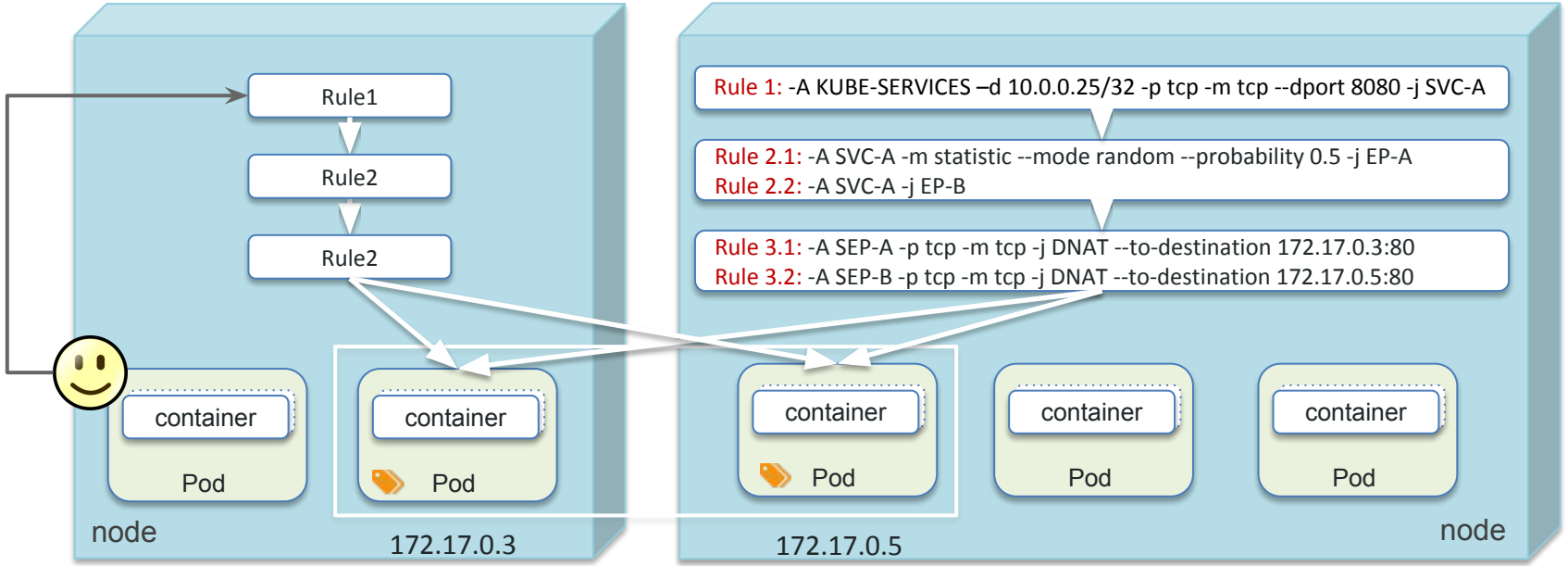


Service (more)

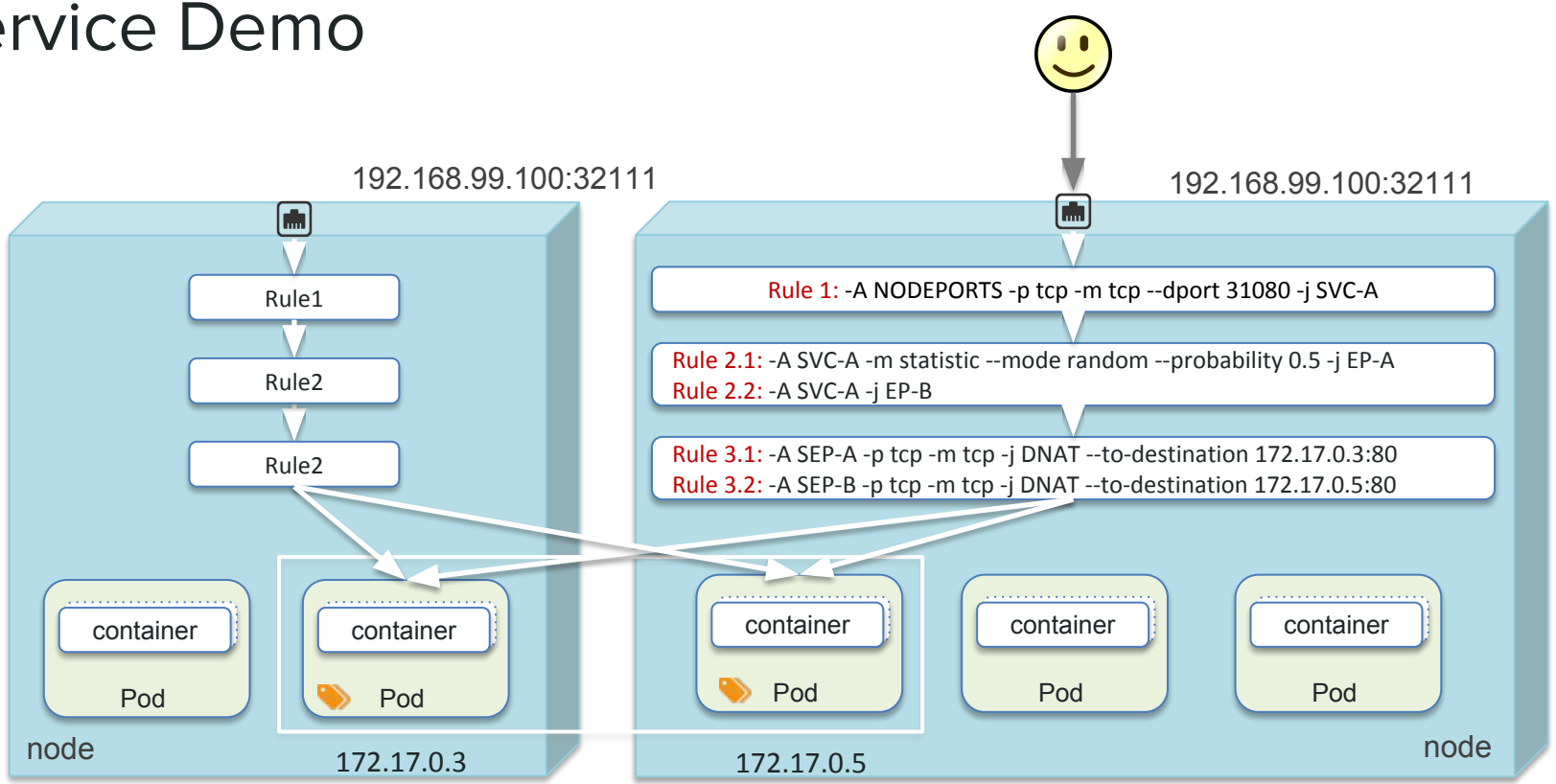
- 为了适应多种需求，kubernetes 为 service 扩展了更多功能
  - external name
  - headless service
  - etc.



Service Demo



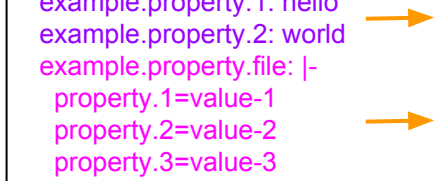
Service Demo



ConfigMap

- 应用镜像和配置分离
- 数据类型为键值对
- Pod 使用 ConfigMap 的三种方式
  - 环境变量
  - 数据卷文件

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data:
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```



ConfigMap Demo（环境变量）

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
  restartPolicy: Never
```

环境变量

环境变量

ConfigMap Demo（数据卷文件）

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh","-c","cat /etc/config/path/to/special-key" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
        - key: special.how
          path: path/to/special-key
  restartPolicy: Never
```

2

key 指定文件路径名
value 作为文件内容

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat /etc/config/special.how" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
  restartPolicy: Never
```

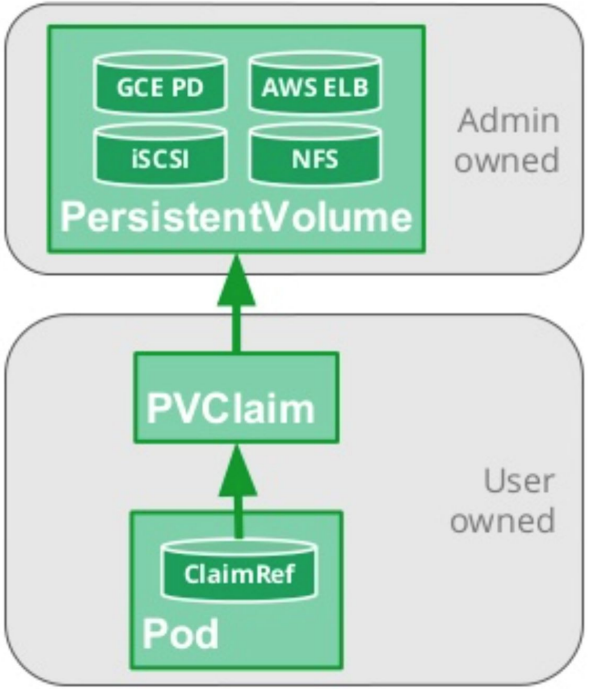
1

key 作为文件名
value 作为文件内容

Persistent Volume

- A higher-level abstraction
  - insulation from any one cloud environment
- Admin provisions PV, user claims them using PVC
  - PV stands for PersistentVolume
  - PVC stands for PersistentVolumeClaim
- Independent lifetime and fate
- Can be used between pods
- Dynamically “scheduled” and managed, like nodes and pods

Kubernetes 并没有提供“存储方案”，kubernetes 只提供了“对接”。kubernetes 管理员需要自己部署存储集群。



Source: Google

Volume and Persistent Volume Demo

Namespace and ResourceQuota

- Namespace
  - 物理集群上的虚拟集群
  - 用户级别的资源限制，用户只需关注 namespace 上的资源情况
  - 默认没有资源限制，需要通过 ResourceQuota 来配额
- ResourceQuota: Namespace 资源配额
  - 计算资源配额
    - cpu, memory
  - k8s 元素数量配额
    - pods, services, replicationcontrollers, resourcequotas, secrets, persistentvolumeclaims

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
spec:
  hard:
    cpu: "20"
    memory: 1Gi
    pods: "10"
    replicationcontrollers: "20"
    resourcequotas: "1"
    services: "5"
```

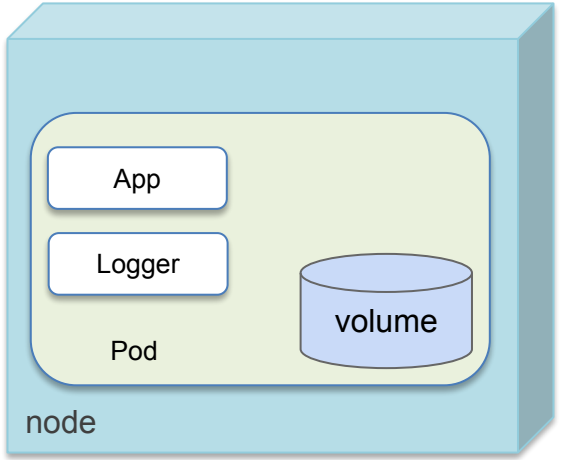
Namespace and ResourceQuota Demo

Kubernetes 架构

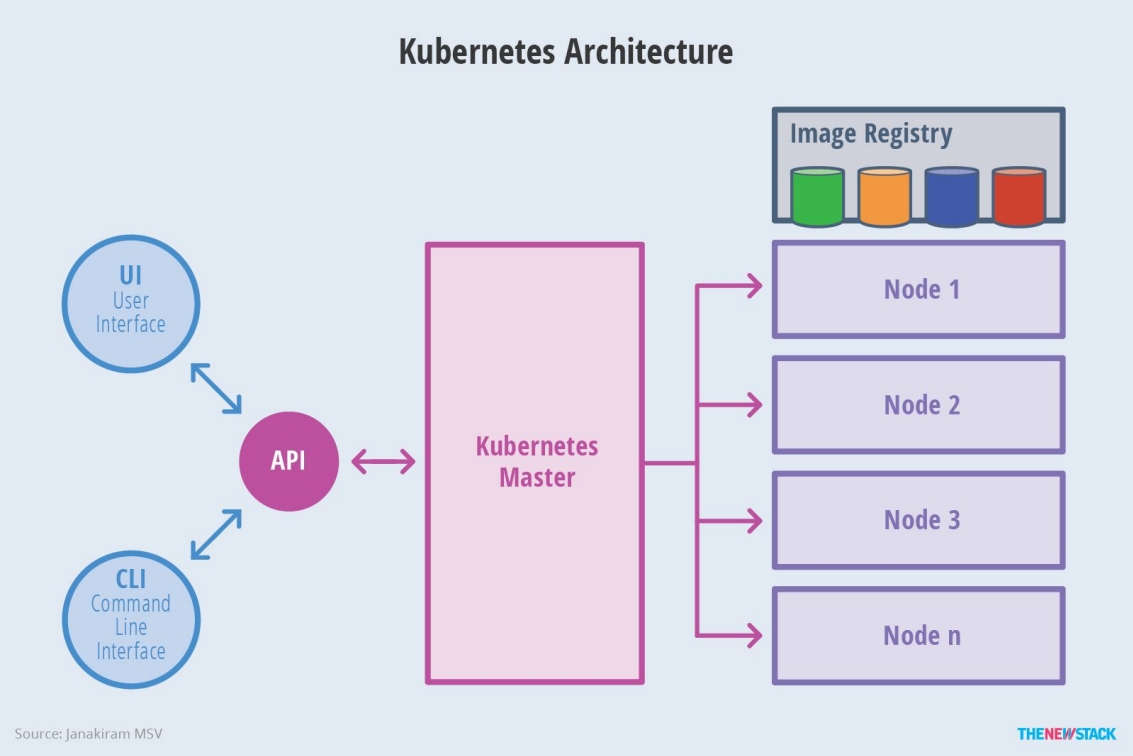
Volume

- ❏ Very similar to Docker's concept
- ❏ Pod scoped storage
- ❏ Share the pod's lifetime & fate
- ❏ Support many types of volume plugins

- emptyDir
- hostPath
- gcePersistentDisk
- awsElasticBlockStore
- nfs
- iscsi
- flocker
- glusterfs
- rbd
- gitRepo
- secret
- persistentVolumeClaim



Kubernetes 架构



Source: TheNewStack

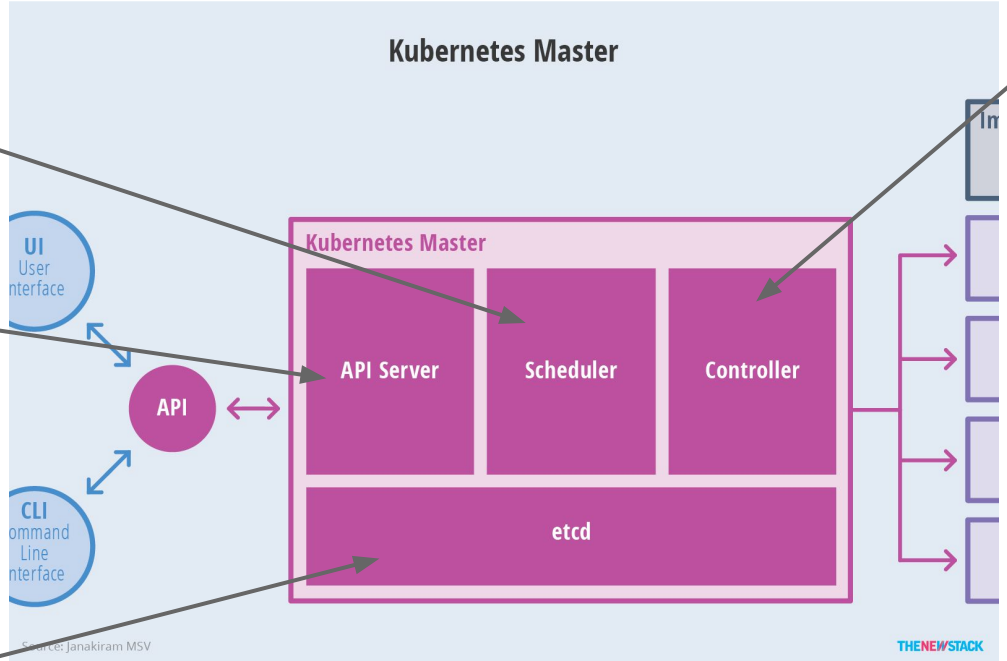
Kubernetes 架构

Controller Manager 由多个 Controller 组成，负责维护应用状态，例如 Deployment Controller 维护 Deployment 部署的 Pod 状态，副本数量等；DaemonSet Controller 维护 DaemonSet Pods 等

Scheduler 是 kubernetes 核心调度单元，负责调度 Pods

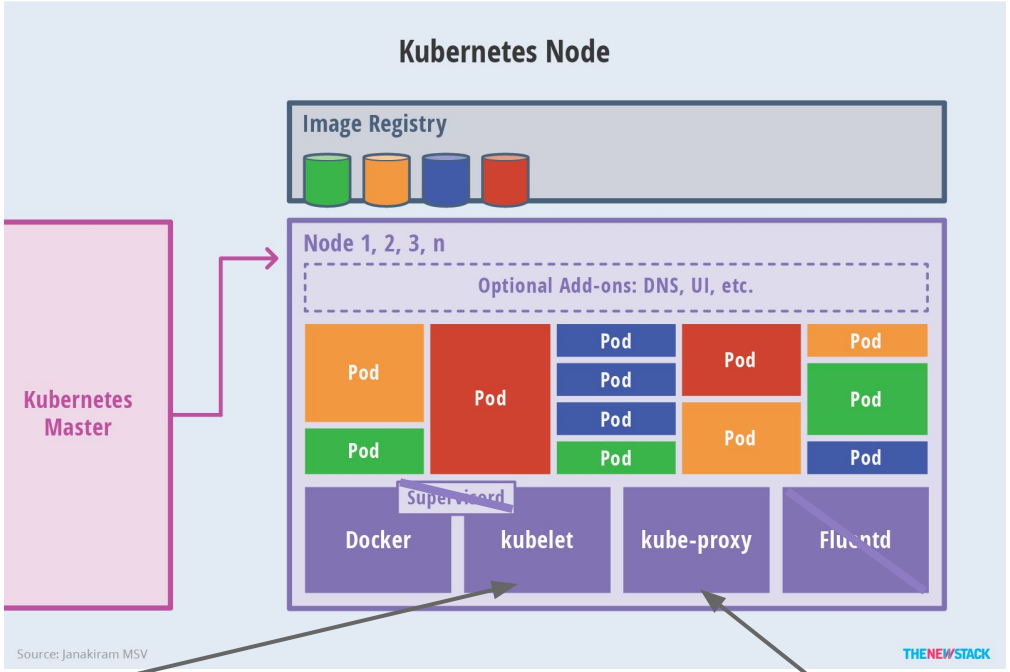
API Server 负责接收所有请求

etcd 是 kubernetes 使用的键值数据库，保存 kubernetes 的所有元数据（例如 Pod Yaml 文件）



Source: TheNewStack

Kubernetes 架构



Kubelet 是 kubernetes 最为核心的 agent，负责管理 Pod，Volume 等与节点相关资源

Proxy 主要负责维护 Service 虚拟 IP

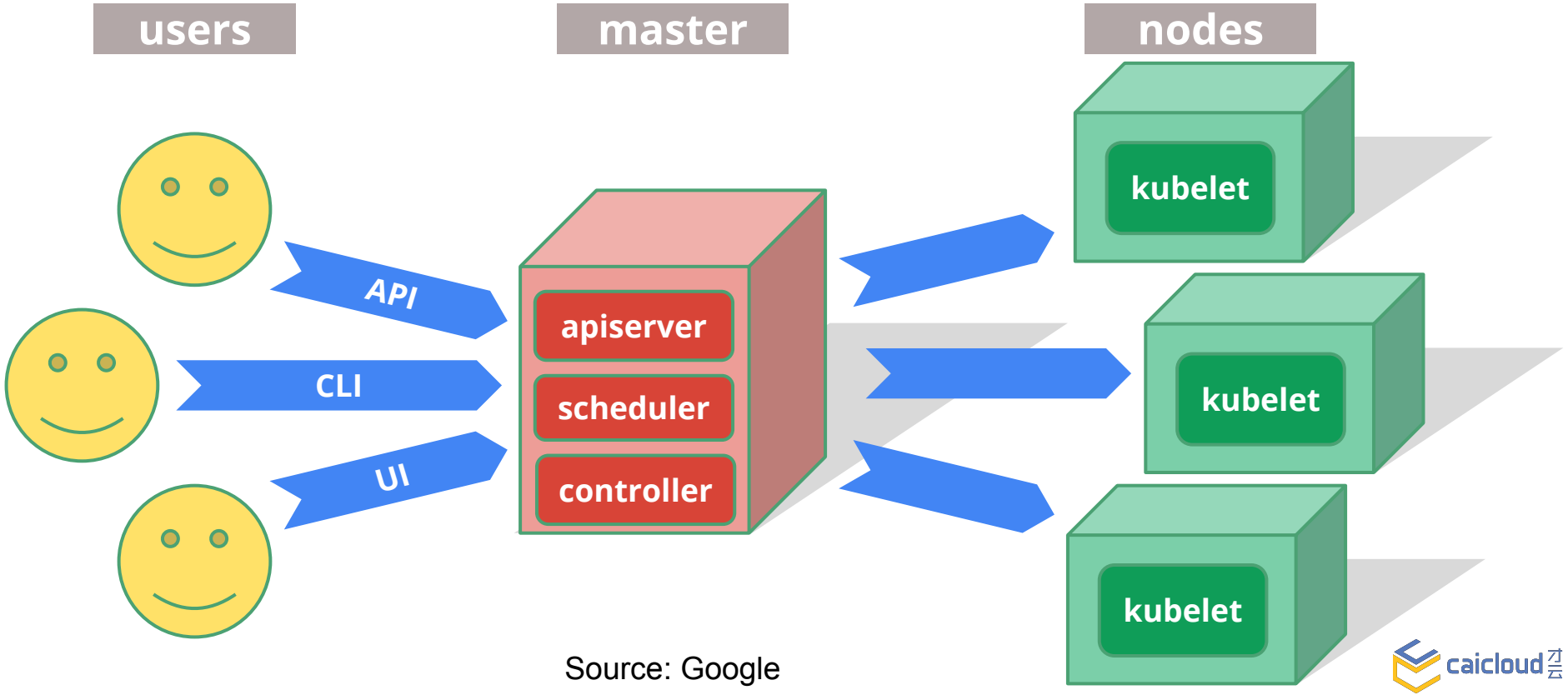
Source: TheNewStack

Kubernetes 架构解析

- K8S 集群组成
  - 分布式存储(Etcd)
  - 控制节点(Master)
  - 工作节点(Node)
- 只有 apiserver 与存储通信
  - 用户直接访问 apiserver
  - 内部进程，包括 kubelet, controller 均通过 apiserver 访问存储
  - 出于安全考虑
- 配置管理操作声明式而非命令式
  - 由 controller manager 来负责维护声明的内容
- 集群状态维护采用 list-watch 方式
  - watch 增量更新：快速低延时
  - list 全量数据：确保数据正确性

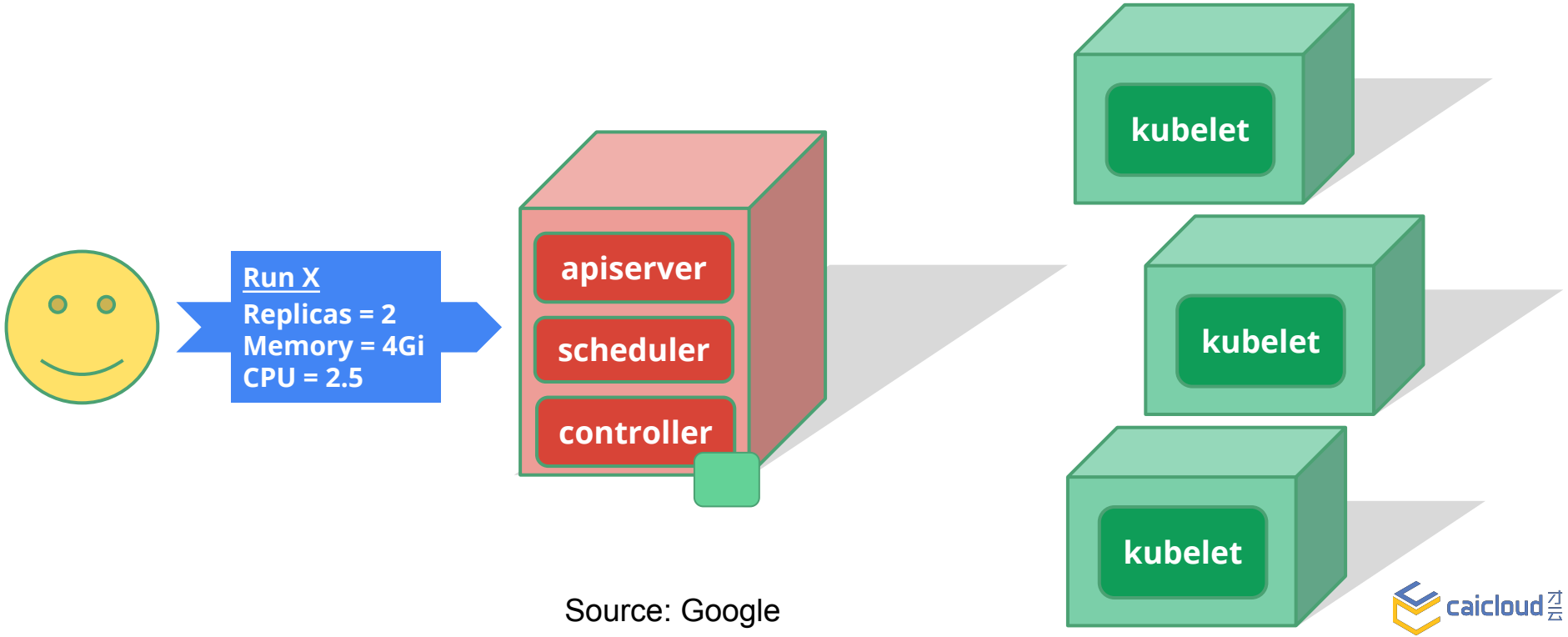
Kubernetes 工作流程

Kubernetes 工作流程



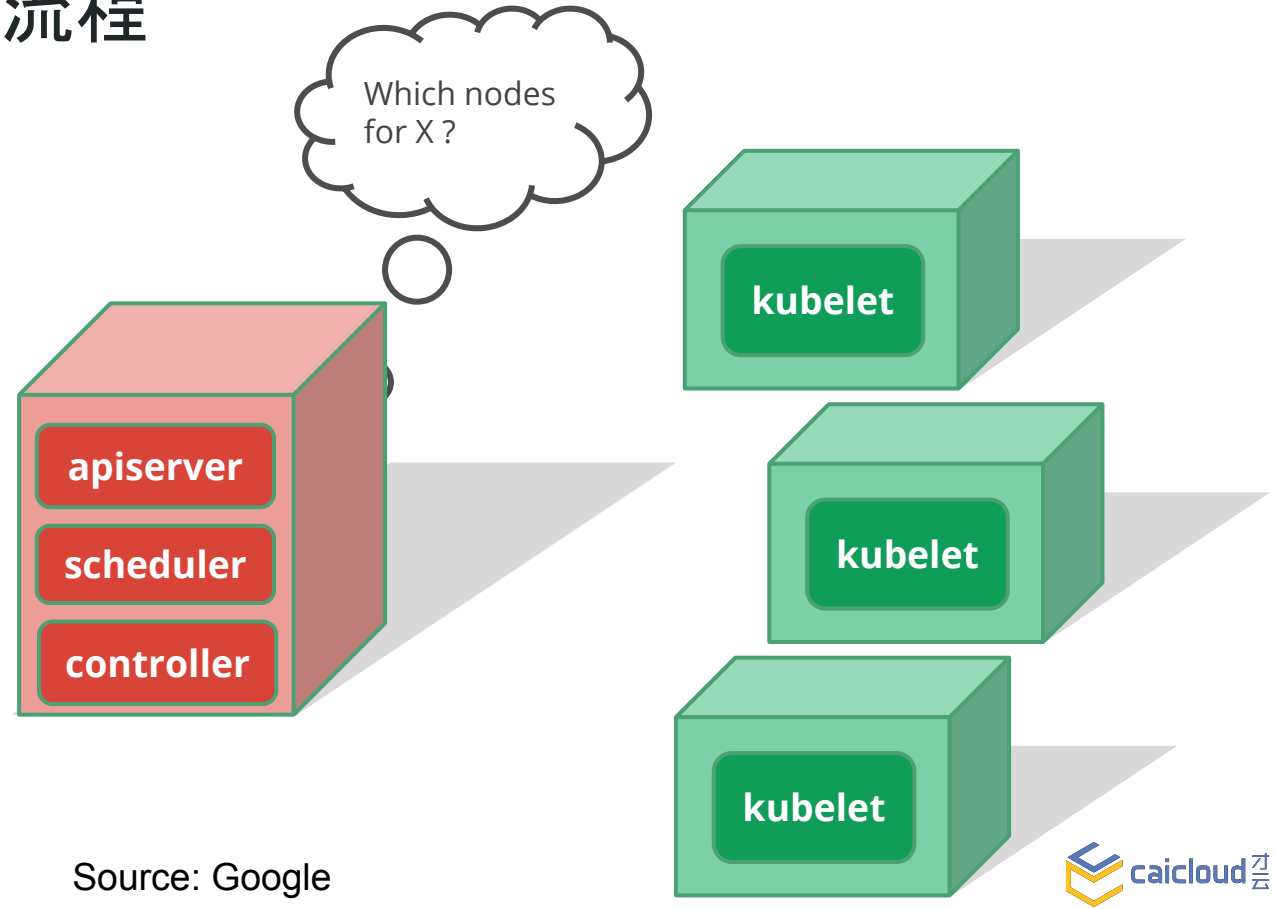
Source: Google

Kubernetes 工作流程



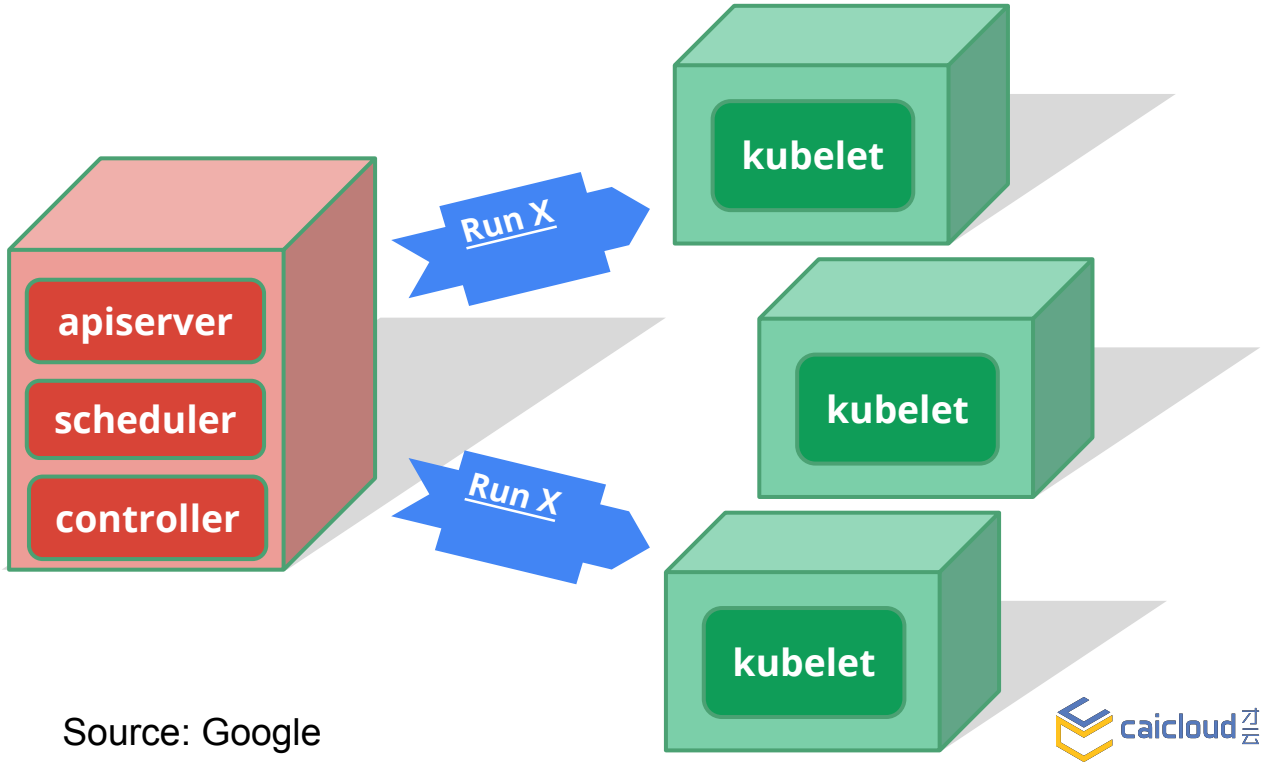
Source: Google

Kubernetes 工作流程



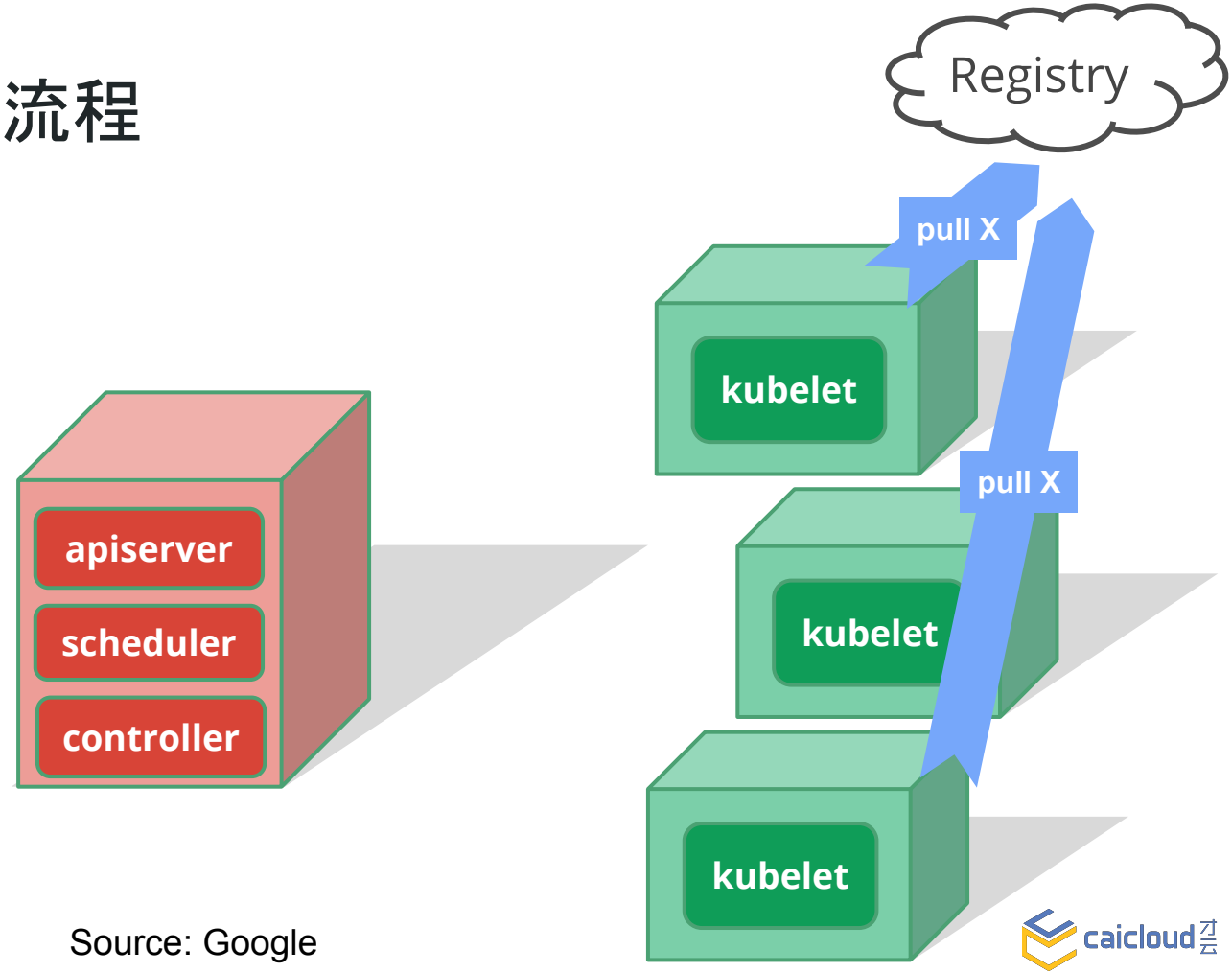
Source: Google

Kubernetes 工作流程



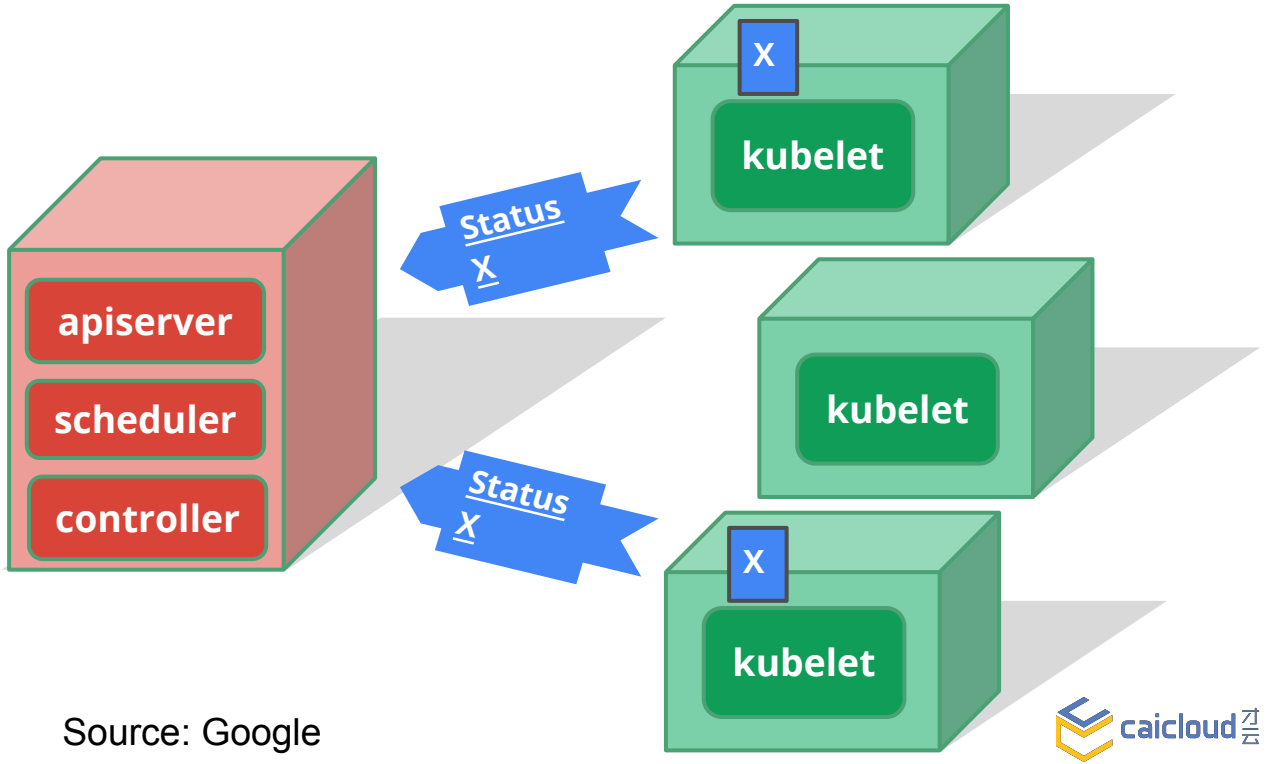
Source: Google

Kubernetes 工作流程



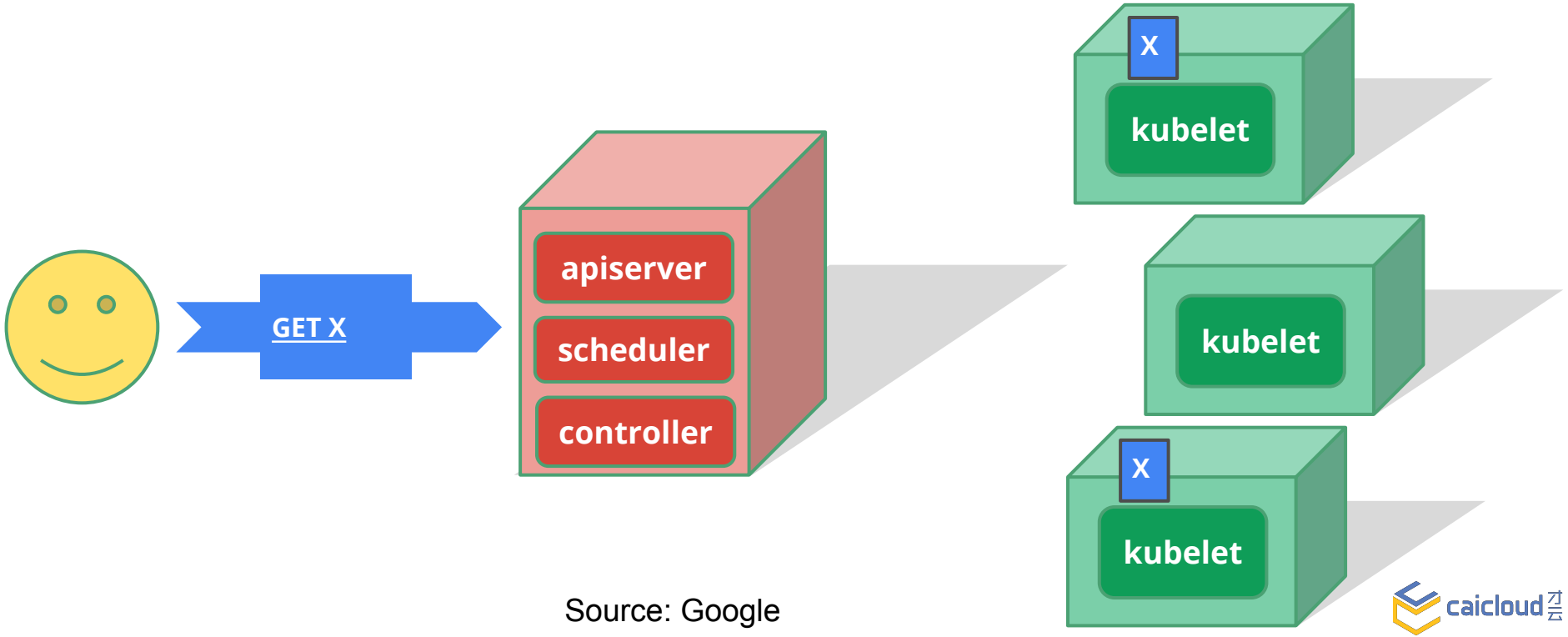
Source: Google

Kubernetes 工作流程



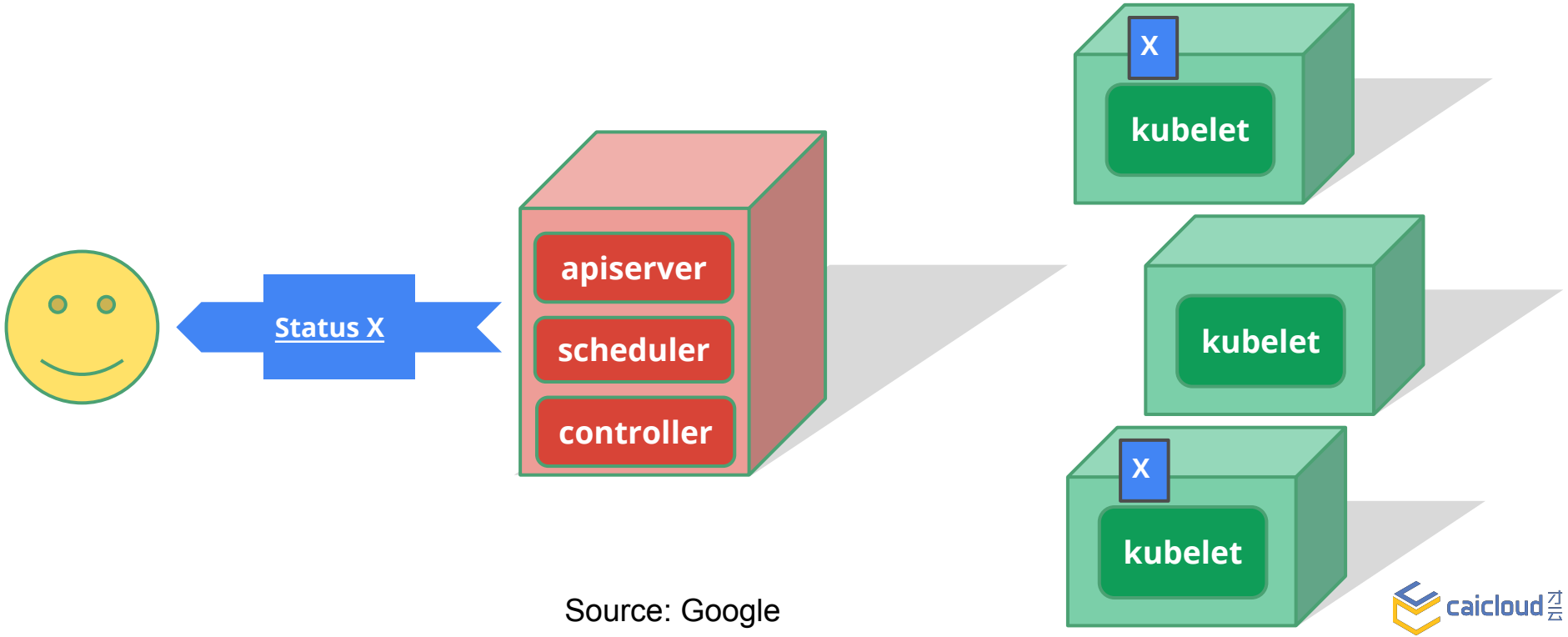
Source: Google

Kubernetes 工作流程



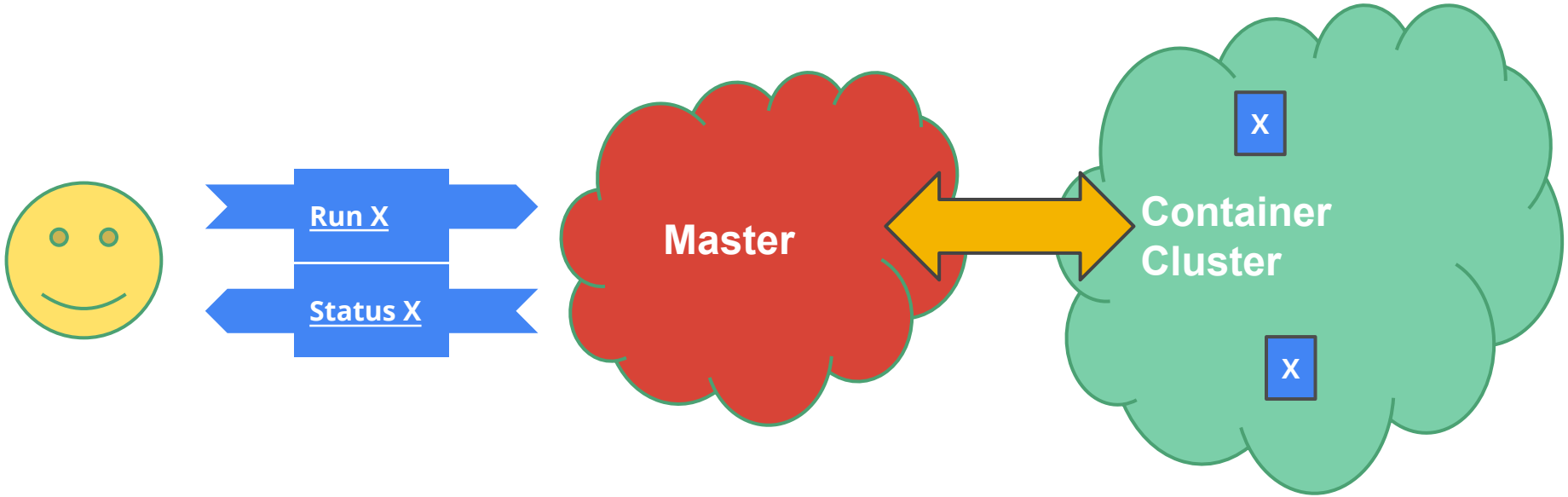
Source: Google

Kubernetes 工作流程



Source: Google

Kubernetes 工作流程 - 用户角度



Source: Google

Putting it Together

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

/etc/resolv.conf

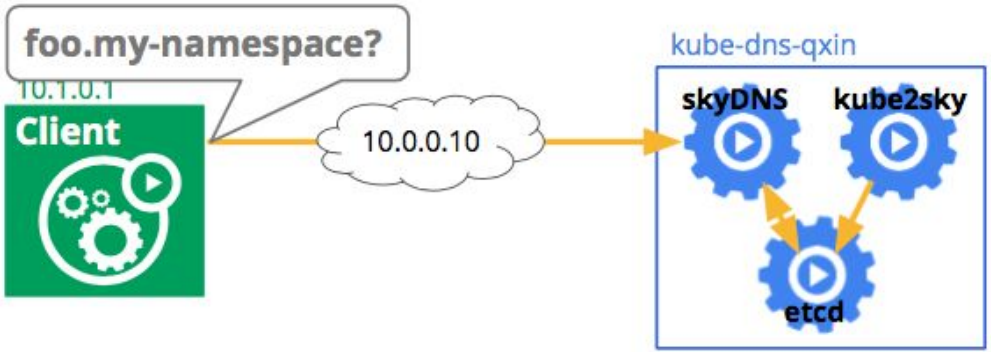
```
nameserver 10.0.0.10
...
```

Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

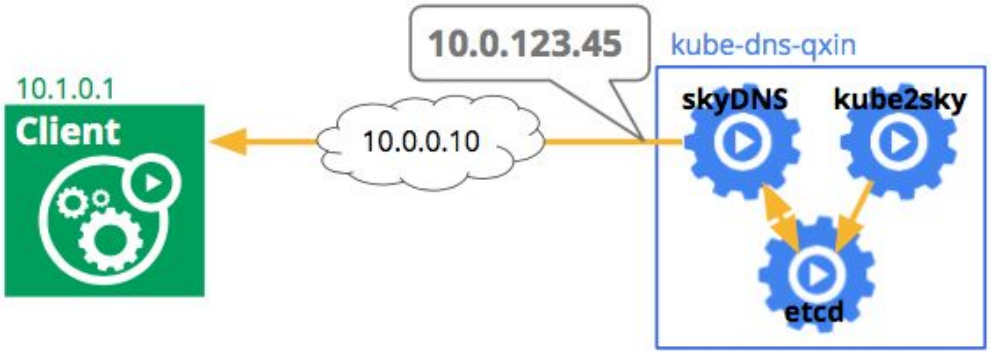


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

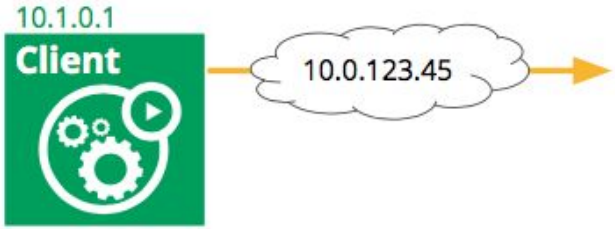


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

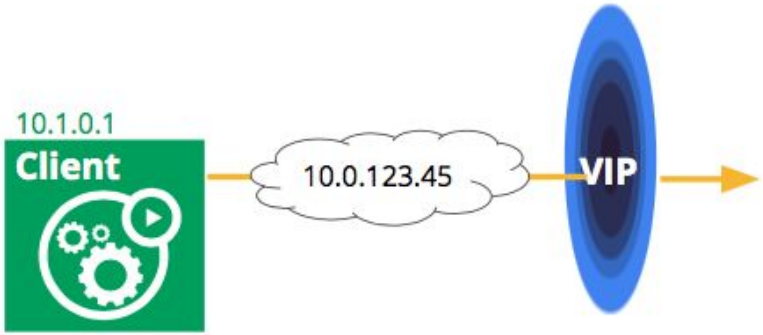


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

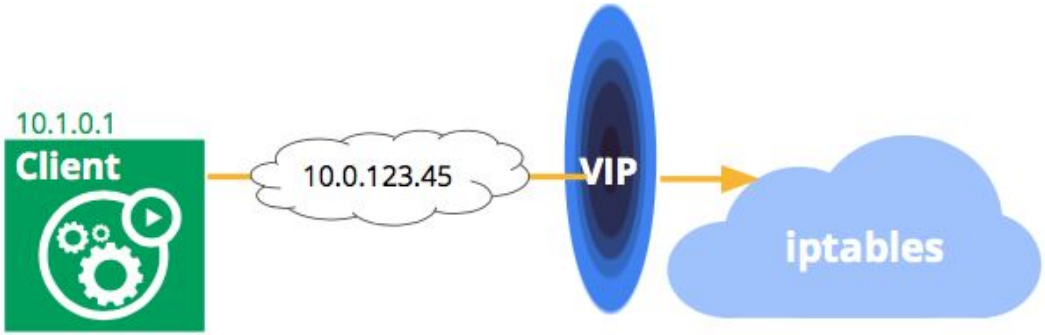


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

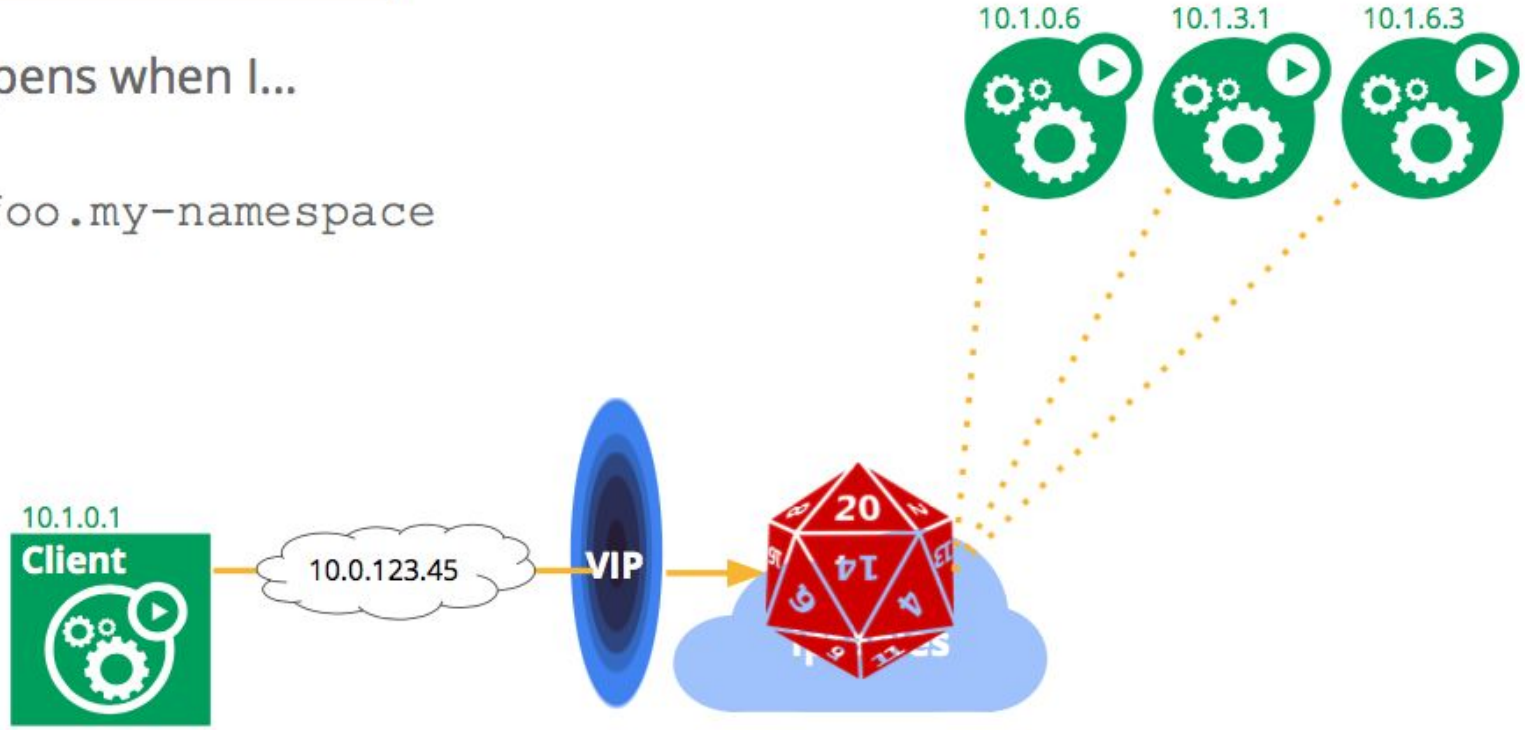


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

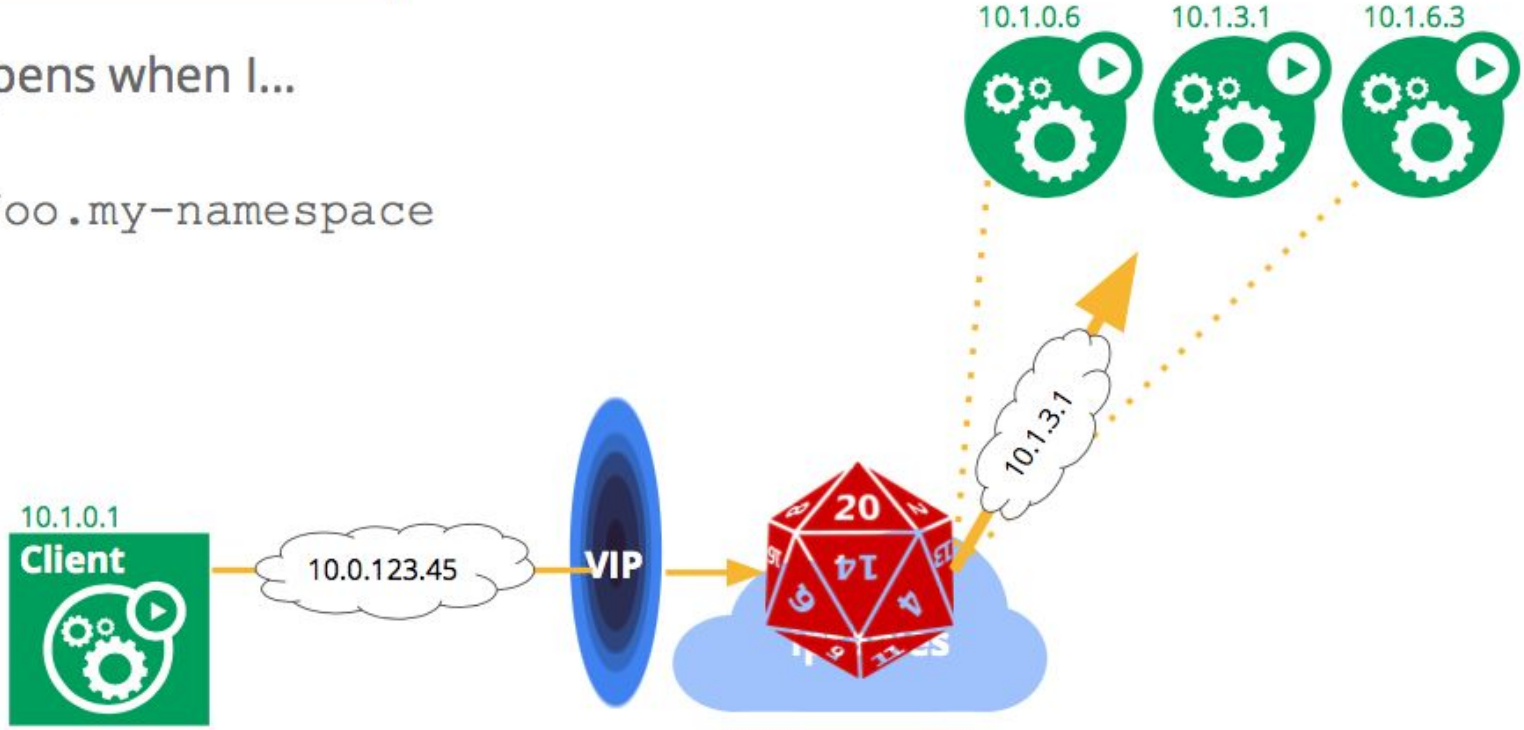


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

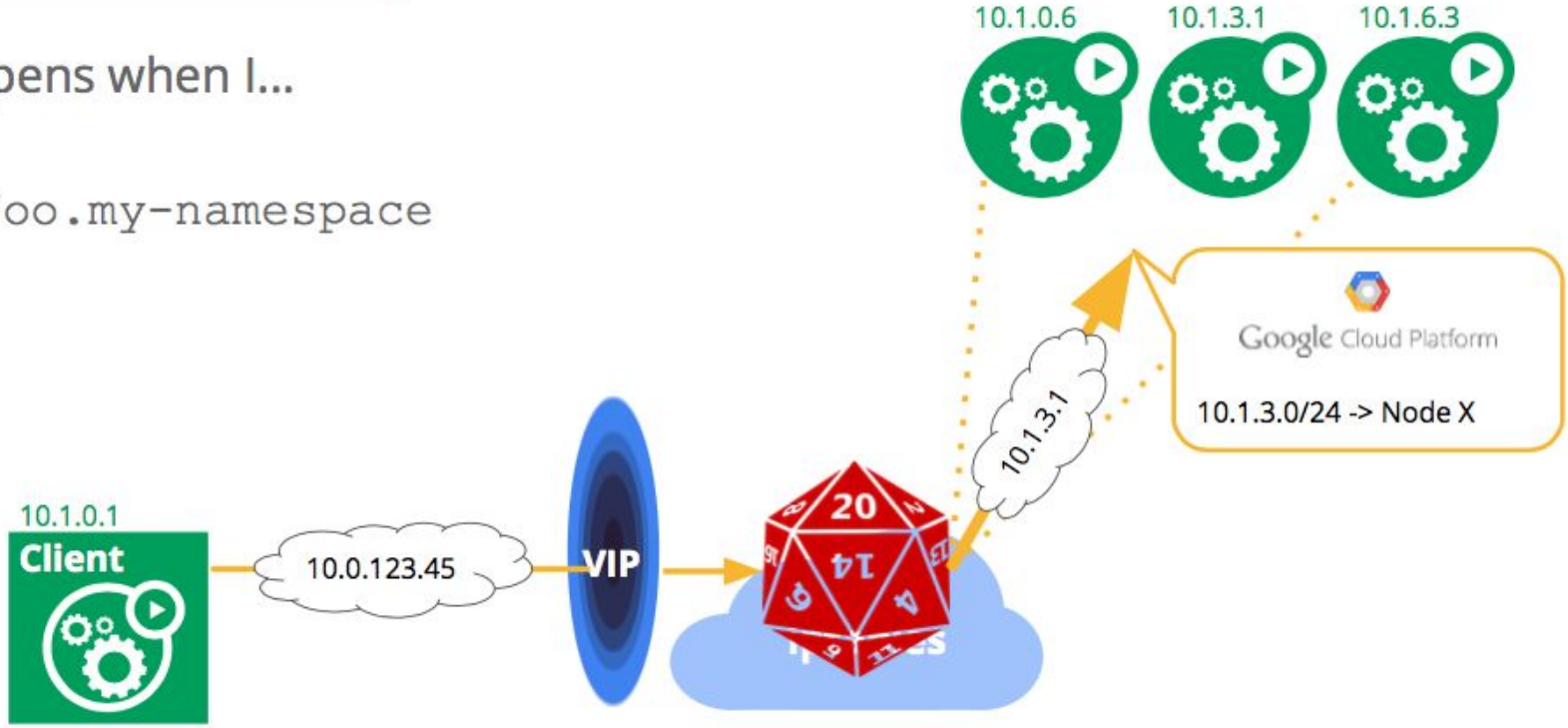


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

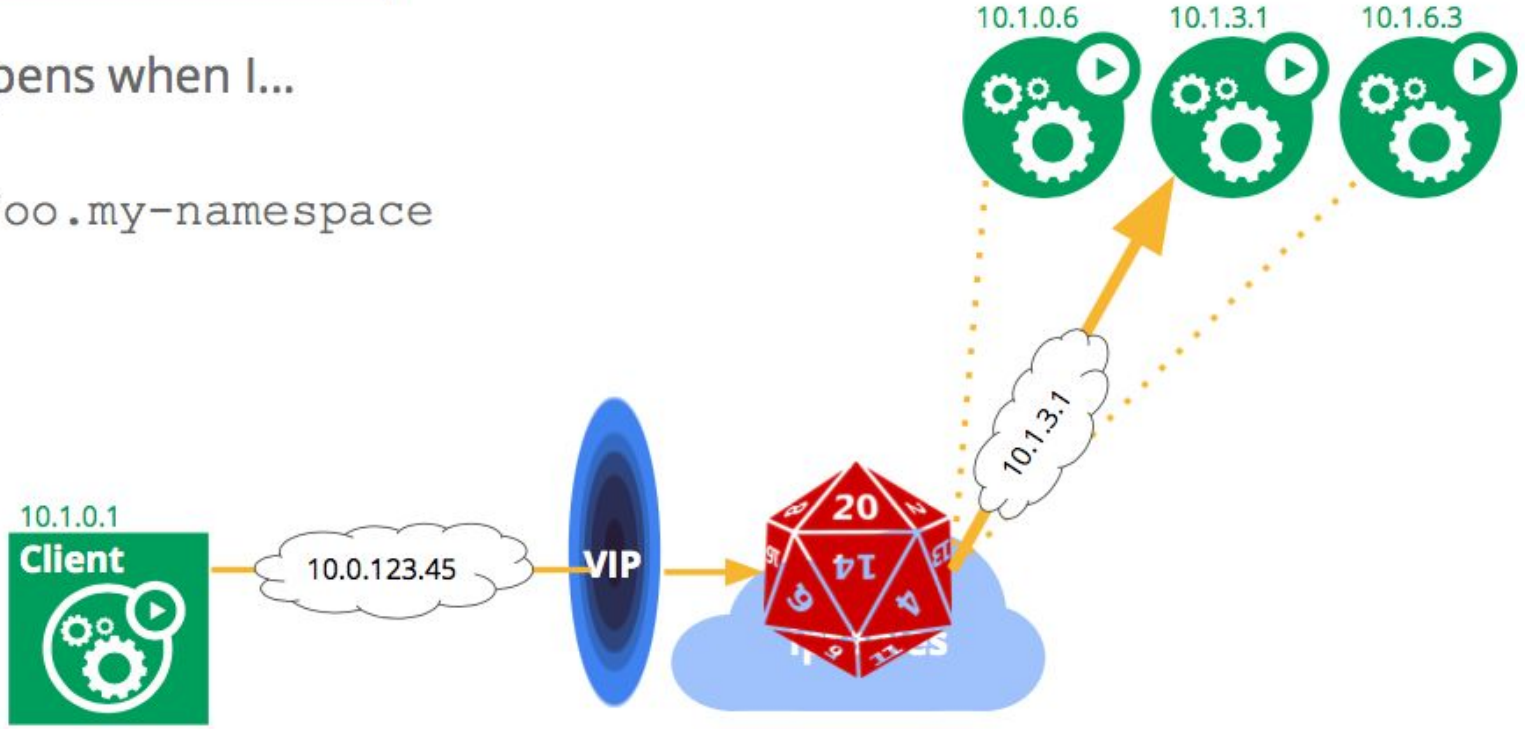


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

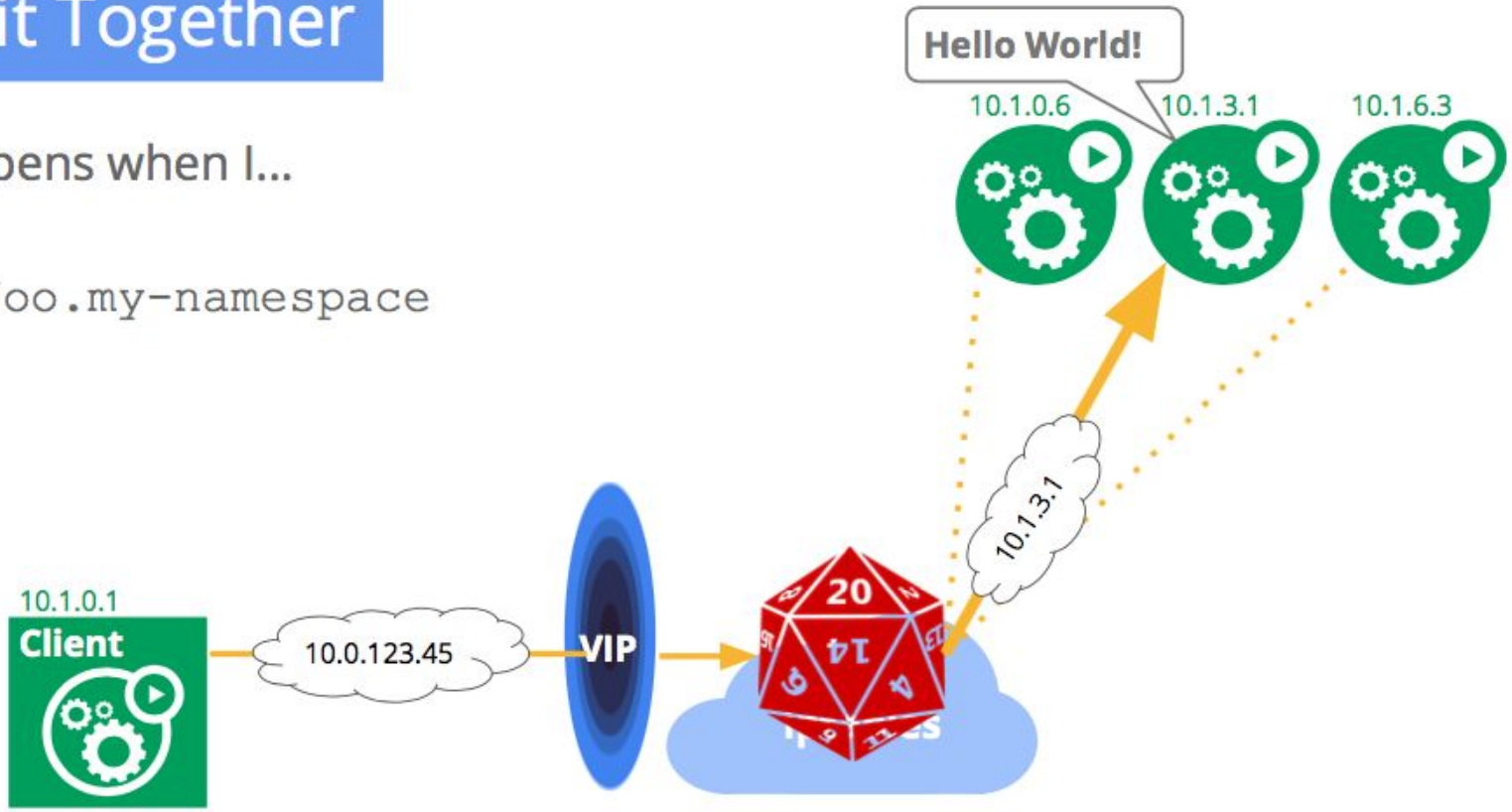


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

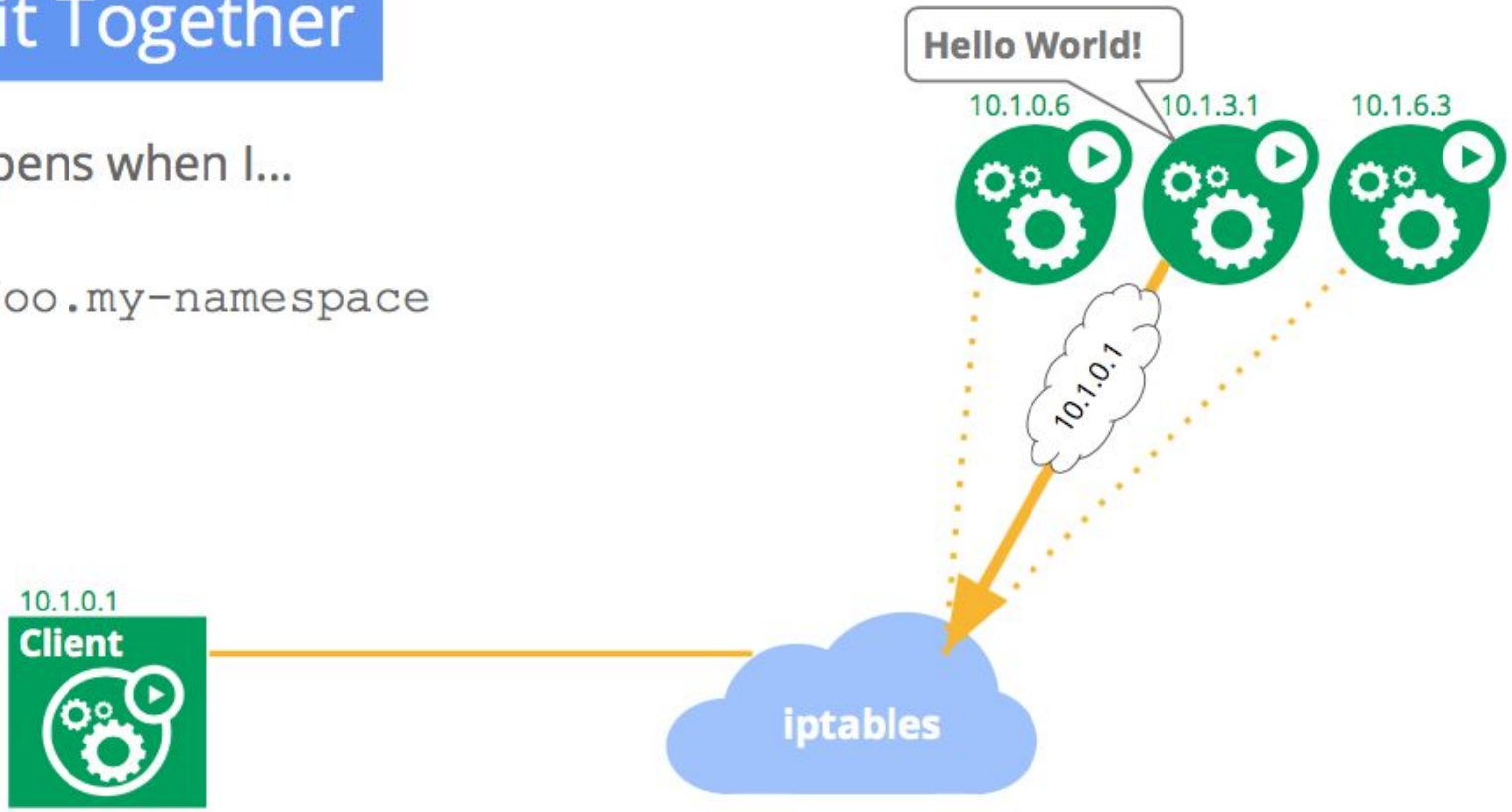


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

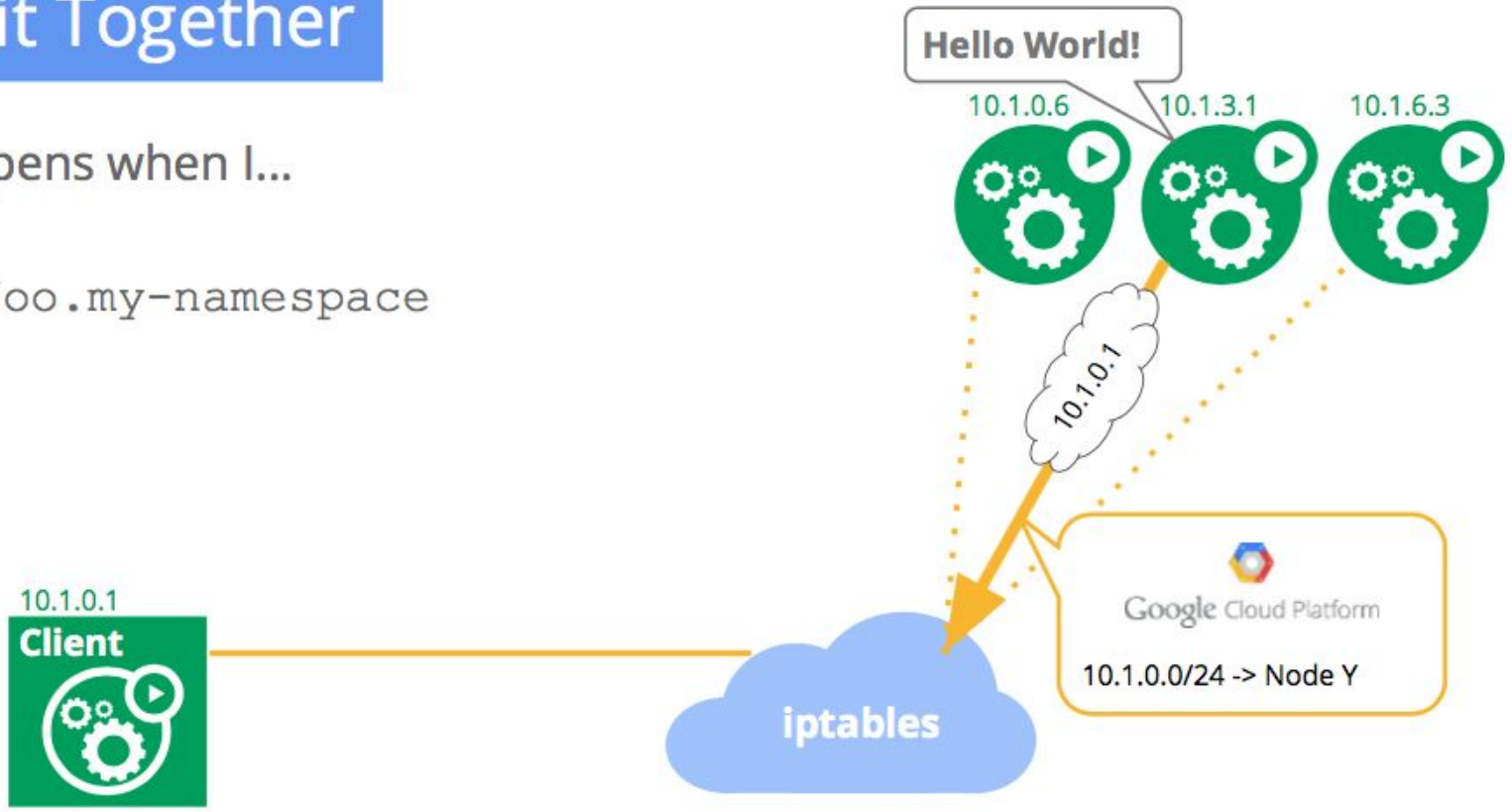


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

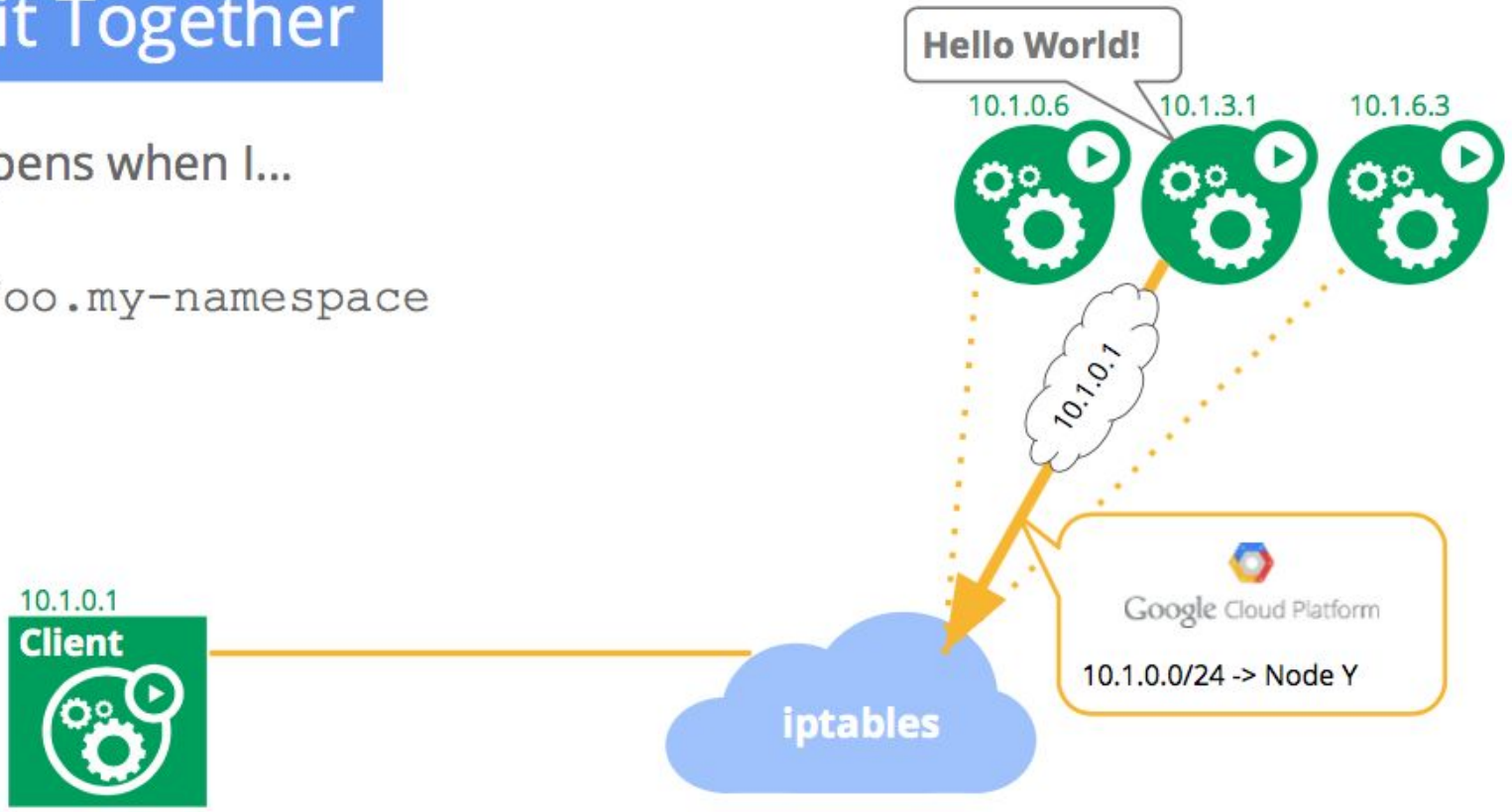


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

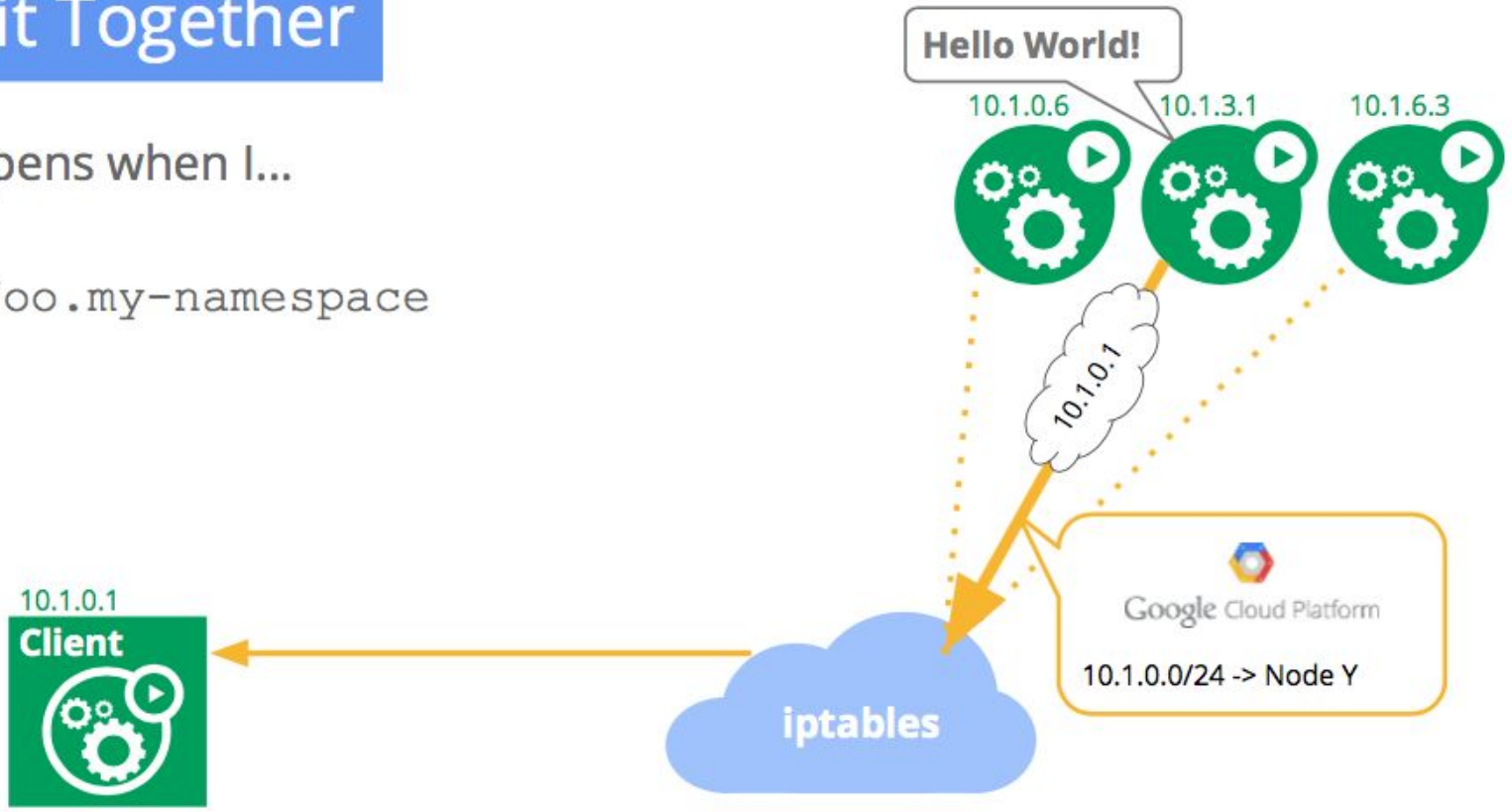


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace
```

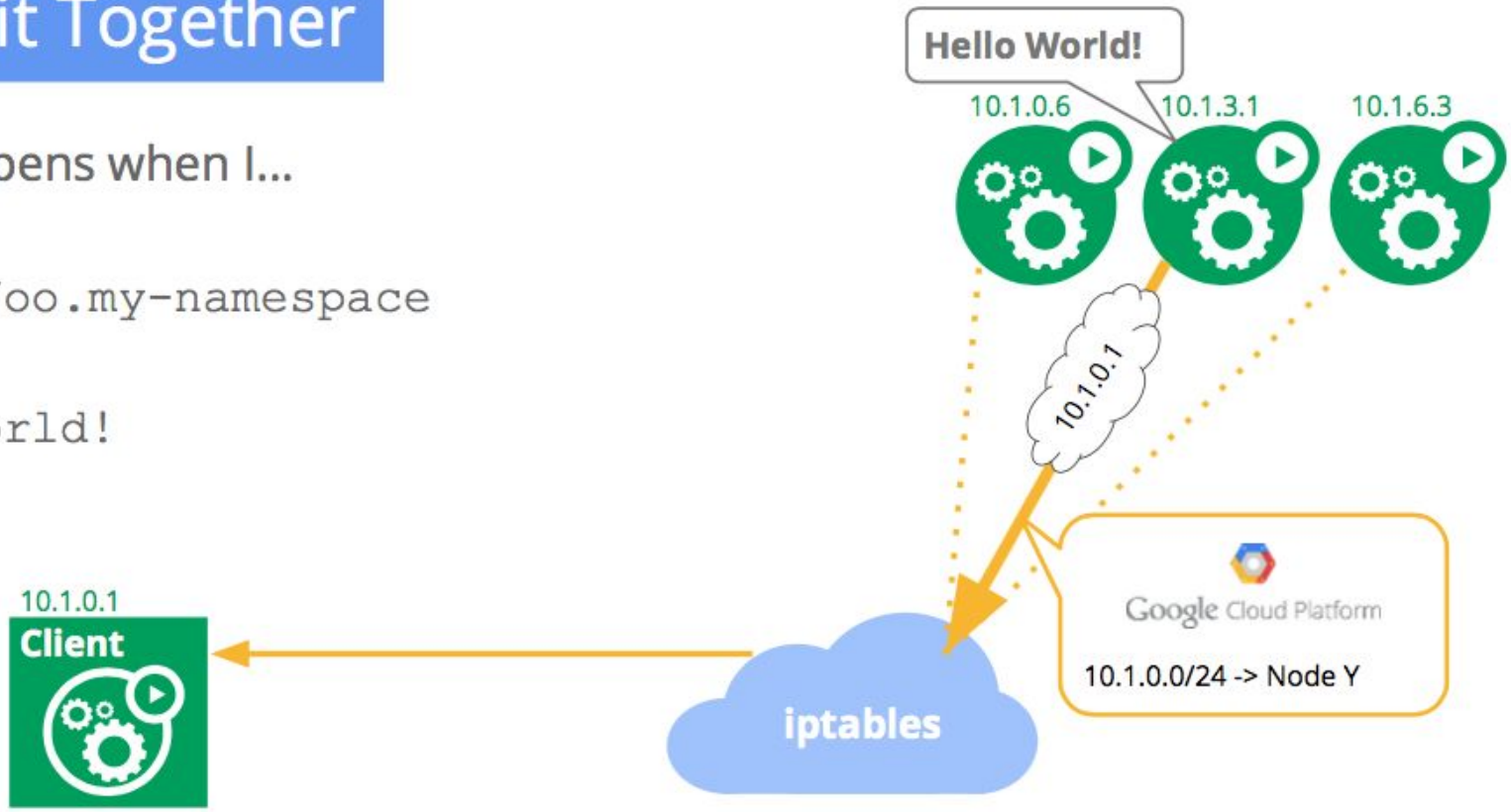


Source: Google

Putting it Together

What happens when I...

```
$ curl foo.my-namespace

Hello World!
```



Source: Google

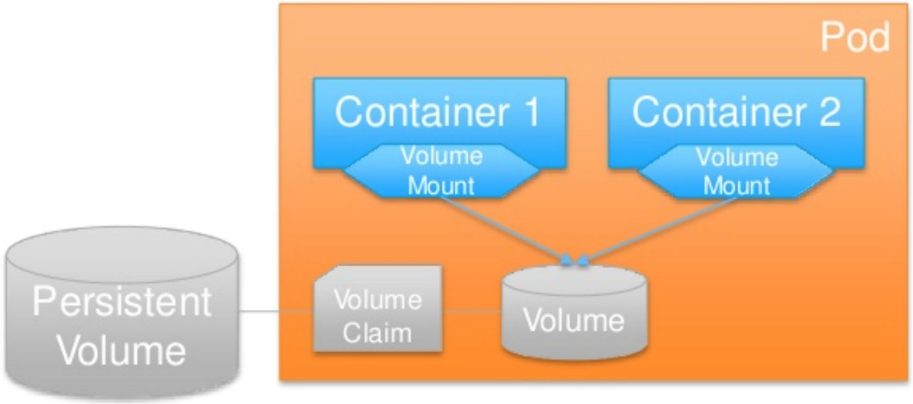
谢谢大家！

Kubernetes Volumes

Volumes

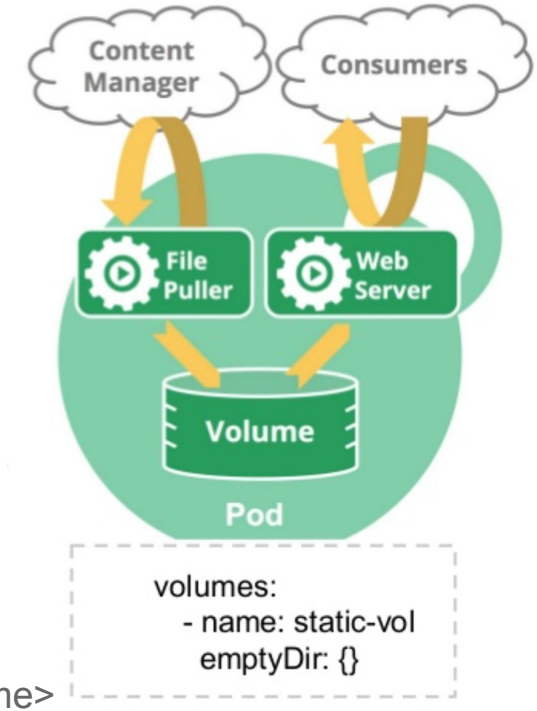
- ❏ Very similar to Docker's concept
- ❏ Pod scoped storage
- ❏ Share the pod's lifetime & fate
- ❏ Support many types of volume plugins

- emptyDir
- hostPath
- gcePersistentDisk
- awsElasticBlockStore
- nfs
- iscsi
- flocker
- glusterfs
- rbd
- gitRepo
- secret
- persistentVolumeClaim



emptyDir

- emptyDir is a temporary directory that shares a pod lifetime.
  - storage provider = local host
  - files will be erased on pod deletion
  - mounted by containers inside the pod

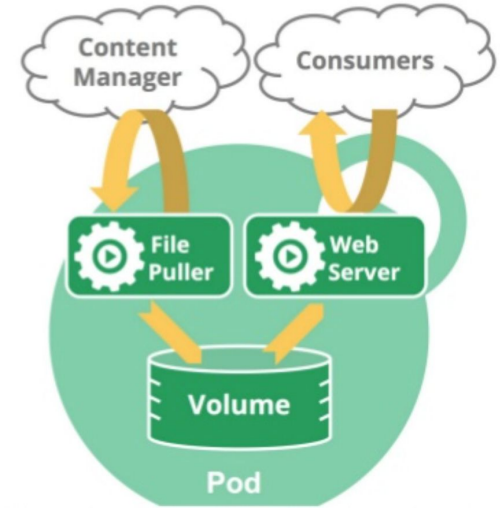


emptyDir path on node:

/var/lib/kubelet/pods/<id>/volumes/kubernetes.io~empty-dir/<volume_name>

hostPath

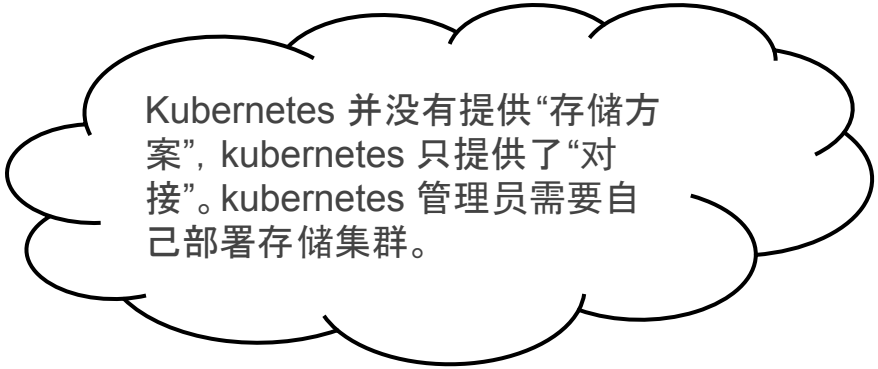
- hostPath is a bare host directory volume
  - acts as data volume in container
  - containers can RW files on localhost
  - there is no control on quota

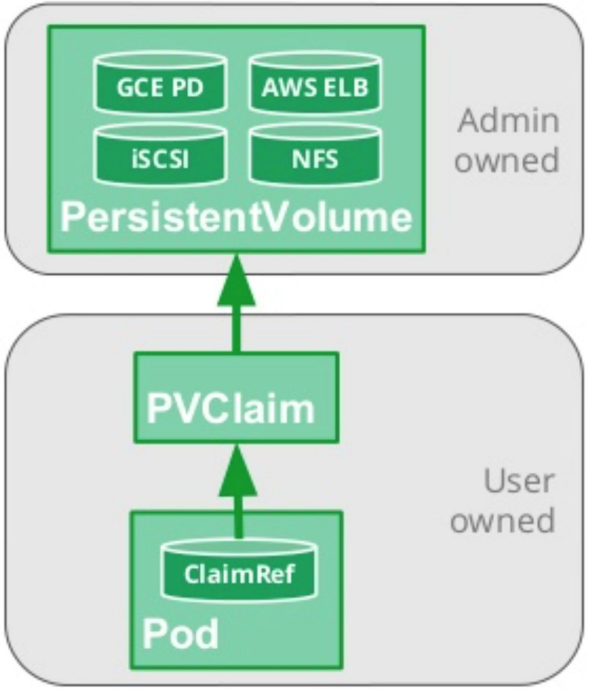



```
Volumes:
  - name: static-vol
    hostPath:
      path: /target/path/on/host
```

Persistent Volumes

- A higher-level abstraction
  - insulation from any one cloud environment
- Admin provisions PV, user claims them using PVC
  - PV stands for PersistentVolume
  - PVC stands for PersistentVolumeClaim
- Independent lifetime and fate
- Can be used between pods
- Dynamically “scheduled” and managed, like nodes and pods





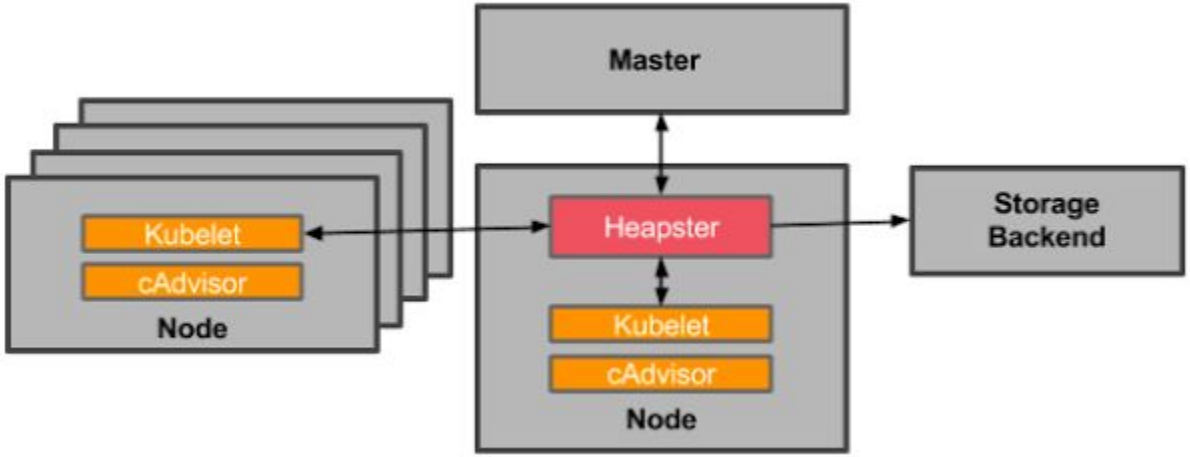
Flex Volume

- Allow custom volume plugin
- A static binary exec'd by kubelet
- Alpha Interface
  - <executable> init
  - <executable> attach <json>
  - <executable> detach <device>
  - <executable> mount <target mount dir> <mount device> <json options>
  - <executable> unmount <mount dir>
- Discovery location
  - /usr/libexec/kubernetes/kubelet-plugins/volume/exec/<vendor~driver>/<driver>

监控系统

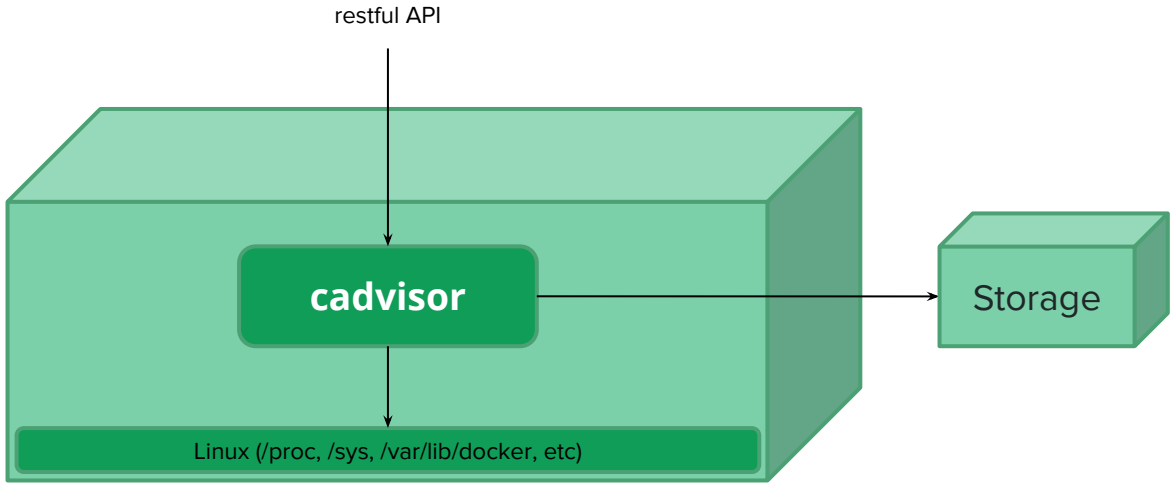
Kubernetes 默认监控系统

Kubernetes 默认的监控系统为 cadvisor + heapster + influxdb + grafana，以 addon 的形式运行在 kubernetes 的 kube-system namespace 下。

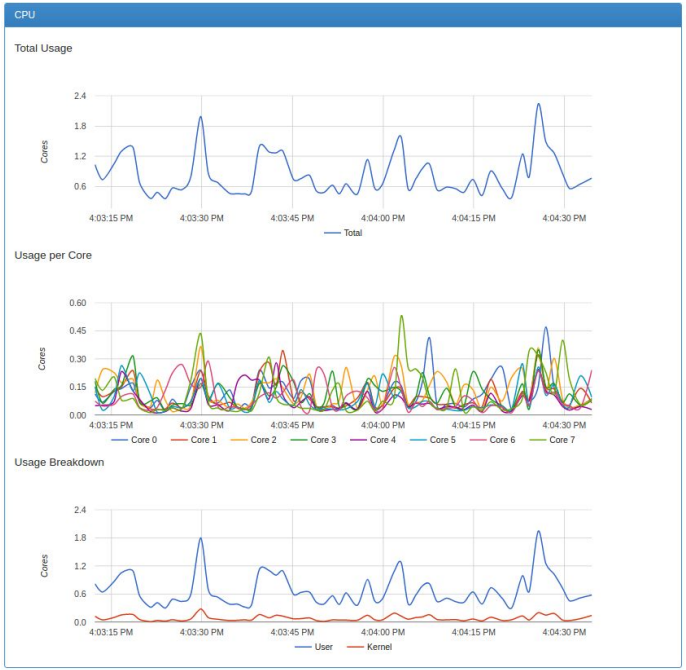


Cadvisor (https://github.com/google/cadvisor)

- 通过读取 procfs, sysfs 以及和 container runtime 相关的目录 (e.g. /var/lib/docker) 来获取监控数据和容器相关信息
- 对外为一些采集组件提供了查询的 restful 的接口
- cadvisor 也可以直接指定一个外部存储，将监控数据直接发送到外部存储中
- cadvisor 提供了一个 UI，对监控数据做了一个简单的展示
- kubernetes 集群中，每个节点上的 kubelet 集成了 cadvisor，并且 kubelet 会通过主机的 10255 端口将这些数据暴露出去

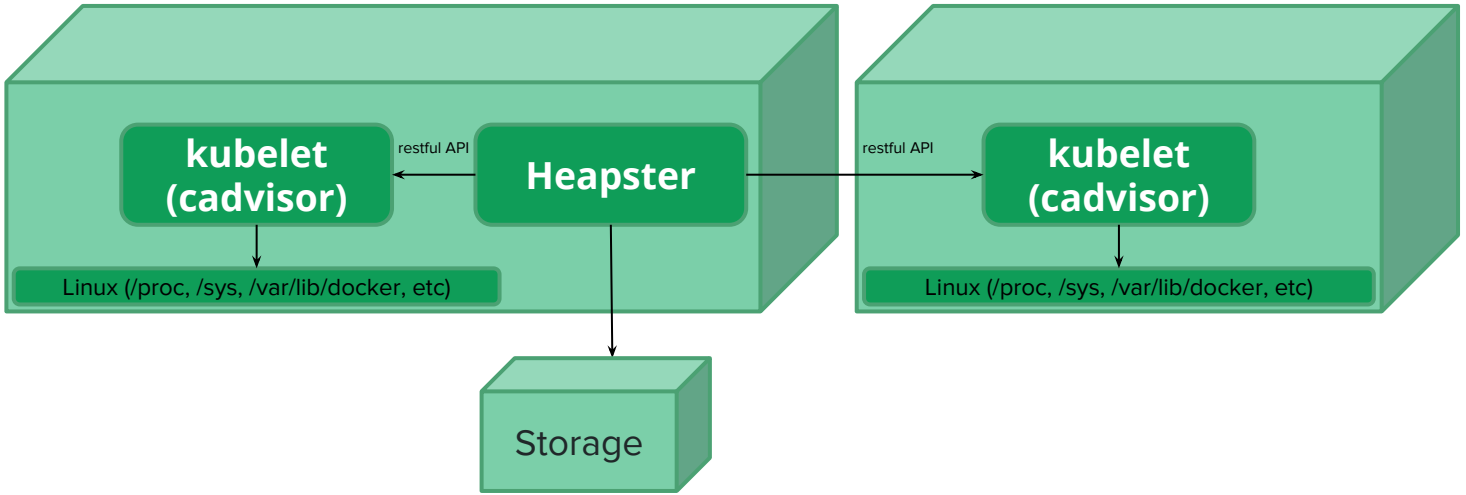


Cadvisor cont’d



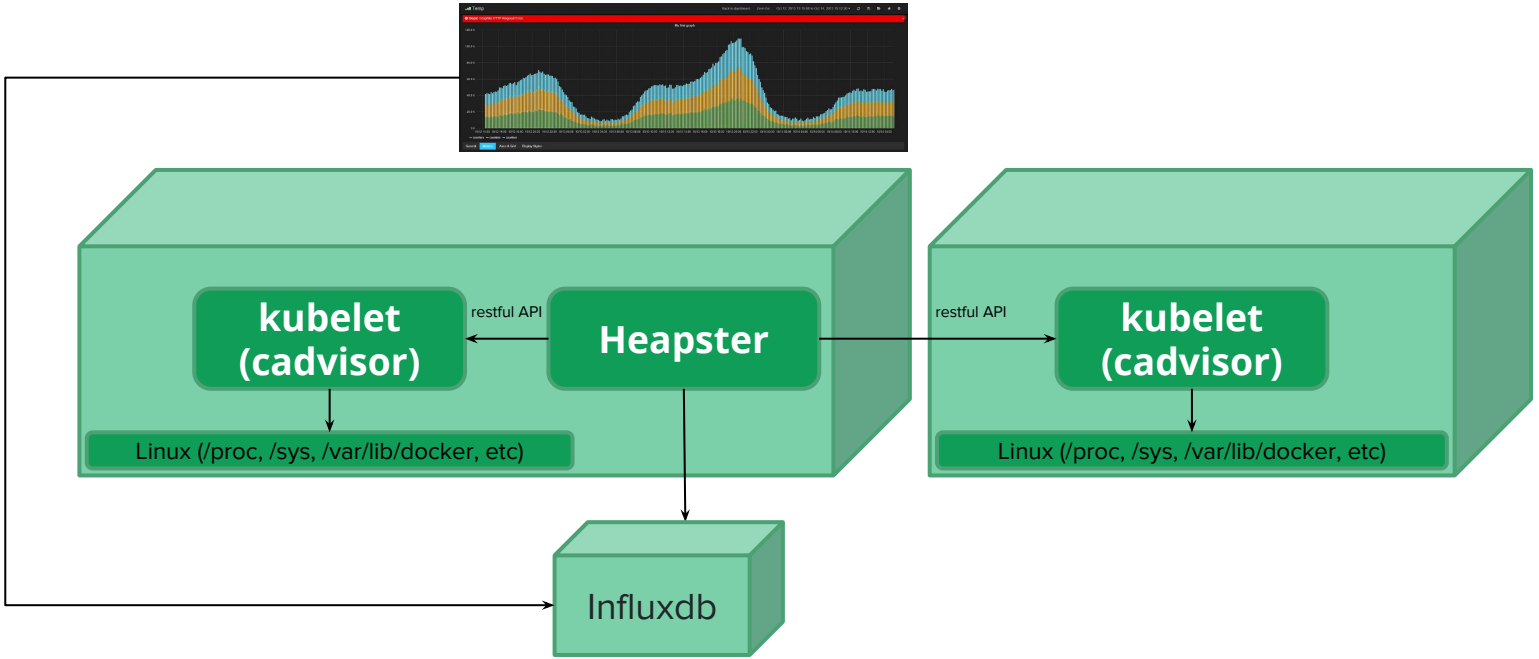
Heapster (https://github.com/kubernetes/heapster)

- 通过 kubelet 暴露的 http 服务获取到每个节点上的监控数据
- 对原始的监控数据和 kubernetes 的系统 event 进行处理，聚合
- 把处理过的监控数据发送给后端存储，目前 heapster 支持 influxdb, hawkular, elasticsearch, opentsdb, kafka 等后端存储
- 同时 heapster 也提供了一个更高层的 http 服务，返回针对 kubernetes 进行过聚合之后的数据 (e.g. 一个 namespace 下面所有的 pod 的内存使用总和)



Grafana + Influxdb

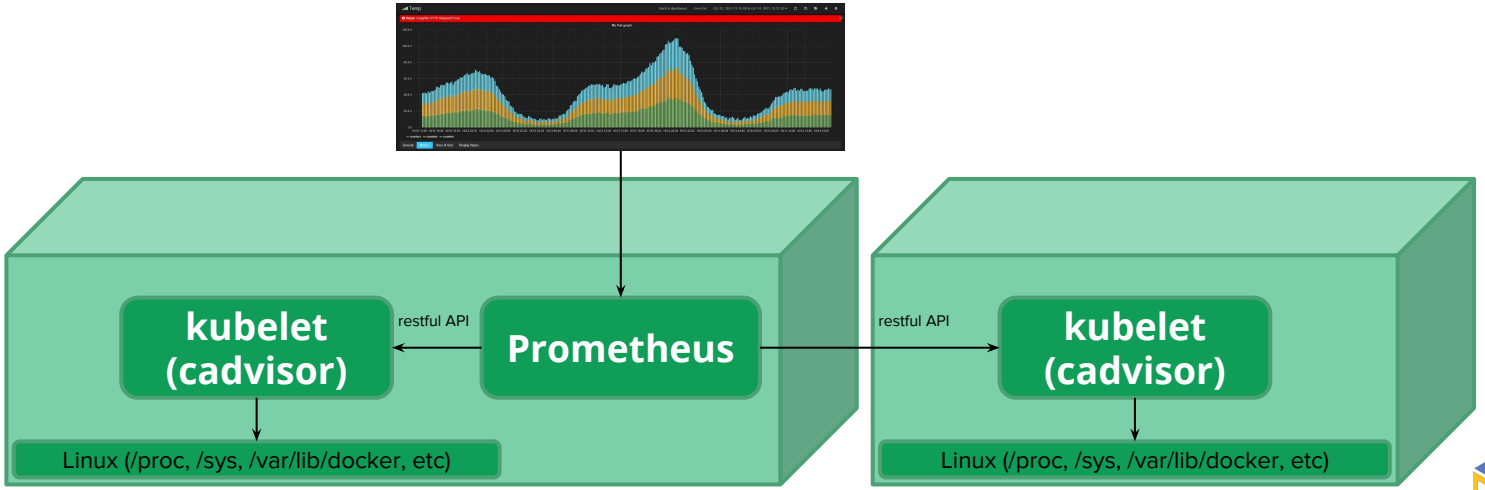
- influxdb 为时序数据库，对外提供了一个简单易用的 API 方便读写数据
- Grafana 是一个实现数据可视化的可配置的 dashboard
- Heapster 将聚合过的数据发送到 Influxdb 中之后，由 Grafana 来完成数据呈现



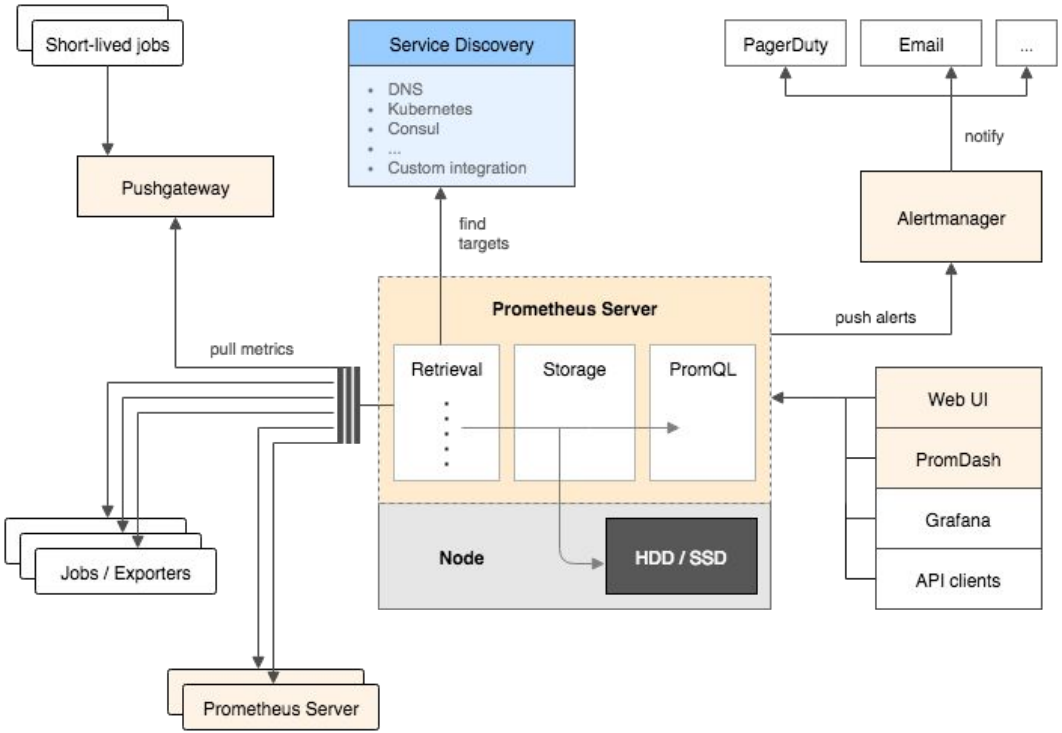
Prometheus (https://prometheus.io)

除了 kubernetes 默认的监控系统，另外一个常见的监控部署系统是 prometheus，工作原理类似于 heapster + influxdb。

- 更加丰富的查询语言
- 部署简单，自带存储和查询界面
- 基于拉模型，支持横向扩展
- 配合 alertmanager 满足报警需求



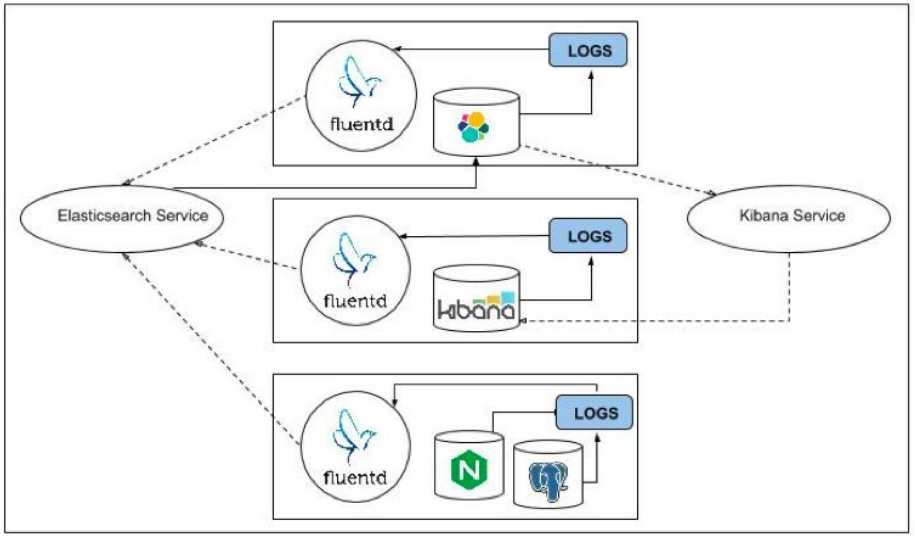
Prometheus (https://prometheus.io)



日志系统

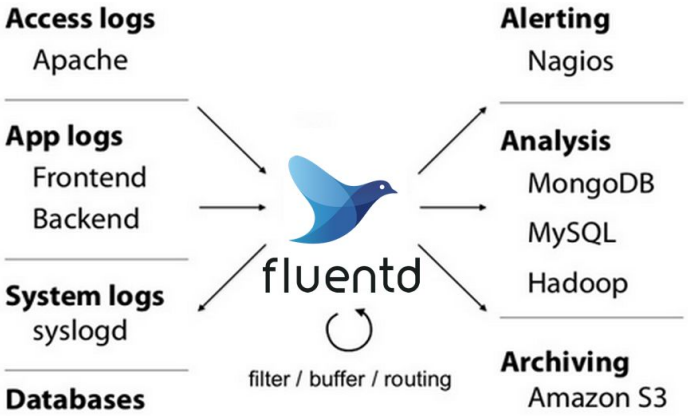
kubernetes 默认日志系统

Kubernetes 默认的日志系统为 Fluentd + Elasticsearch + Kibana，与监控系统类似，也是以 addon 的形式运行在 kube-system namespace 下。



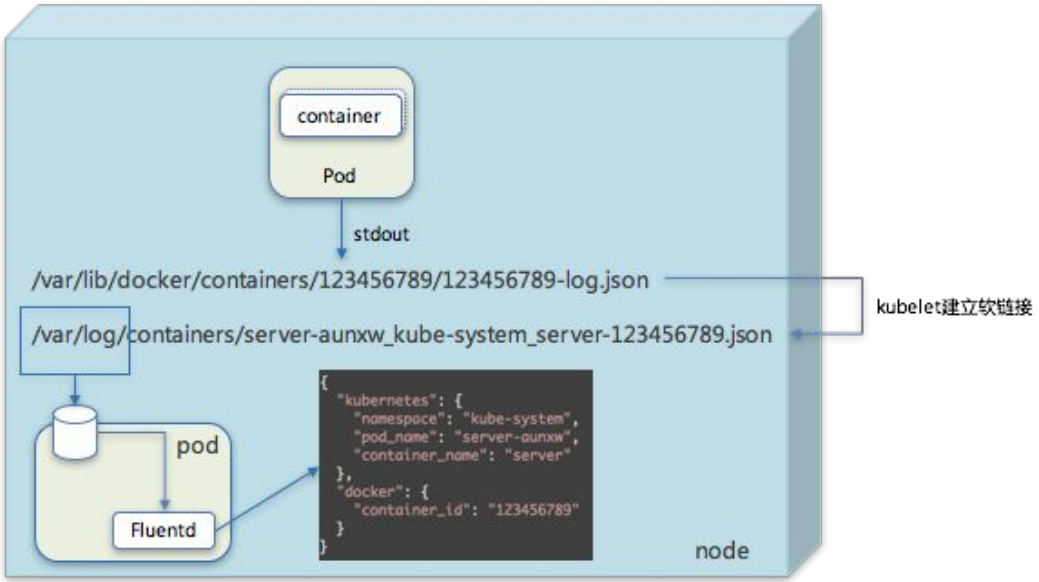
Fluentd (https://github.com/fluent/fluentd)

- 一个统一的日志提取层，可以把收集到的日志以统一的格式输出
- 日志输入可以是多种形式，比如文件，http，tcp 等
- 日志输出可以是转存，比如 mongo，amazon s3 等



Fluentd cont’d

- docker 日志以 json 格式存放在 /var/lib/docker/containers/<container-id>/<container-id>-json.log
- kubelet 自动会在 /var/log/containers 下创建软连接指向这个 docker 日志文件
- fluentd 直接解析 /var/log/containers 下的日志文件并输出



Fluentd cont’d

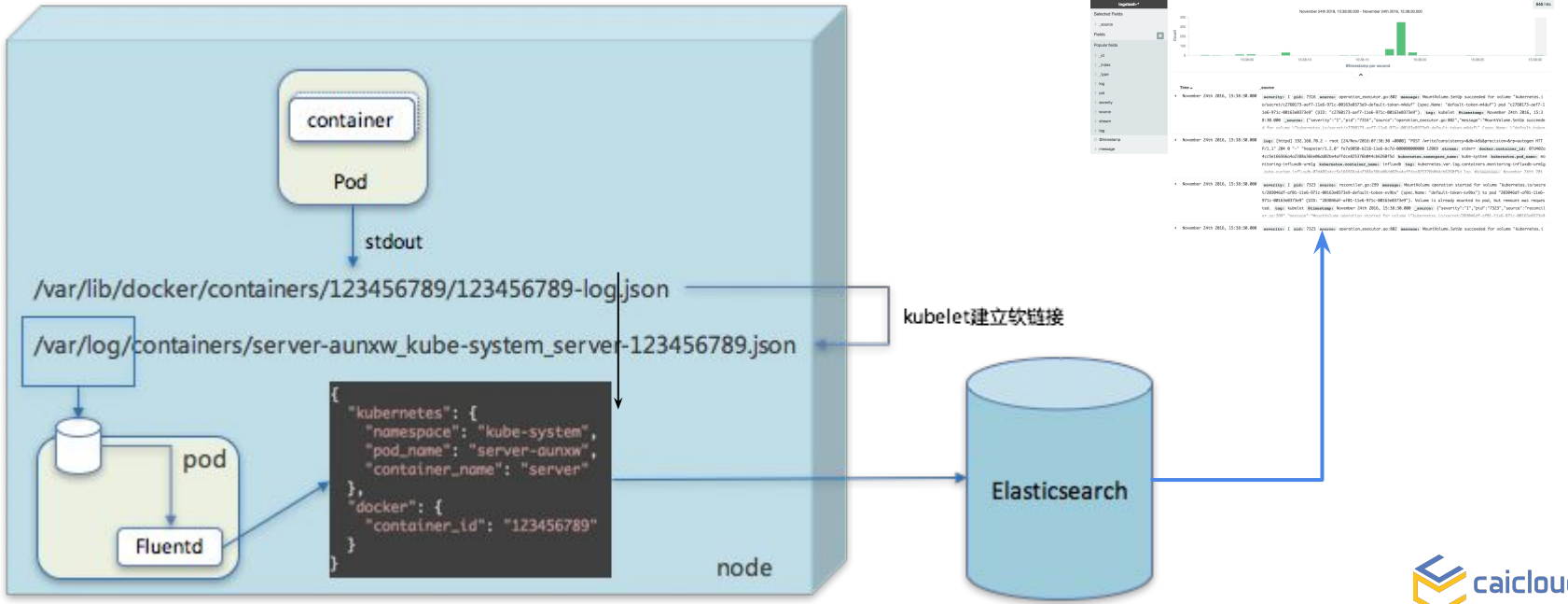
- source 中定义了收集路径，格式，以及，并且给日志打上标签
- filter 匹配标签并且对日志做进一步的处理，这里的 kubernetes_metadata 为一个 fluentd 的 kubernetes 插件，可以为日志添加一些 kubernetes 相关的 metadata
- match 一般被用来描述最后的日志被输出到什么地方

td-agent.conf

```
<source>
   type tail
   path /var/log/containers/*.log
   pos_file /var/log/es-containers.log.pos
   time_format %Y-%m-%dT%H:%M:%S.%NZ
   tag kubernetes.*
   format json
   read_from_head true
</source>

<filter kubernetes.**>
    type kubernetes_metadata
</filter>

<match **>
    type elasticsearch
    host elasticsearch-logging
    port 9200
</match>
```

Elasticsearch + Kibana

- elasticsearch 是一个开源的分布式搜索引擎，用于构建一个高可用，可横向扩展的系统
- kibana 是一个 UI dashboard，可以方便的对 elasticsearch 进行各种查询操作，并且提供了一些可视化的展示



Kubernetes 项目导读

Kubernetes 目录结构

目录	说明
api	k8s restful api 接口的 swagger-spec 文档，kube-apiserver 通过 swagger 工具展示 k8s 所支持的 restful api 接口
build	源码编译相关脚本，支持二进制程序和容器镜像的构建
cluster	k8s 集群部署脚本，支持在裸机、vagrant 以及 aws、azure、gce 等云平台上的部署
cmd	所有二进制程序的入口代码，比如 kube-apiserver、kubectl、kubelet 等
contrib	属于 k8s 生态系统的各种各样的组件，而非 k8s 核心代码，已经转移至单独的 repo
docs	各种文档，包括用户文档、管理员文档、设计文档、新功能提议等，已经转移至单独的 repo
examples	包含一些如何在 k8s 上运行实际应用的例子
federation	集群联邦相关代码，可以直接编译出集群联邦相关的组件
Godeps	该目录由 godep 工具自动生成，存放所有 k8s 项目依赖的 go 第三方包的信息

Kubernetes 目录结构

目录	说明
hack	包含一些编译、构建、测试、校验的脚本
hooks	git commit 之前的一些校验工作
pkg	k8s 核心代码，各个功能组件的具体实现
plugin	k8s 的插件代码，例如 admission control, scheduler 等等
release	k8s 版本发表脚本
test	测试相关代码
third_party	一些第三包工具包
vendor	存放所有 k8s 项目依赖的 go 第三方包

Kubernetes 项目运作

社区合作

- https://github.com/kubernetes/community
- community meeting
- sig-meetings, e.g. sig-security, sig-storage, sig-network
- proposals, design docs

功能开发

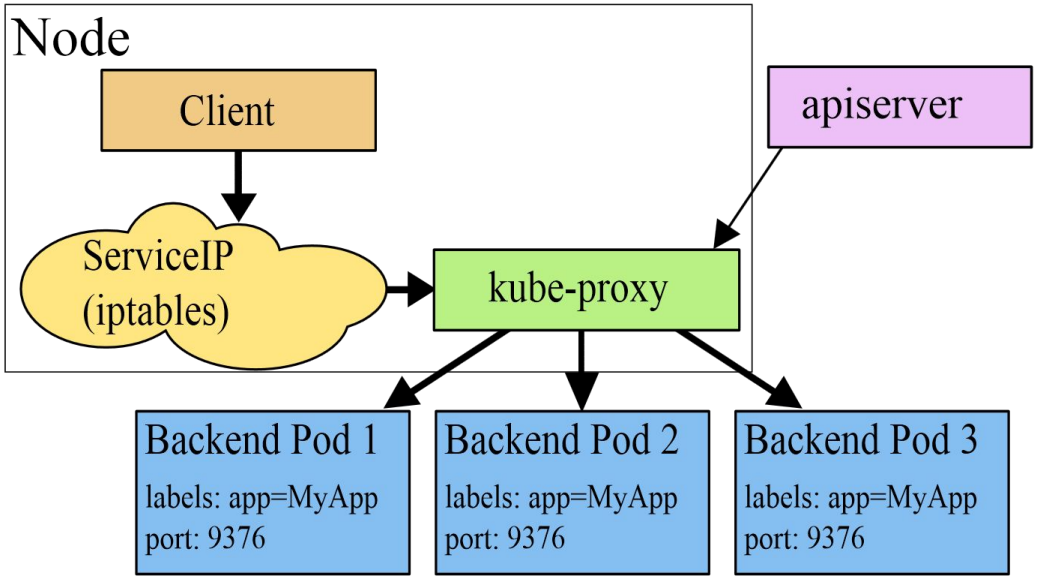
- https://github.com/kubernetes/features
- feature roadmap
- incubator projects: https://github.com/kubernetes-incubator

Backup

Kube-proxy 工作原理：userspace 模式

The iptables rule forwards to a local port where a go binary (kube-proxy) is listening for connections. The binary (running in userspace) terminates the connection, establishes a new connection to a backend for the service, and then forwards requests to the backend and responses back to the local process.

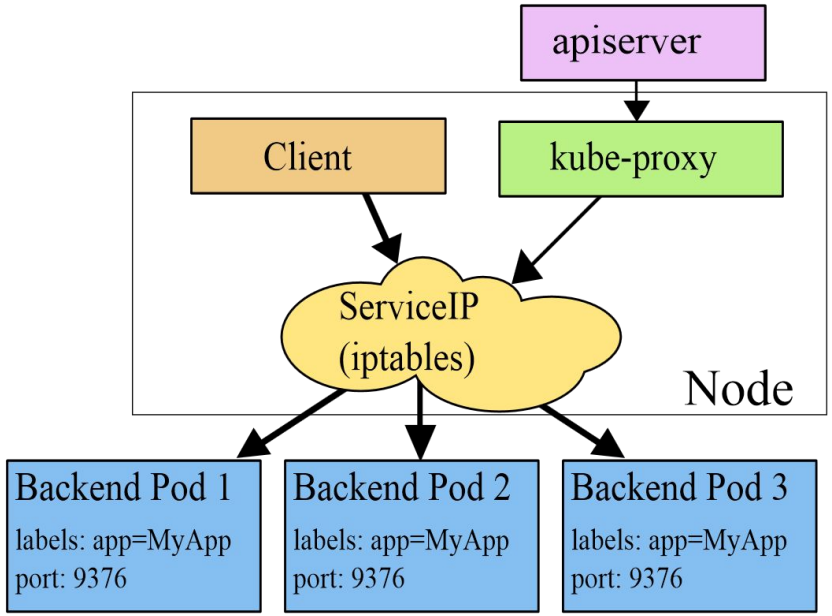
缺点：Packets from the kernel to kube-proxy and then back to the kernel so it results in low throughput and big tail latency.



Kube-proxy 工作原理：iptables 模式

In iptables mode, the iptables rules are installed to directly forward packets that are destined for a service to a backend for the service.

缺点：The main downside is that it is more difficult to debug, because instead of a local binary that writes a log to /var/log/kube-proxy you have to inspect logs from the kernel processing iptables rules.



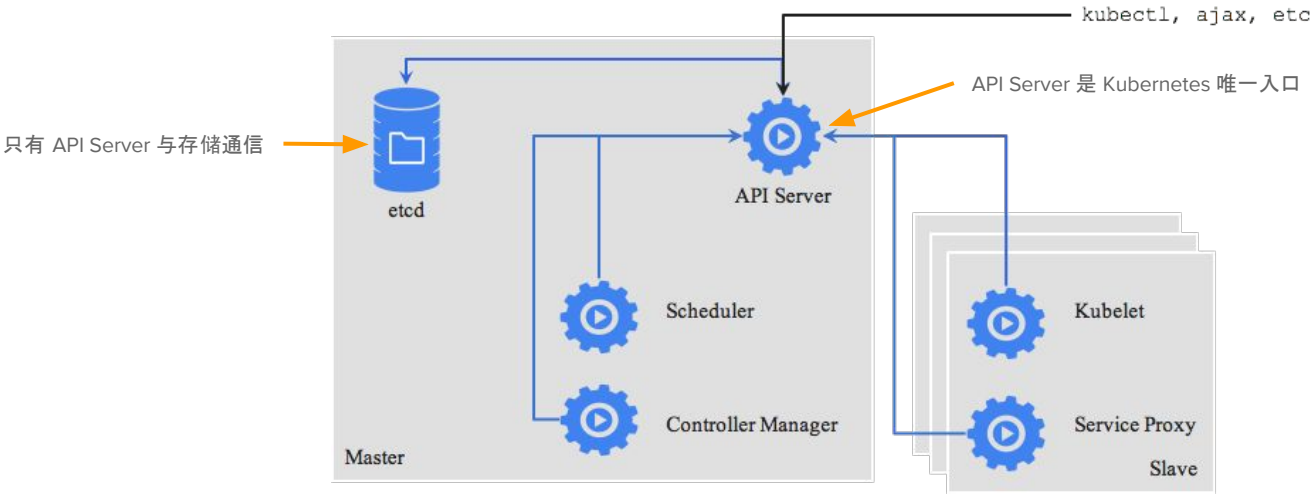
其他的应用管理操作

- 标签操作
  - kubectl label pods -l app=nginx tier=fe
  - kubectl get pods -l app=nginx -L tier
- 更新注释
  - kubectl annotate pods my-nginx-v4-9gw19 description='my frontend running nginx'
  - kubectl get pods my-nginx-v4-9gw19 -o yaml
- 更改应用体量
  - kubectl scale deployment/my-nginx --replicas=1
- 修改应用配置
  - kubectl apply -f docs/user-guide/nginx/nginx-deployment.yaml
  - kubectl edit deployment/my-nginx
  - kubectl patch deployment my-nginx -p'{"spec":{"template":{"spec":{"containers":[{"name":"nginx","image":"nginx"}]}}}}'
  - kubectl replace -f docs/user-guide/nginx/nginx-deployment.yaml --force

Label Matching

```yaml
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: frontend
  # these labels can be applied automatically
  # from the labels in the pod template if not set
  # labels:
    # app: guestbook
    # tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  # selector can be applied automatically
  # from the labels in the pod template if not set,
  # but we are specifying the selector here to
  # demonstrate its usage.
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
```

Kubernetes 架构解析



API definition

```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

kubernetes 大版本号

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

API definition

```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

资源类型

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

API definition

```
apiVersion: v1
kind: Pod
metadata:
  namespace: default
spec:
  containers:
  dnsPolicy: ClusterFirst
  nodeName: i-2zea47skez7ye2xr438v
  restartPolicy: Always
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  volumes: xxx
status:
  conditions: xxx
  hostIP: 10.44.164.150
  phase: Running
  podIP: 192.168.79.9
  startTime: 2016-11-22T14:54:57Z
```

元数据

```
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: minikube
  name: minikube
  resourceVersion: "1027609"
  selfLink: /api/v1/nodesminikube
  uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
  externalID: minikube
status:
  addresses:
  - address: 192.168.99.101
    type: LegacyHostIP
  - address: 192.168.99.101
    type: InternalIP
  - address: minikube
    type: Hostname
```

API definition

```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

资源说明书，配置

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

API definition

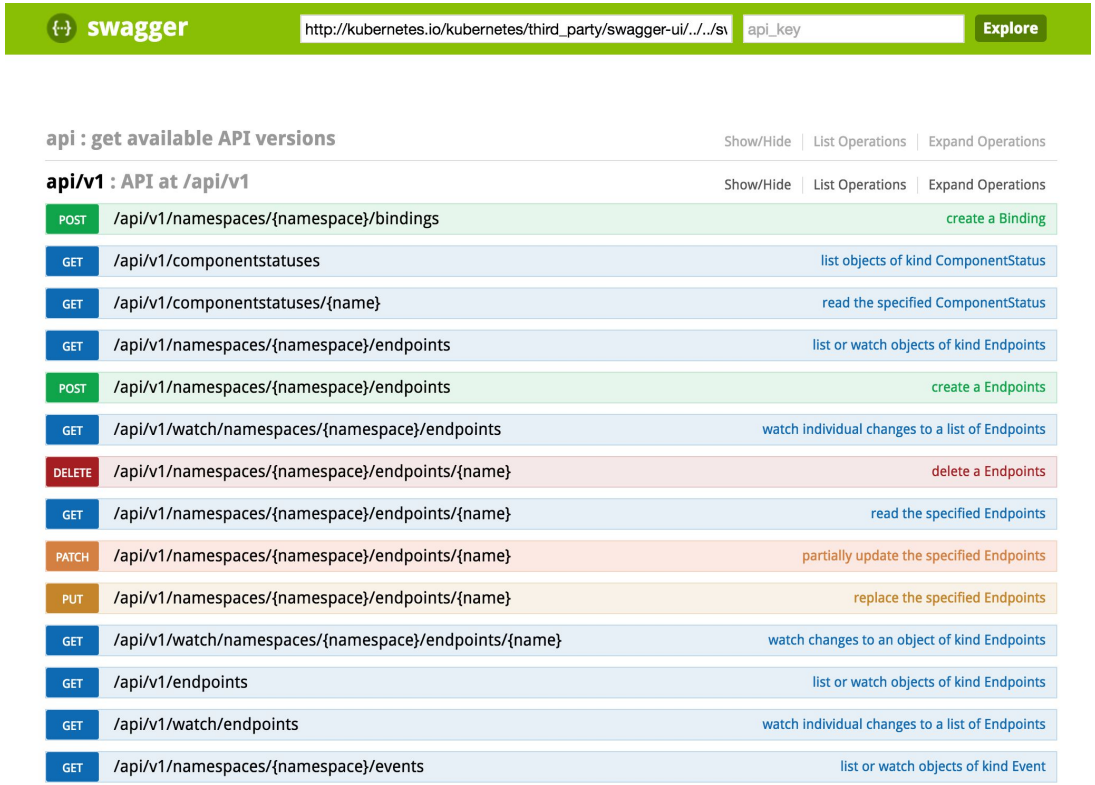
```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    volumes: xxx
status:
    conditions: xxx
    hostIP: 10.44.164.150
    phase: Running
    podIP: 192.168.79.9
    startTime: 2016-11-22T14:54:57Z
```

资源状态

```
apiVersion: v1
kind: Node
metadata:
    labels:
      beta.kubernetes.io/arch: amd64
      beta.kubernetes.io/os: linux
      kubernetes.io/hostname: minikube
    name: minikube
    resourceVersion: "1027609"
    selfLink: /api/v1/nodesminikube
    uid: 21ffcb42-0f69-11e7-a9ff-080027e561ba
spec:
    externalID: minikube
status:
    addresses:
    - address: 192.168.99.101
      type: LegacyHostIP
    - address: 192.168.99.101
      type: InternalIP
    - address: minikube
      type: Hostname
```

API endpoints

http://kubernetes.io/kubernetes/third_party/swagger-ui/#/



API group

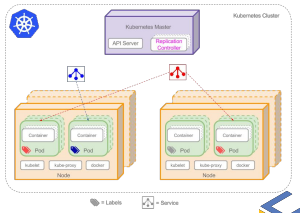
- 每个 API group 中的资源可以单独的 enable/disable
- 每个 API group 有不同的版本，单独开发维护


```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/apps",
    "/apis/apps/v1alpha1",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1beta1",
    "/apis/autoscaling",
    "/apis/autoscaling/v1",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v2alpha1",
    "/apis/certificates.k8s.io",
    "/apis/certificates.k8s.io/v1alpha1",
    "/apis/extensions",
    "/apis/extensions/v1beta1",
    "/apis/k8s.io",
    "/apis/k8s.io/v1",
    "/apis/policy",
    "/apis/policy/v1alpha1",
    "/apis/rbac.authorization.k8s.io",
    "/apis/rbac.authorization.k8s.io/v1alpha1",
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1beta1",
    "/healthz",
    "/healthz/ping",
    "/logs",
    "/metrics",
    "/swaggerapi/",
    "/ui/",
    "/version"
  ]
}
```

API example: Pod Health Check

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        livenessProbe:
          httpGet:
            # Path to probe; should be cheap, but representative of typical behavior
            path: /index.html
            port: 80
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

POST to “/api/v1/namespaces/default/pods”



caicloud

Kubernetes advanced concepts

LimitRange

- LimitRange: 默认 Pod，Container 资源配额
  - 计算资源配额
    - CPU，MEM
- 另外，可以在 Pod 中设置 Container 资源配额

```
apiVersion: v1
kind: ReplicationController
...
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        cpu: 100m
        memory: 100Mi
```

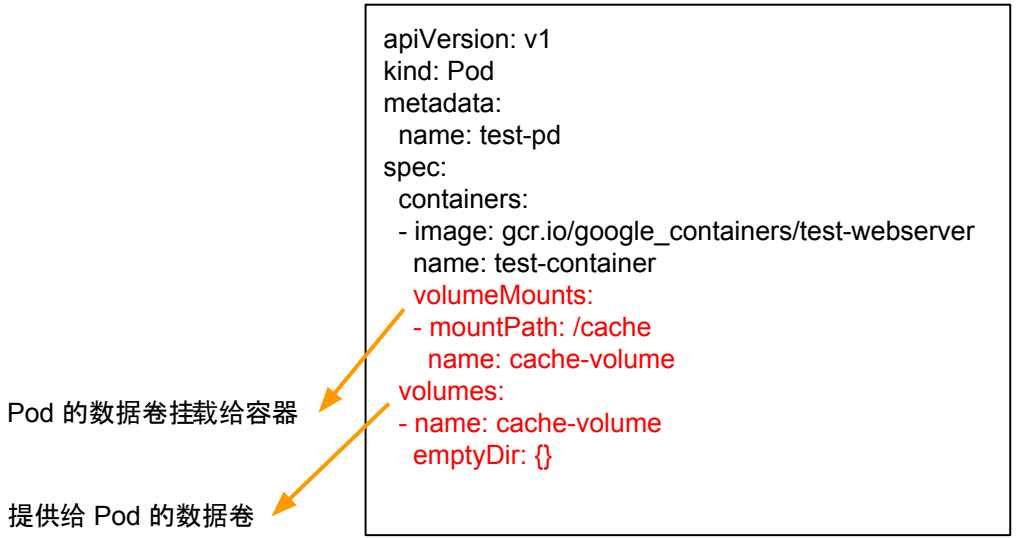
1. 一个Pod的所有容器内存使用必须在6Mi ~ 1Gi
2. 一个Pod的所有容器的CPU使用必须在250m ~ 2 cores
3. 一个容器的内存使用必须在6Mi ~ 1Gi，默认是100Mi
4. 一个容器的CPU使用必须在250m ~ 2 cores，默认是250m

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mylimits
spec:
  limits:
  - type: Pod
    max:
      cpu: "2"
      memory: 1Gi
    min:
      cpu: 250m
      memory: 6Mi
  - type: Container
    default:
      cpu: 250m
      memory: 100Mi
    max:
      cpu: "2"
      memory: 1Gi
    min:
      cpu: 250m
      memory: 6Mi
```

Namespace, etc Demo

Volumes

- Pod 生命周期内一直可用
- Pod 内容器共享
- 当前支持的类型
  - emptyDir: Pod 与 Node 绑定/解绑时创建/删除
  - hostPath: 挂载 Node 上的目录
  - gcePersistentDisk
  - nfs
  - glusterfs
  - ...



PersistentVolume (PV) 和 PersistentVolumeClaim (PVC)

- PV（持久卷）
  - 集群中的一块网络存储
  - 独立于 Pod 的生命周期
  - 支持的 PV 类型
    - GCEPersistentDisk
    - AWSElasticBlockStore
    - NFS
    - Cinder
    - iSCSI
    - ...
  - 集群管理员创建一系列的 PV
- PVC（持久卷申请）
  - 请求特定尺寸和访问模式的 PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: nfs
spec:
 capacity:
   storage: 1Mi
 accessModes:
  - ReadWriteMany
 nfs:
   # FIXME: use the right IP
   server: 10.244.1.4
   path: "/exports"
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: nfs
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 1Mi
```

```
apiVersion: v1
kind: ReplicationController
metadata:
 name: nfs-web
spec:
 replicas: 2
 template:
   metadata:
     labels:
       role: web-frontend
   spec:
     containers:
     - name: web
       image: nginx
       ports:
         - name: web
           containerPort: 80
       volumeMounts:
           - name: nfs
             mountPath: "/usr/share/nginx/html"
     volumes:
     - name: nfs
       persistentVolumeClaim:
         claimName: nfs
```

StorageClass, PV/PVC Demo

Node Affinity / Pod Affinity & Anti-affinity

- 核心：选择一个 Node 运行
- Node Affinity
  - “节点” 满足某个需求才 调度
- Pod Affinity
  - “节点” 有满足需求 Pod
- Pod Anti-affinity
  - “节点” 没有满足需求的 Pod

Node Affinity / Pod Affinity & Anti-affinity Demo

Taints/Toleration

- 核心：选择一个 Node 运行
- Taints
  - 污点
- Toleration
  - 容忍

只有 “容忍” 了 Node “污点” 的 Pod 可以运行在此 Node 上

Taints/Toleration Demo

HPA (Horizontal Pod Autoscaler)

- 基于 CPU 利用率自动增加或者减少 Pod 副本数
- 支持 RS/RC/Deployment

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
 name: frontend-scaler
spec:
 scaleTargetRef:
   kind:
ReplicaSet/ReplicationController/Deployment
   name: frontend
 minReplicas: 3
 maxReplicas: 10
 targetCPUUtilizationPercentage: 50
```

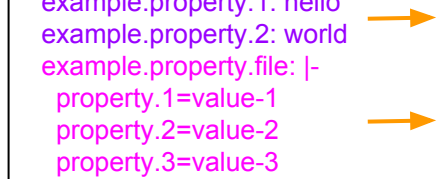
还可以通过 kubectl 设置 HPA：

kubectl autoscale deployment frontend --cpu-percent=50 --min=3 --max=10

ConfigMap

- 应用镜像和配置分离
- 数据类型为键值对
- Pod 使用 ConfigMap 的三种方式
  - 命令行参数
  - 环境变量
  - 数据卷文件

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data:
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```



ConfigMap 作为环境变量和命令行参数

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
  restartPolicy: Never
```

命令行参数

环境变量

ConfigMap 作为数据卷文件

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

2

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh","-c","cat /etc/config/path/to/special-key"
]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
        - key: special.how
          path: path/to/special-key
  restartPolicy: Never
```

key 指定文件路径名
value 作为文件内容

1

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat /etc/config/special.how"
]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
  restartPolicy: Never
```

key 作为文件名
value 作为文件内容

ConfigMap 配置 redis 实例

创建配置:

```
$ cat redis-config.txt
maxmemory 2mb
maxmemory-policy allkeys-lru

$ kubectl create configmap example-redis-config --from-file=redis-config.txt

$ kubectl get configmap example-redis-config -o yaml

apiVersion: v1
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
kind: ConfigMap
metadata:
  creationTimestamp: 2016-03-30T18:14:41Z
  name: example-redis-config
  namespace: default
  resourceVersion: "24686"
  selfLink: /api/v1/namespaces/default/configmaps/example-redis-config
  uid: 460a2b6e-f6a3-11e5-8ae5-42010af00002
```

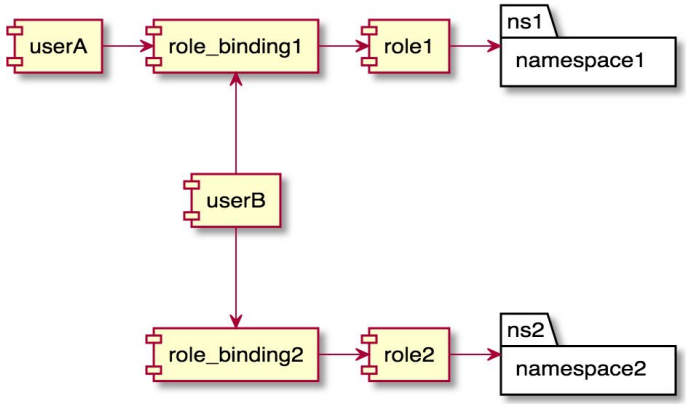
使用配置:

```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
  - name: redis
    image: kubernetes/redis:v1
    env:
    - name: MASTER
      value: "true"
    ports:
    - containerPort: 6379
    resources:
      limits:
        cpu: "0.1"
    volumeMounts:
    - mountPath: /redis-master-data
      name: data
    - mountPath: /redis-master
      name: config
  volumes:
    - name: data
      emptyDir: {}
    - name: config
      configMap:
        name: example-redis-config
        items:
        - key: redis-config
          path: redis.conf
```

Role-Based Access Control（RBAC）

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1alpha1
metadata:
 namespace: default
 name: pod-reader
rules:
 - apiGroups: [""] # The default API Group.
   resources: ["pods"]
   verbs: ["get", "watch", "list"]
   nonResourceURLs: []
```

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1alpha1
metadata:
 name: read-pods
 namespace: default
subjects:
 - kind: User # May be "User", "Group" or "ServiceAccount"
   name: jane
roleRef:
 kind: Role
 namespace: default
 name: pod-reader
 apiVersion: rbac.authorization.k8s.io/v1alpha1
```



- RBAC 引入角色（Role）和角色绑定（RoleBinding）抽象概念
- 访问策略可以跟某个或者多个角色关 联

```
kube-apiserver    --authorization-mode=RBAC
--runtime-config=extensions/v1beta1/networkpolicies=true,rbac.authorization.k8s.io/v1alpha1
--authorization-rbac-super-user=caicloud-admin
  ...
```

Attribute-Based Access Control（ABAC）

```
{

"apiVersion":"abac.authorization.kubernetes.io/v1beta1",
   "kind":"Policy",
   "spec":{
      "user":"alice",
      "namespace":"*",
      "resource":"*",
      "apiGroup":"*"
   }
}
```

Alice can do anything to all resources

```
{
   "apiVersion":"abac.authorization.kubernetes.io/v1beta1",
   "kind":"Policy",
   "spec":{
      "user":"bob",
      "namespace":"test",
      "resource":"pods",
      "readonly":true
   }
}
```

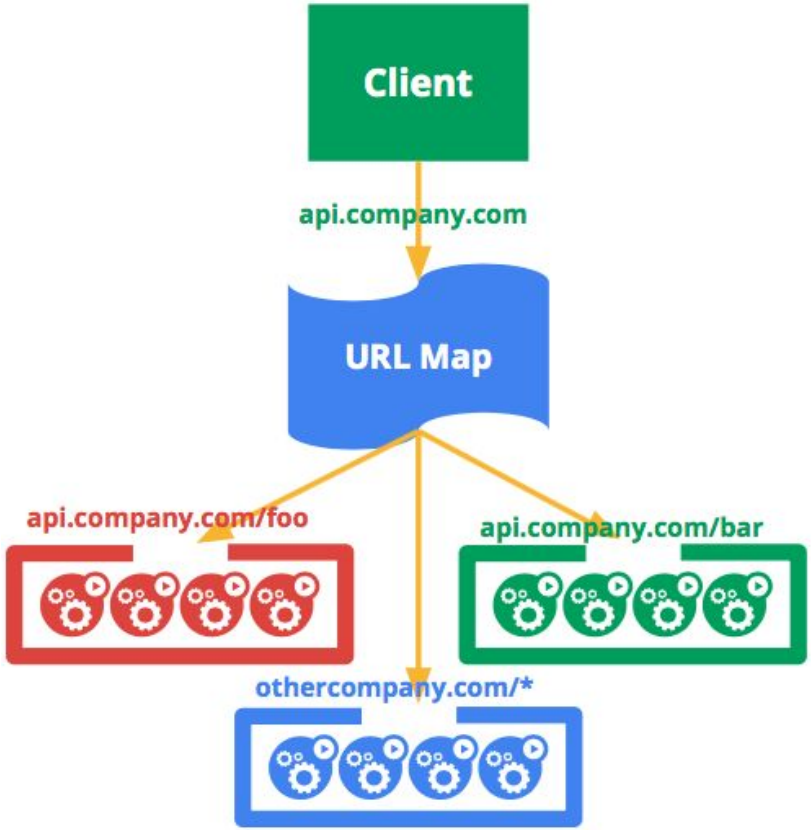
Bob can just read pods in namespace ‘test’

- ABAC 配置不同用户的访问权限
- 缺点：
  - 对权限做修改，必须重启 apiserver
  - 访问策略只能跟用户直接关联

```
kube-apiserver  --authorization-mode=ABAC  --authorization-policy-file=/etc/kubernetes/tokens/abac.json  ...
```

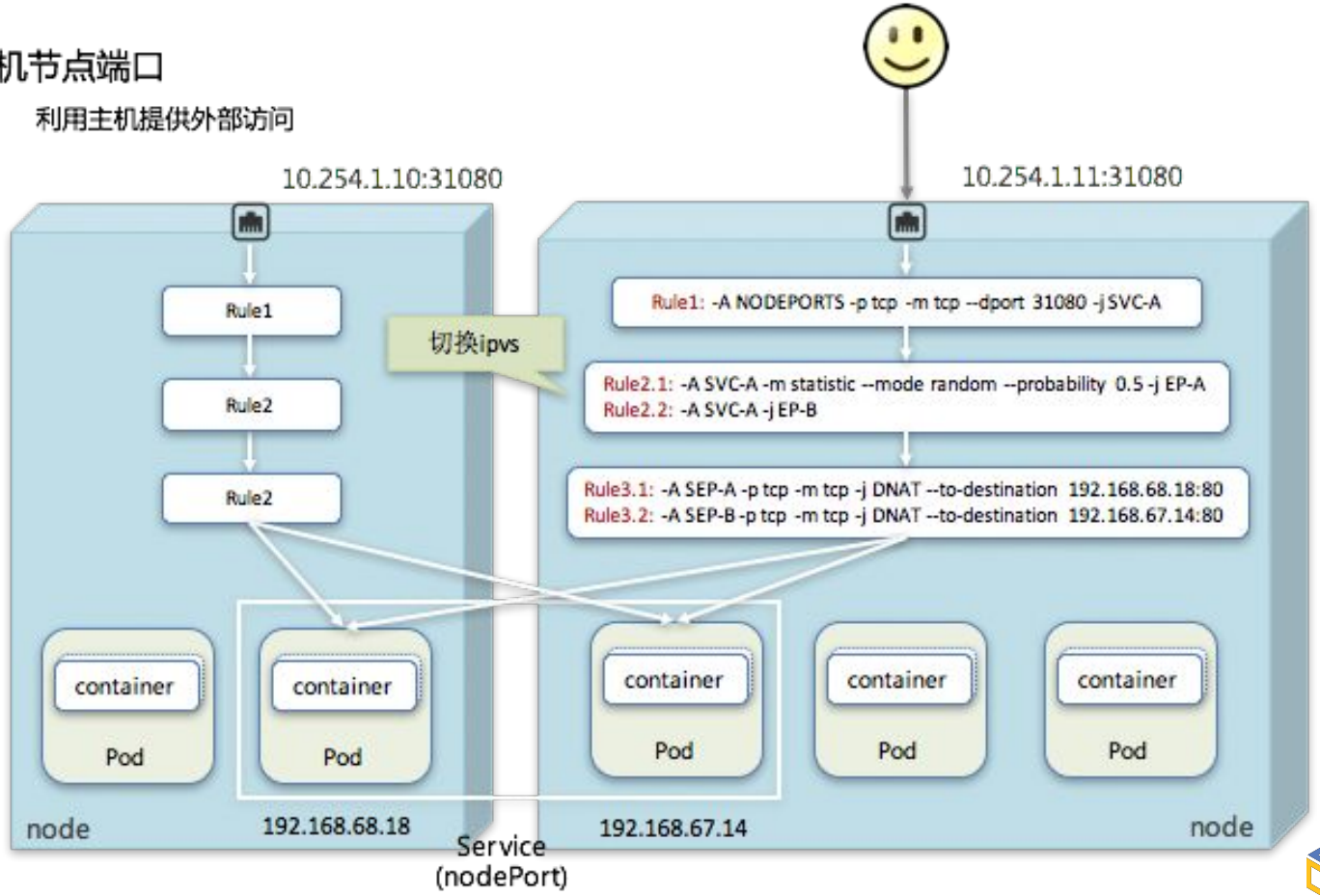
Ingress (L7)

- Services are assumed L3/L4
- Lots of apps want HTTP/HTTPS
- Ingress maps incoming traffic to backend services
  - by HTTP host headers
  - by HTTP URL paths
- Nginx, HAProxy and GCE implementation



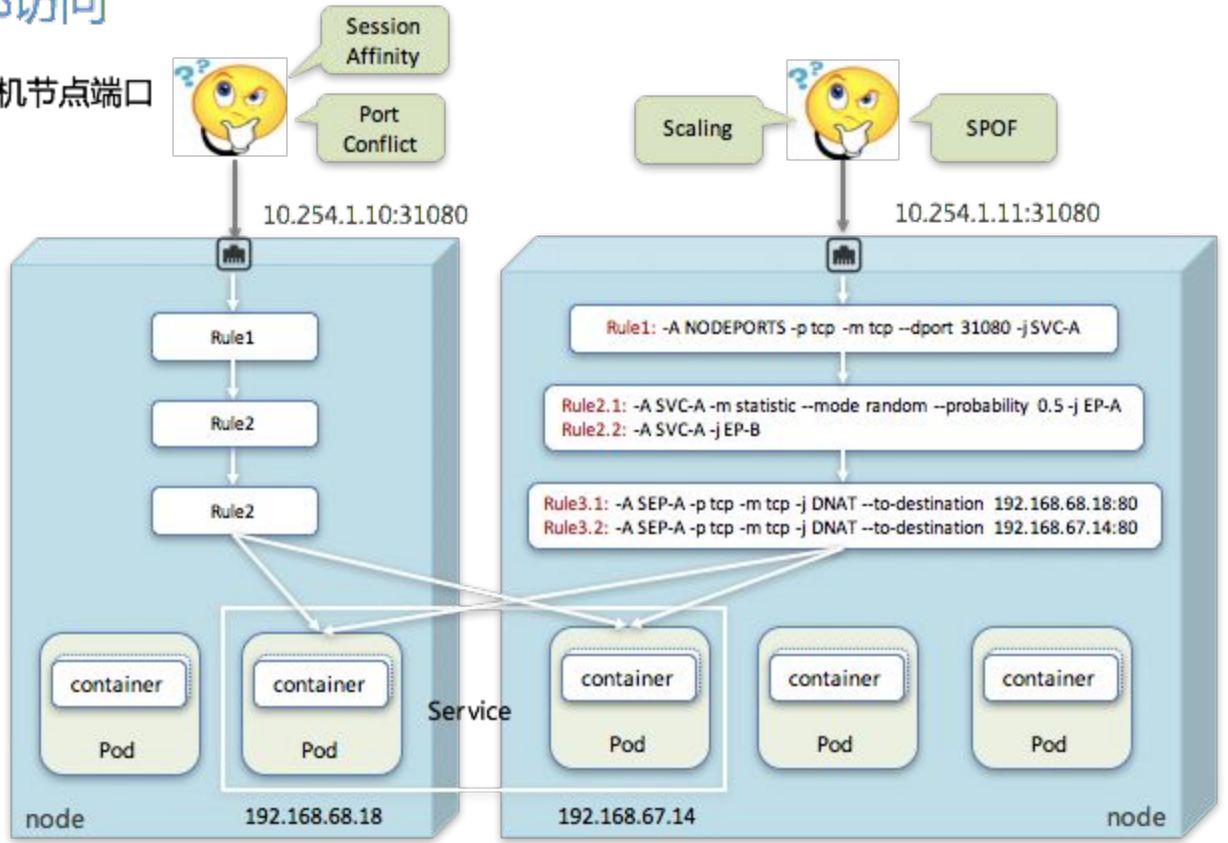
外部访问

- 主机节点端口
  - 利用主机提供外部访问



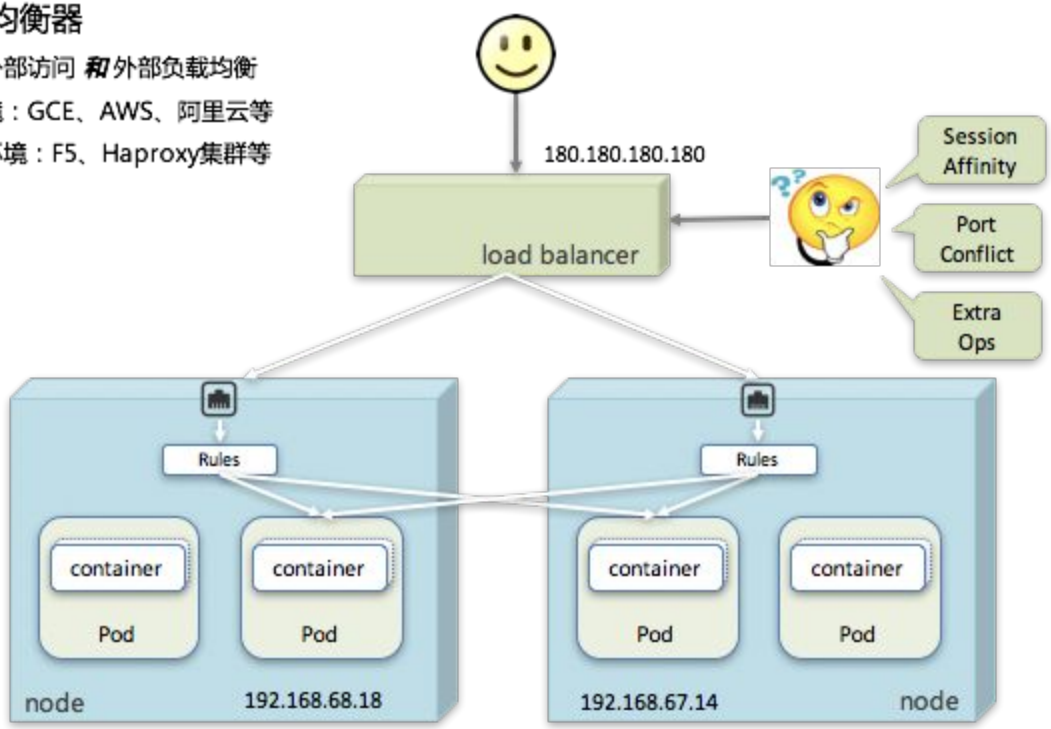
外部访问

- 主机节点端口


Session Affinity
Port Conflict
Scaling
SPOF
10.254.1.10:31080
10.254.1.11:31080
Rule1
Rule2
Rule2
Rule1: -A NODEPORTS -p tcp -m tcp --dport 31080 -j SVC-A
Rule2.1: -A SVC-A -m statistic --mode random --probability 0.5 -j EP-A
Rule2.2: -A SVC-A -j EP-B
Rule3.1: -A SEP-A -p tcp -m tcp -j DNAT --to-destination 192.168.68.18:80
Rule3.2: -A SEP-A -p tcp -m tcp -j DNAT --to-destination 192.168.67.14:80
container
Pod
container
Pod
Service
container
Pod
container
Pod
container
Pod
node
192.168.68.18
192.168.67.14
node

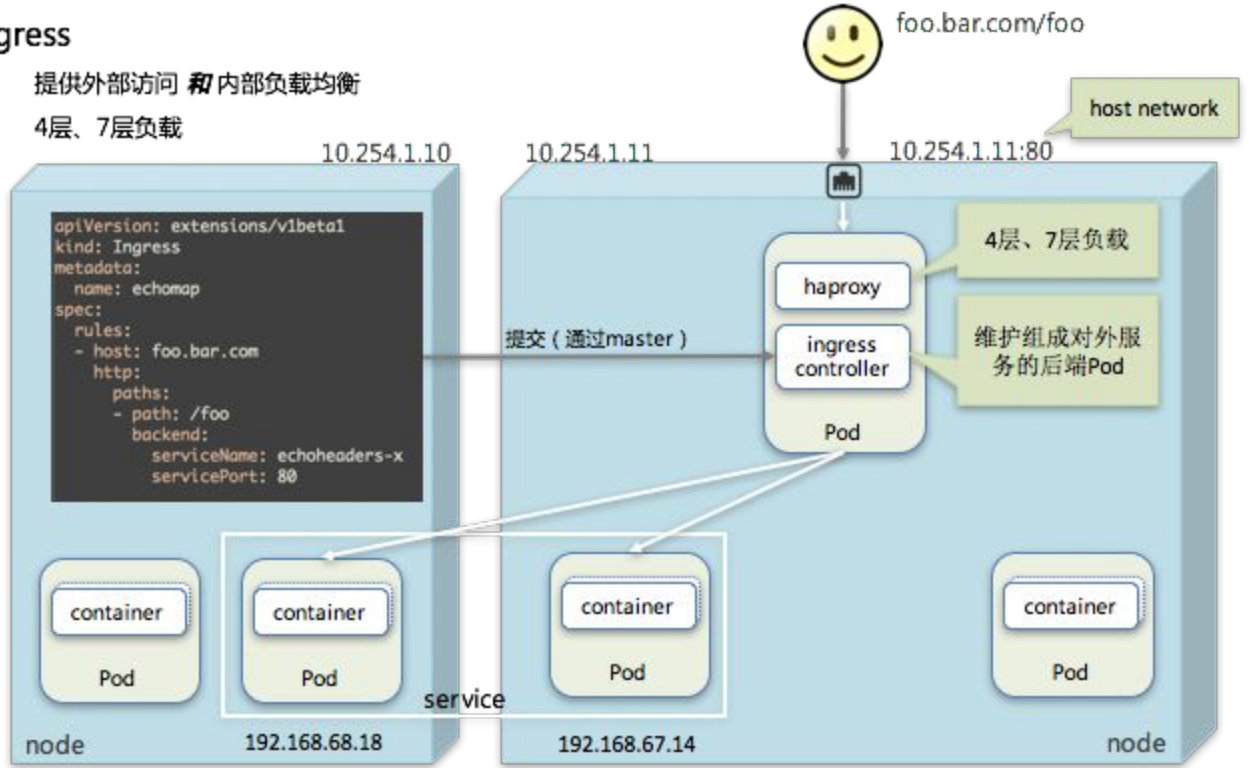
外部访问

- 外部负载均衡器
  - 提供外部访问 ***和*** 外部负载均衡
  - 云环境：GCE、AWS、阿里云等
  - 私有环境：F5、Haproxy集群等



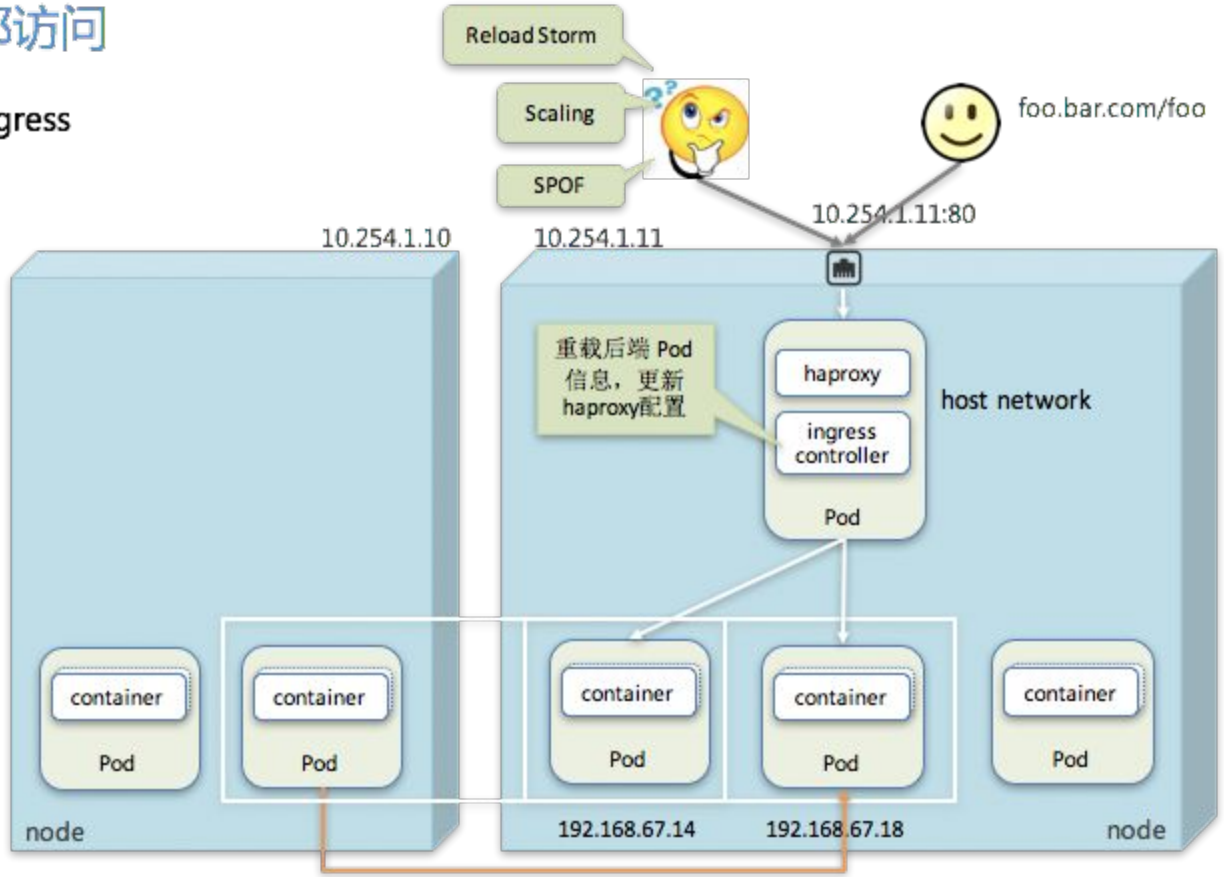
外部访问

- Ingress
  - 提供外部访问 ***和*** 内部负载均衡
  - 4层、7层负载



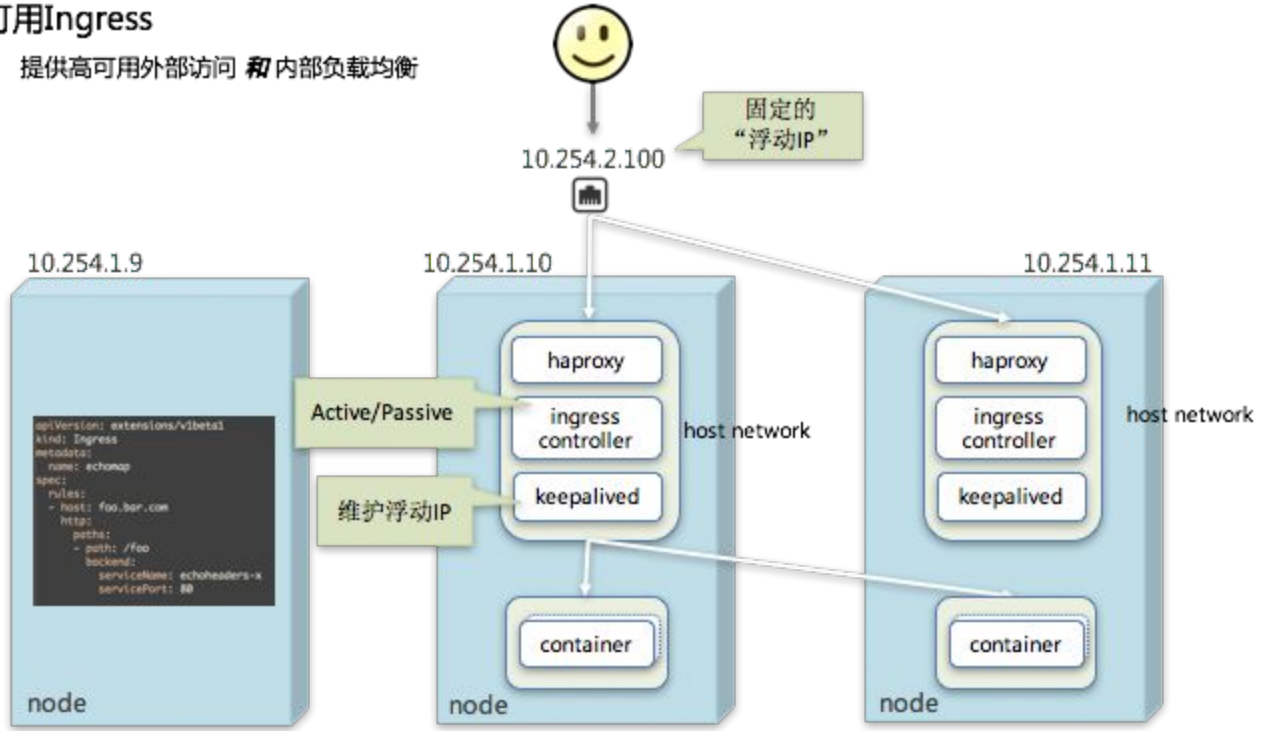
外部访问

- Ingress



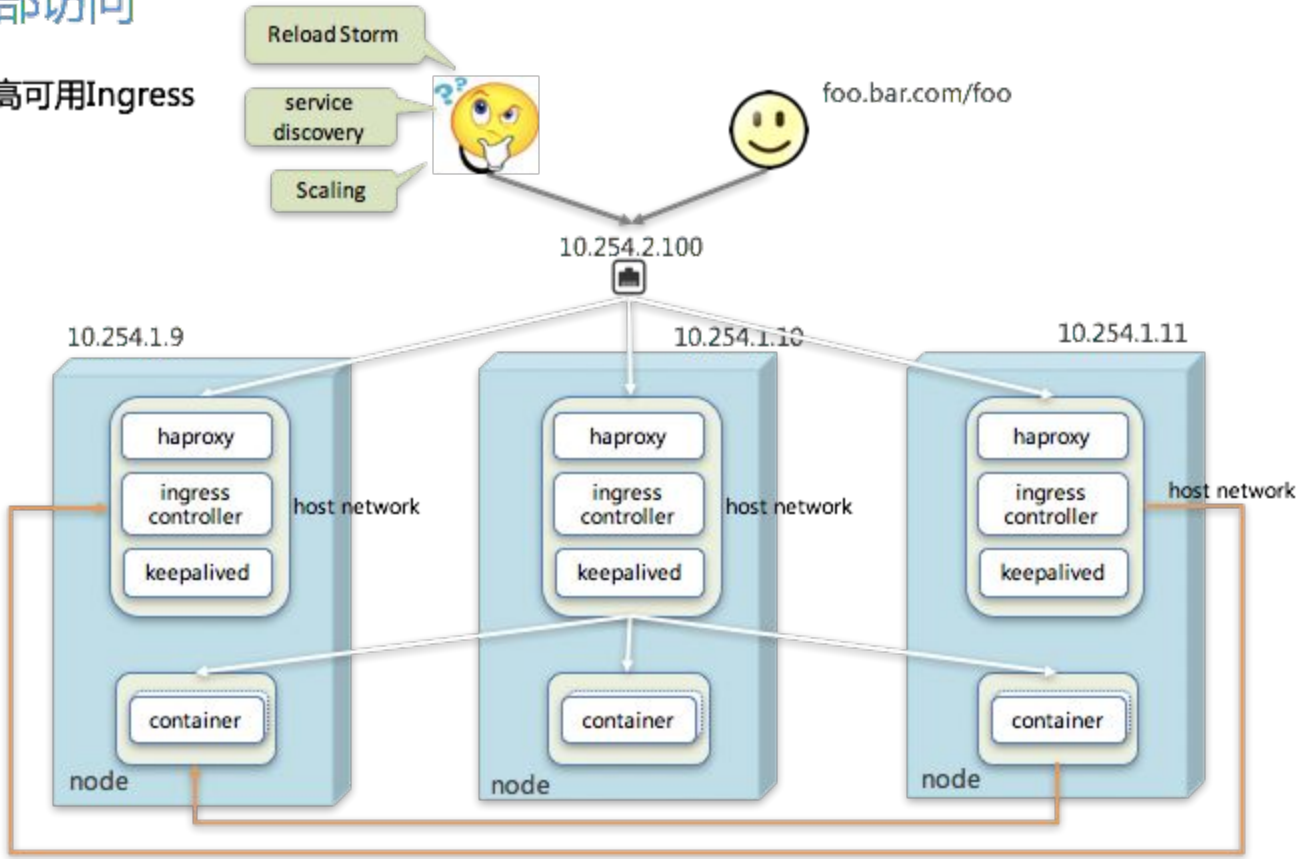
外部访问

- 高可用Ingress
  - 提供高可用外部访问 *和* 内部负载均衡



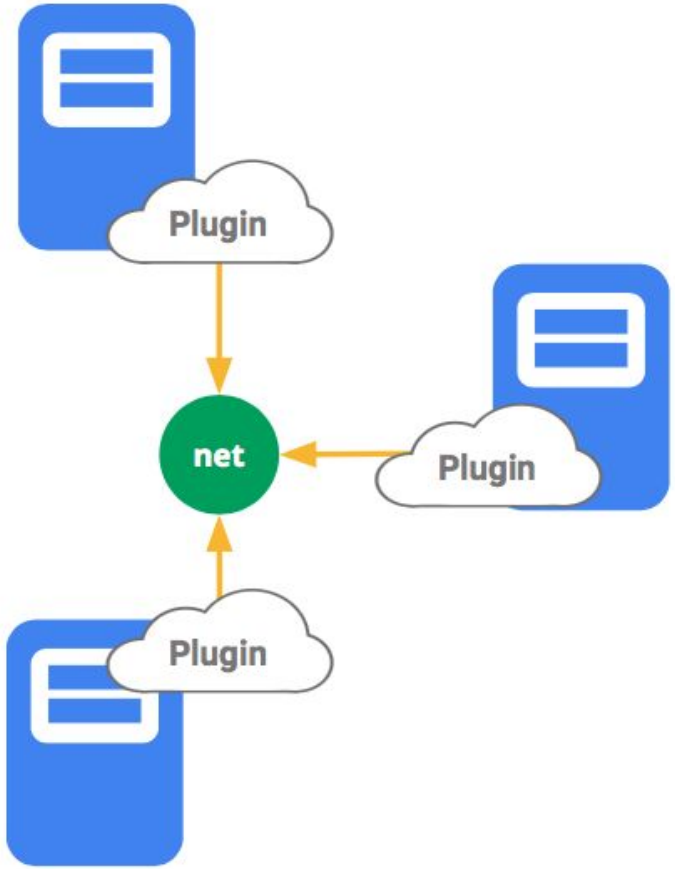
外部访问

- 高可用Ingress



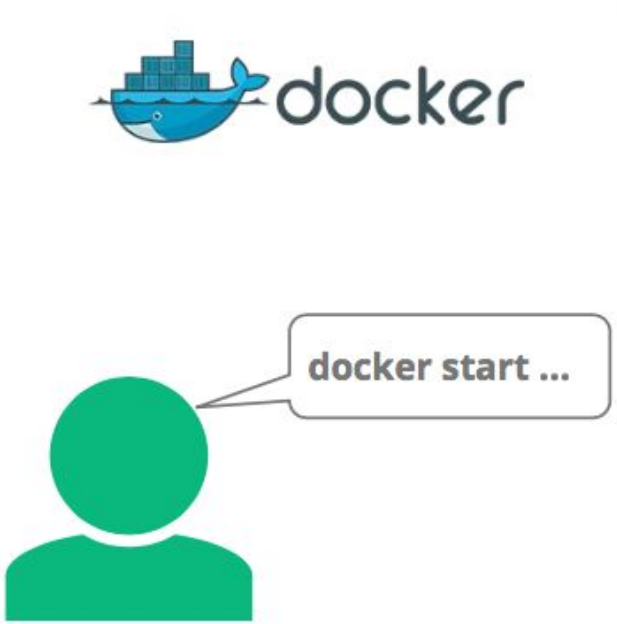
Network Plugins

- Introduced in Kubernetes v1.0
  - very experimental
- Uses CNI (From CoreOS) in v1.1
  - simple exec interface
  - not using docker libnetwork
- Cluster admins can customize their installs
  - DHCP, MacVlan, Flannel, custom plugin

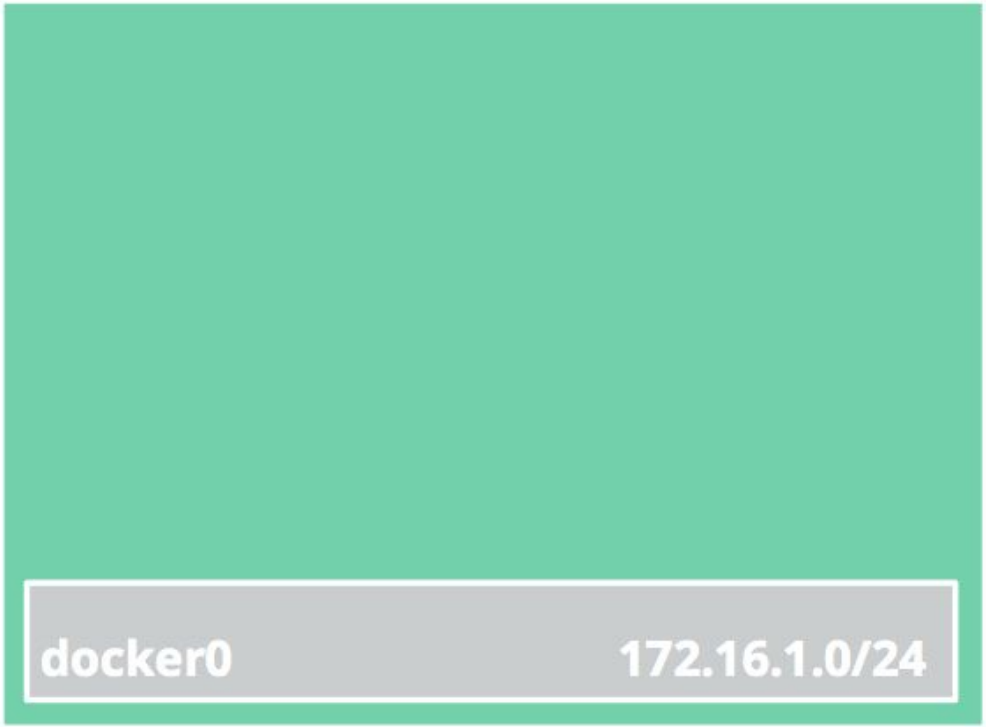


Kubernetes 网络模型

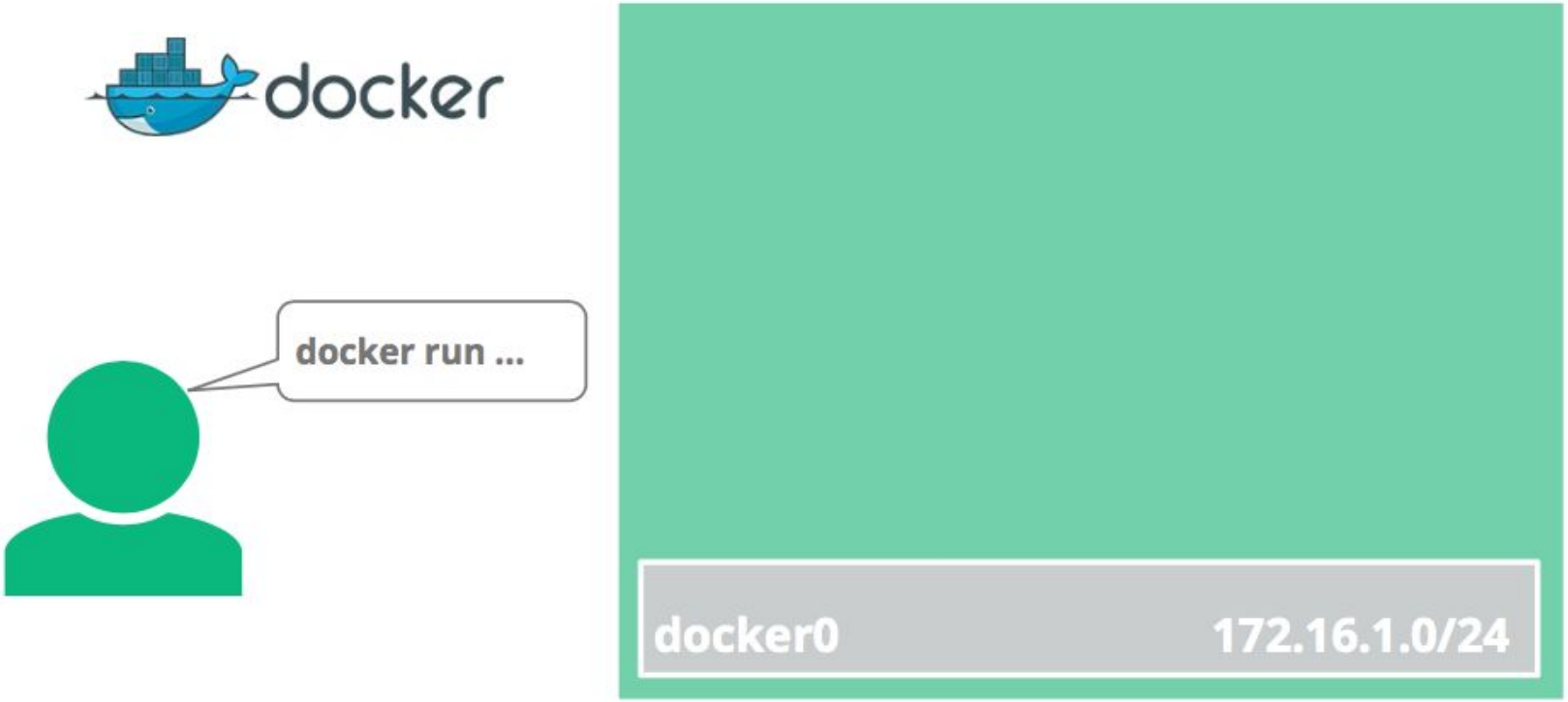
Docker Networking



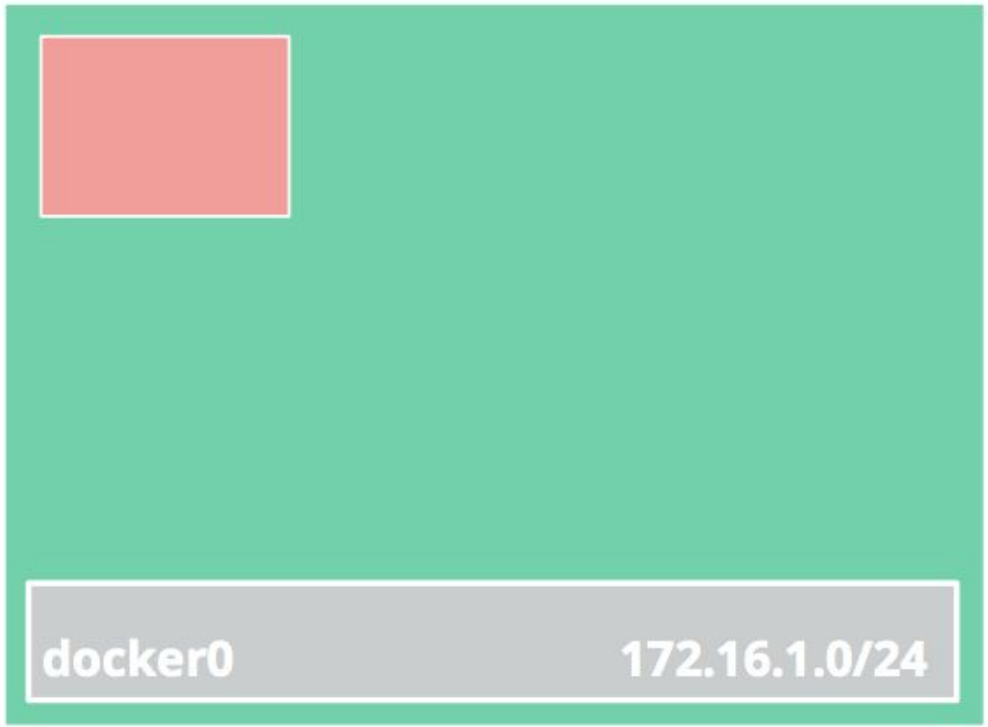
Docker Networking



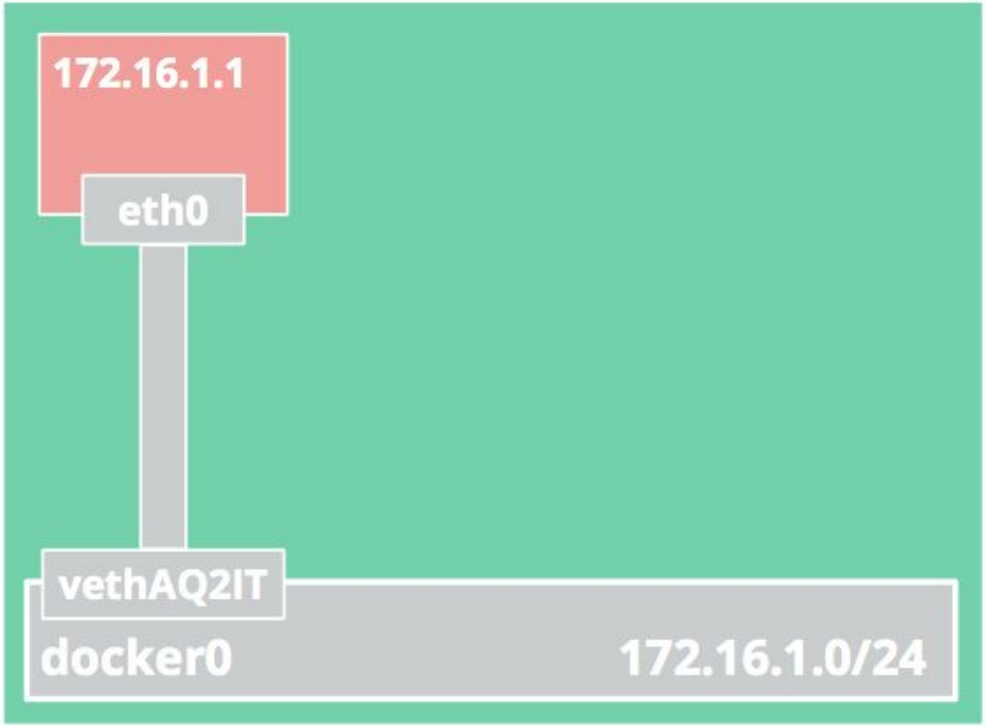
Docker Networking



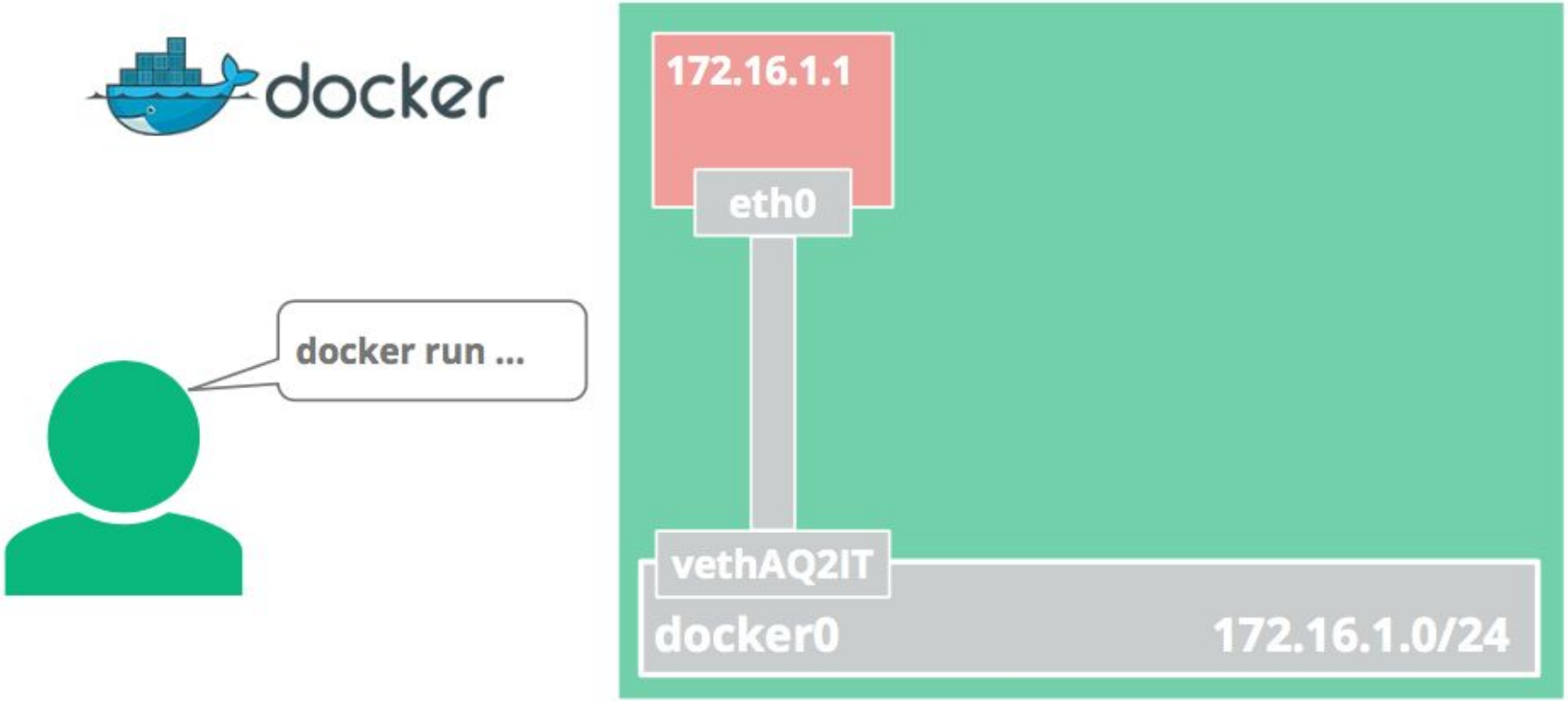
Docker Networking



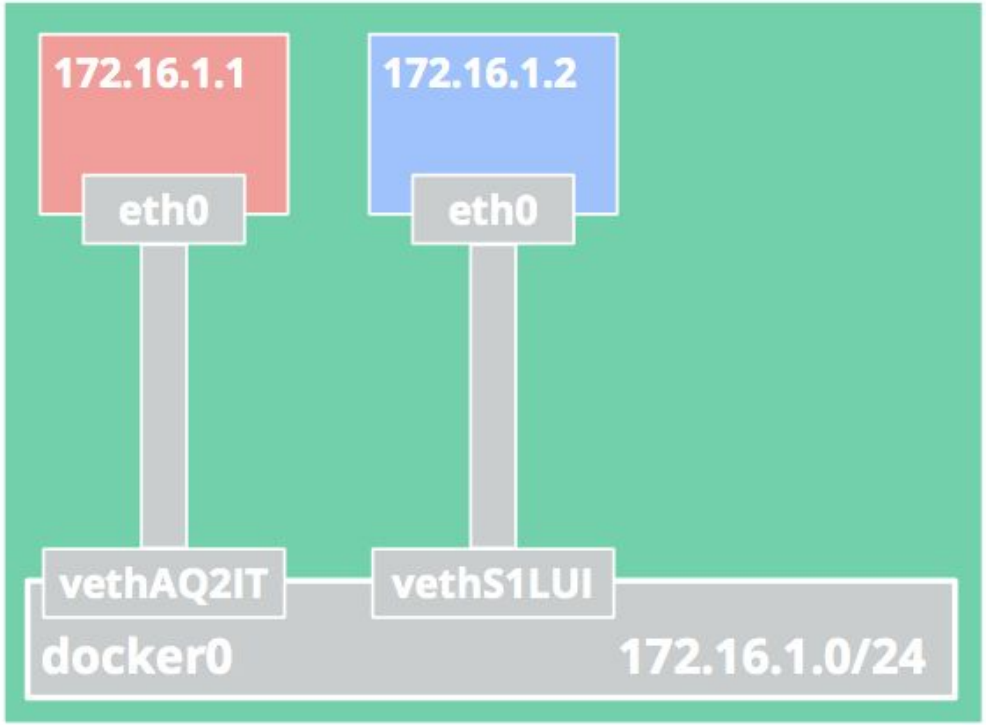
Docker Networking



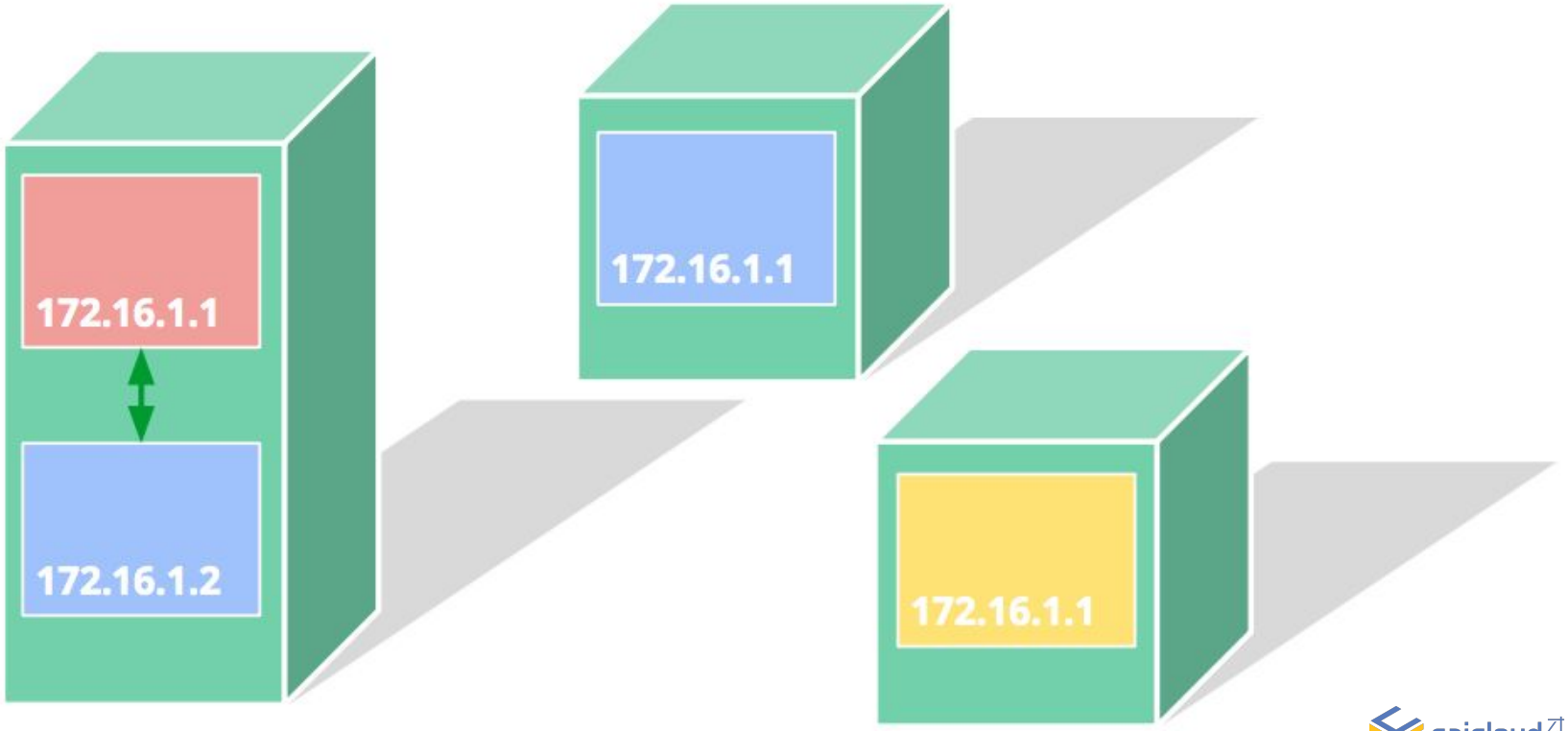
Docker Networking



Docker Networking

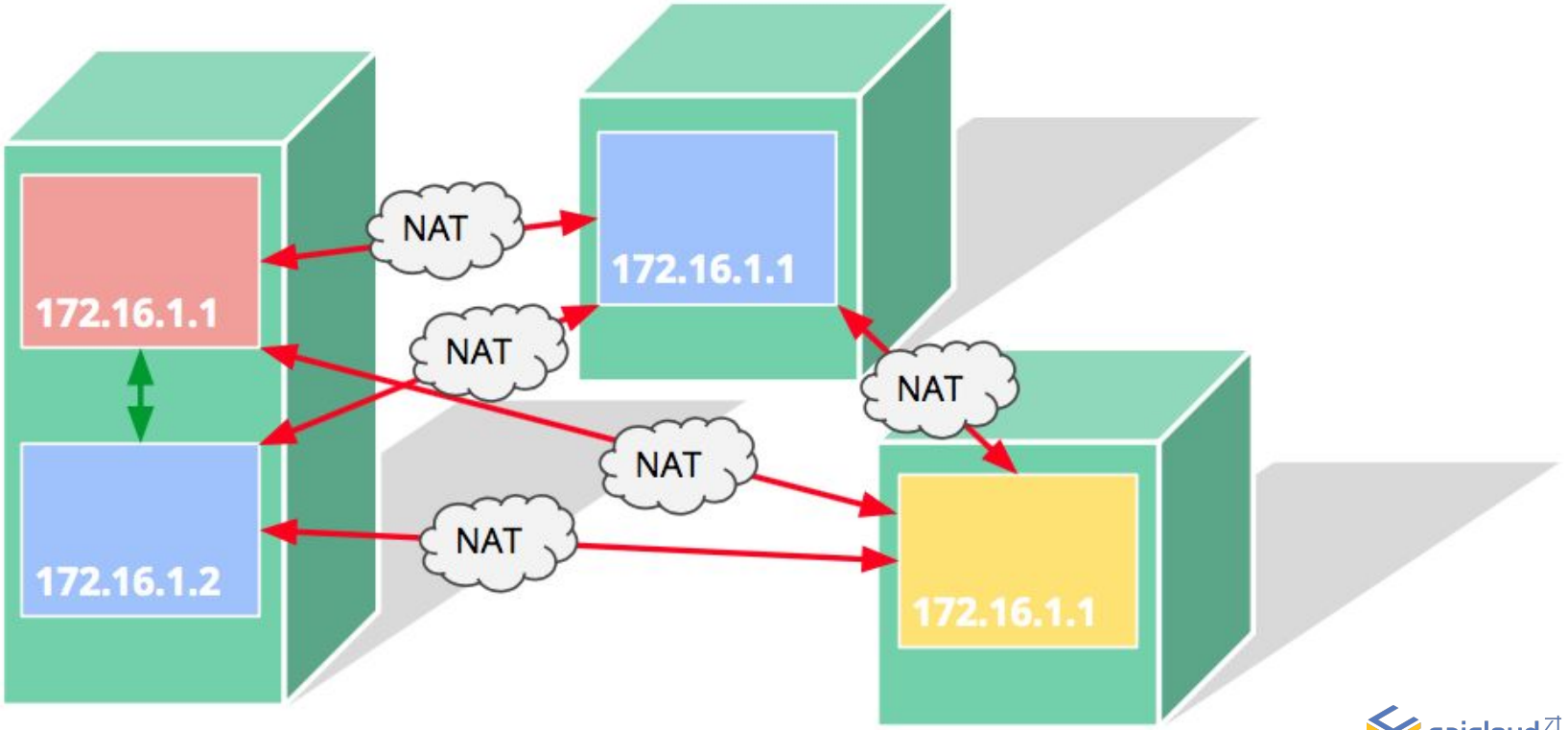


Docker Networking

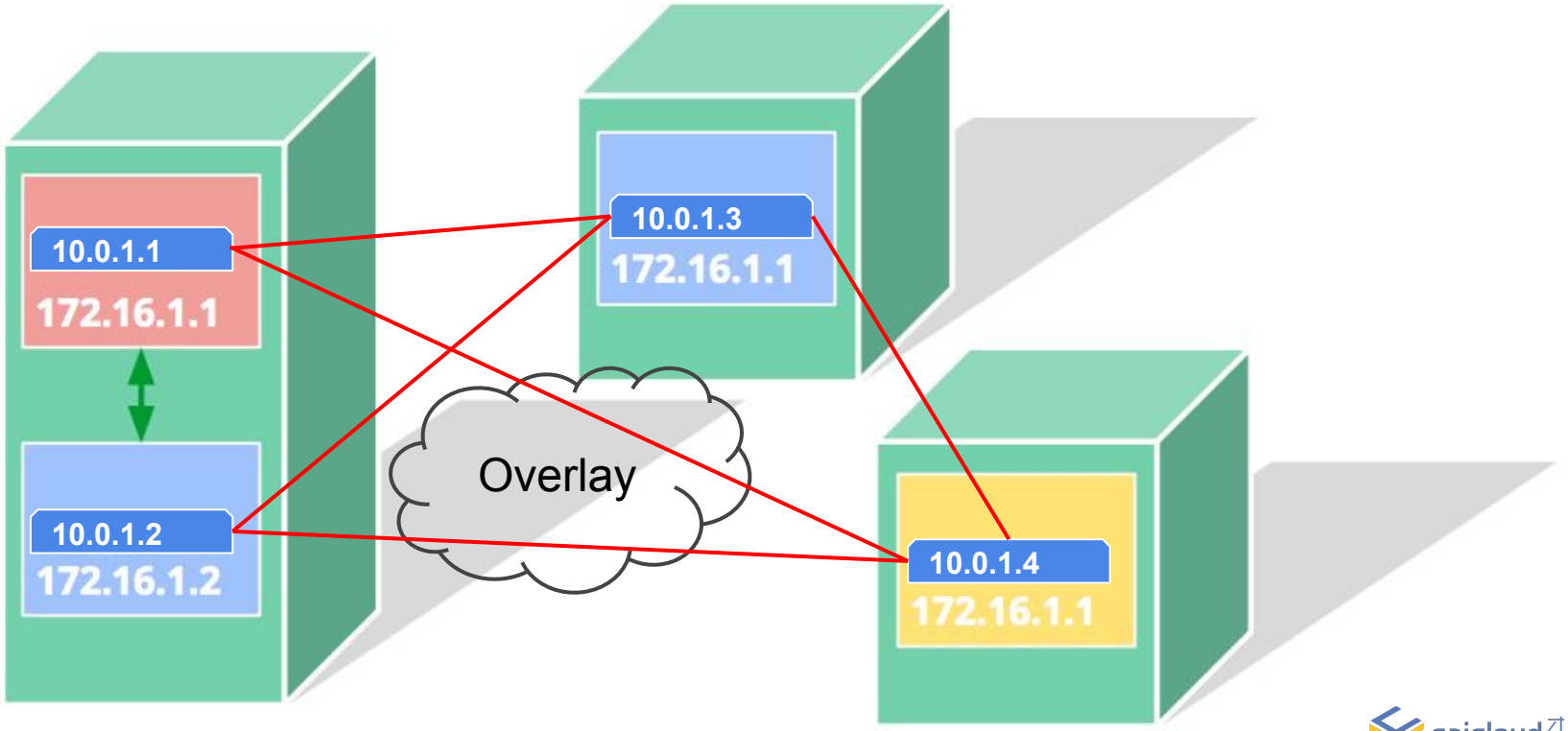


Docker Networking - HostPort

Docker Networking

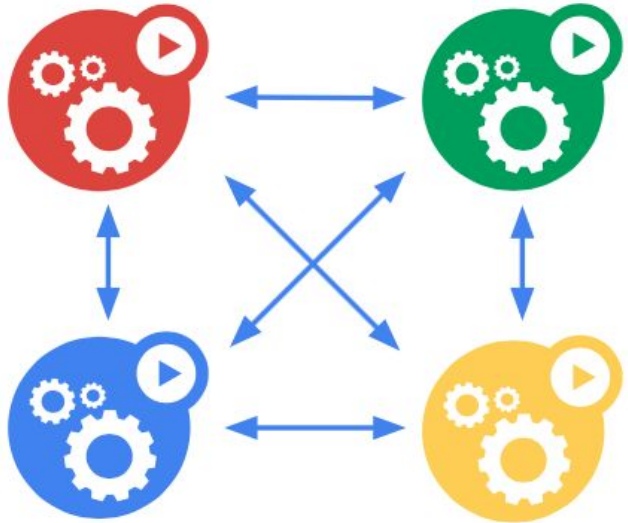


Docker Networking - Overlay

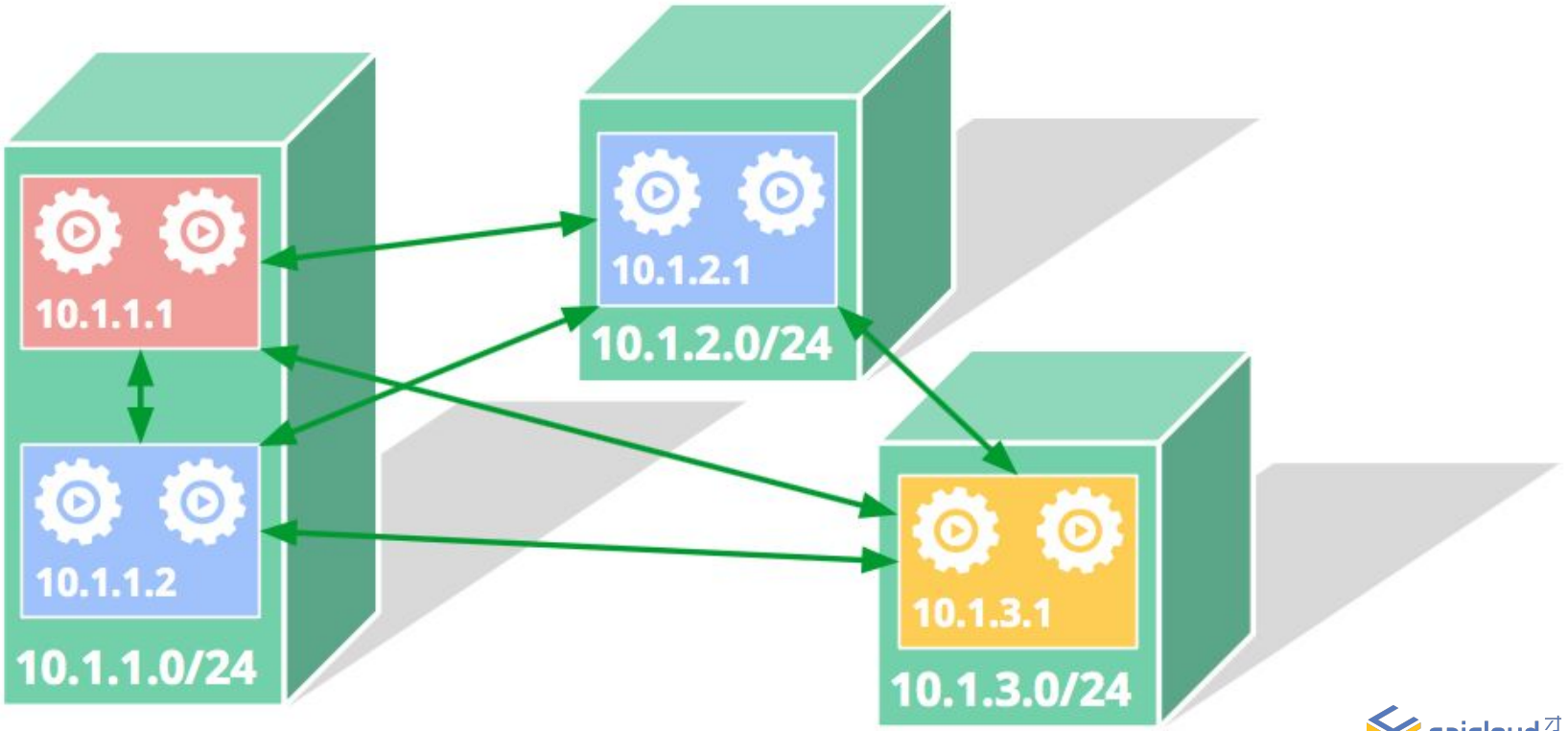


Kubernetes Networking

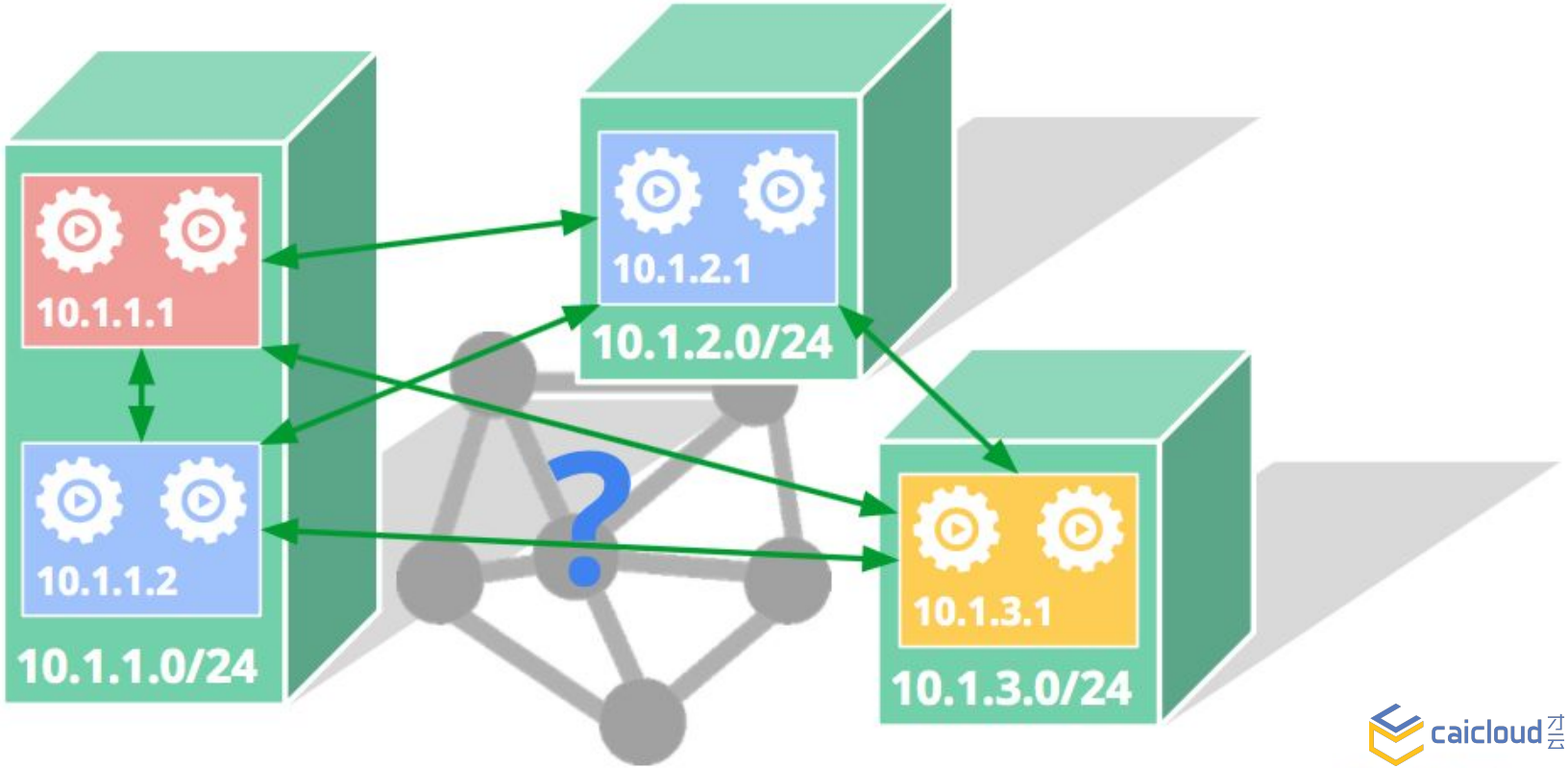
- IPs are routable
  - vs docker default private IP
- Pods can reach each other without NAT
  - even across nodes
- No brokering of port numbers
  - Too complex, why bother?
- Every host is assigned a subnet
- This is a fundamental requirement



Kubernetes Networking



Kubernetes Networking

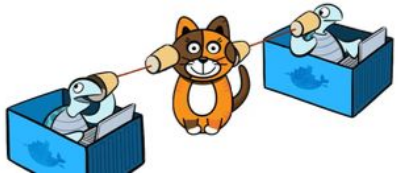


Kubernetes Networking

- On GCE/GKE
  - GCE advanced routes (program the fabric)
  - Everything to “10.1.1.0/24”, send to this VM
- On AWS
  - Route table
- Plenty of other ways
  - Weave
  - Calico
  - Flannel
  - OVS
  - OpenContrail
  - Cisco Contiv
  - Canal
  - Others
  - ...

Google Cloud Platform

weave



CISCO

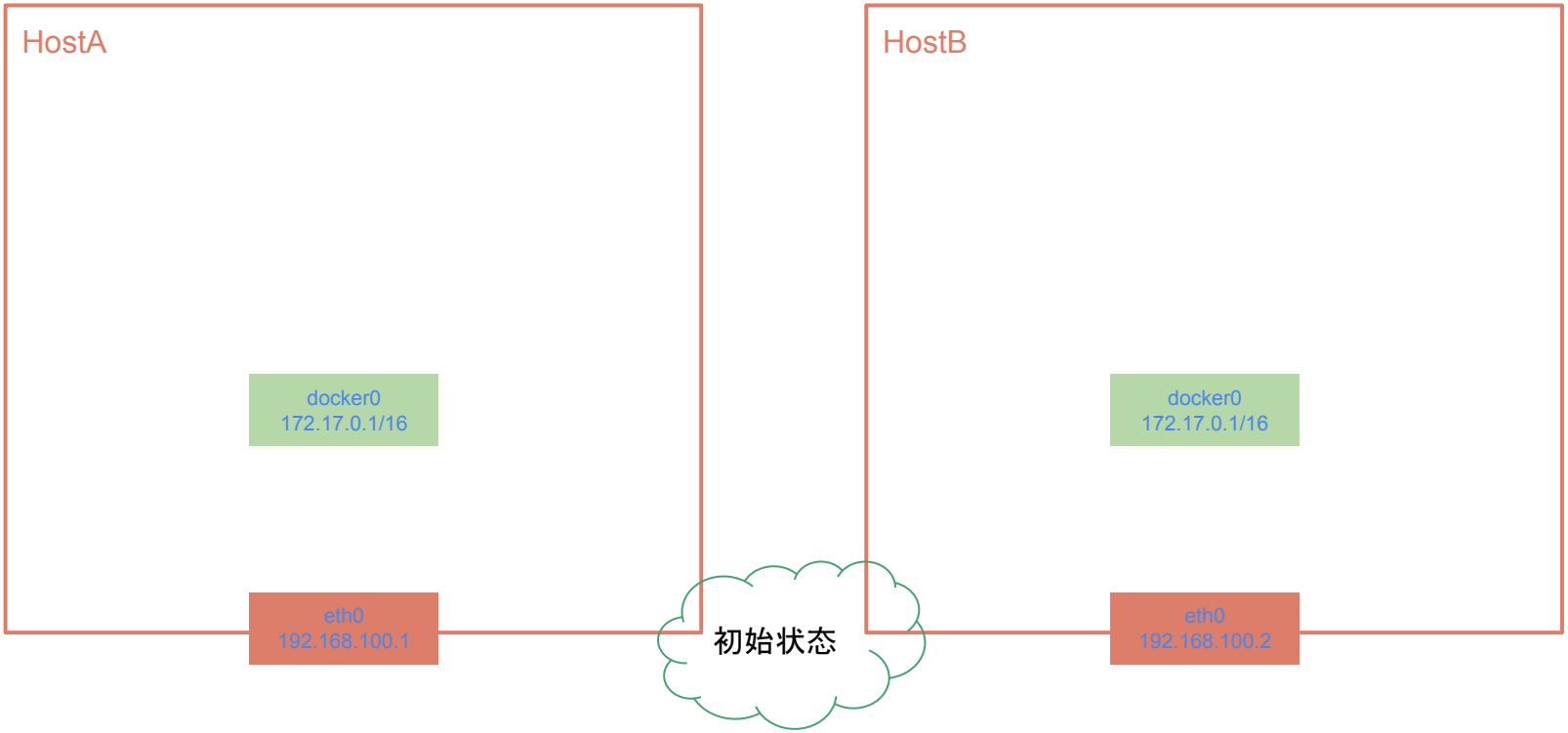
OvS
Open vSwitch

OPENCONTRAIL

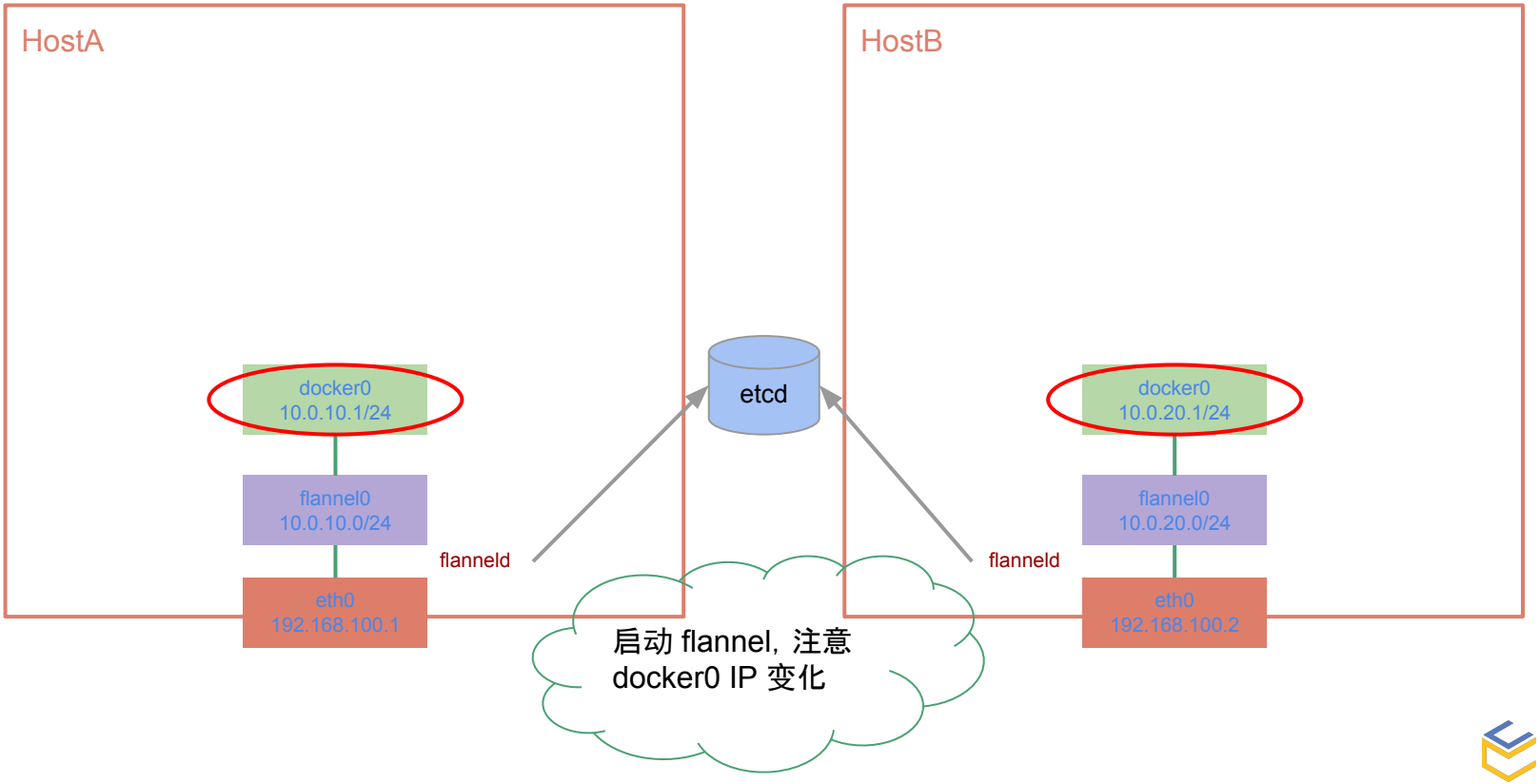
Flannel (https://github.com/coreos/flannel)

- 解决 docker 容器直接跨主机的通信问题
- 为每一个主机分配一个 IP 段，然后主机上的容器被分配该 IP 网段不同的 IP
- 所有的容器在一个虚拟的大二层网络，底层通过 UDP/VxLAN 等进行报文的封装和转发

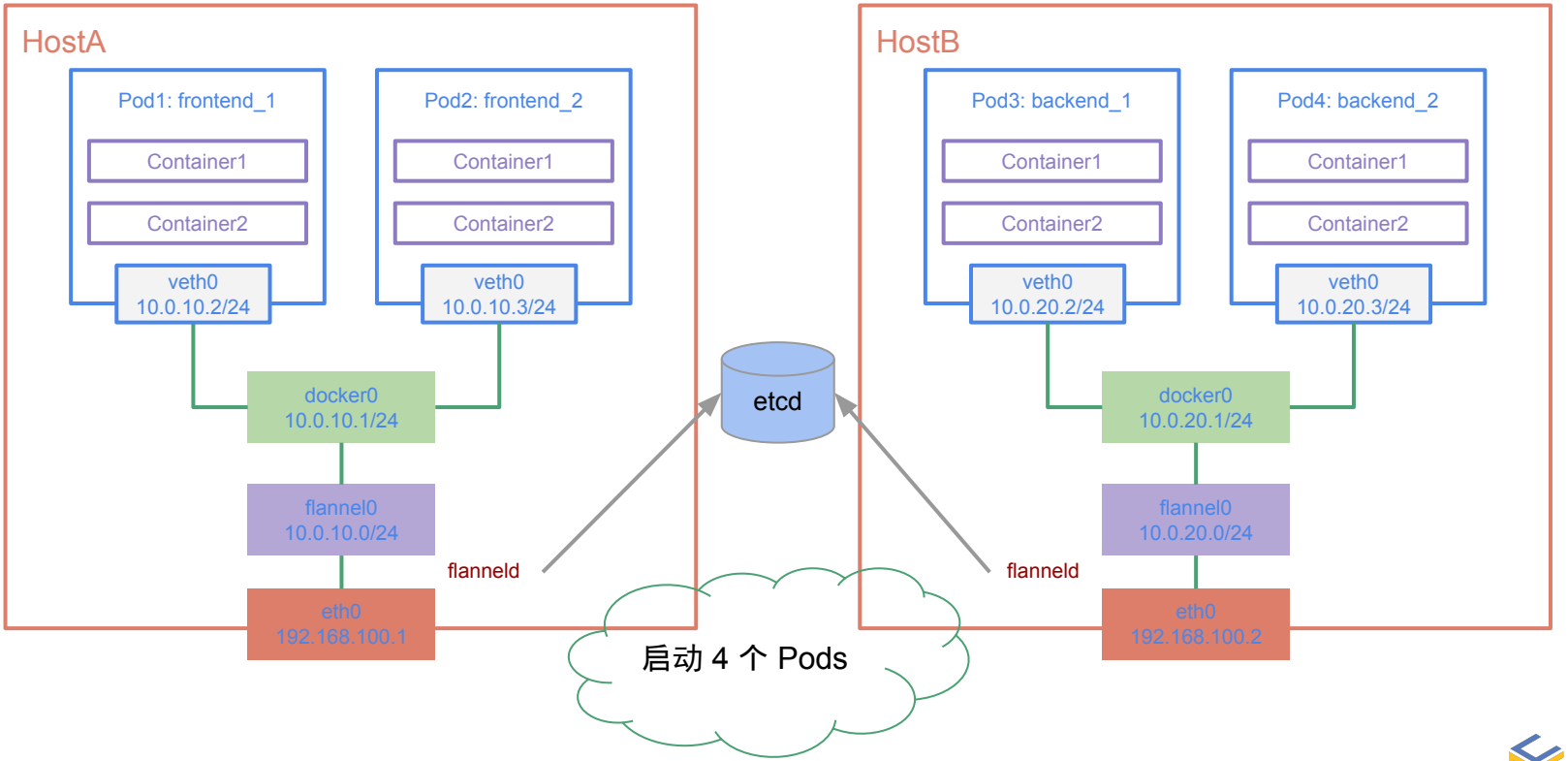
Flannel



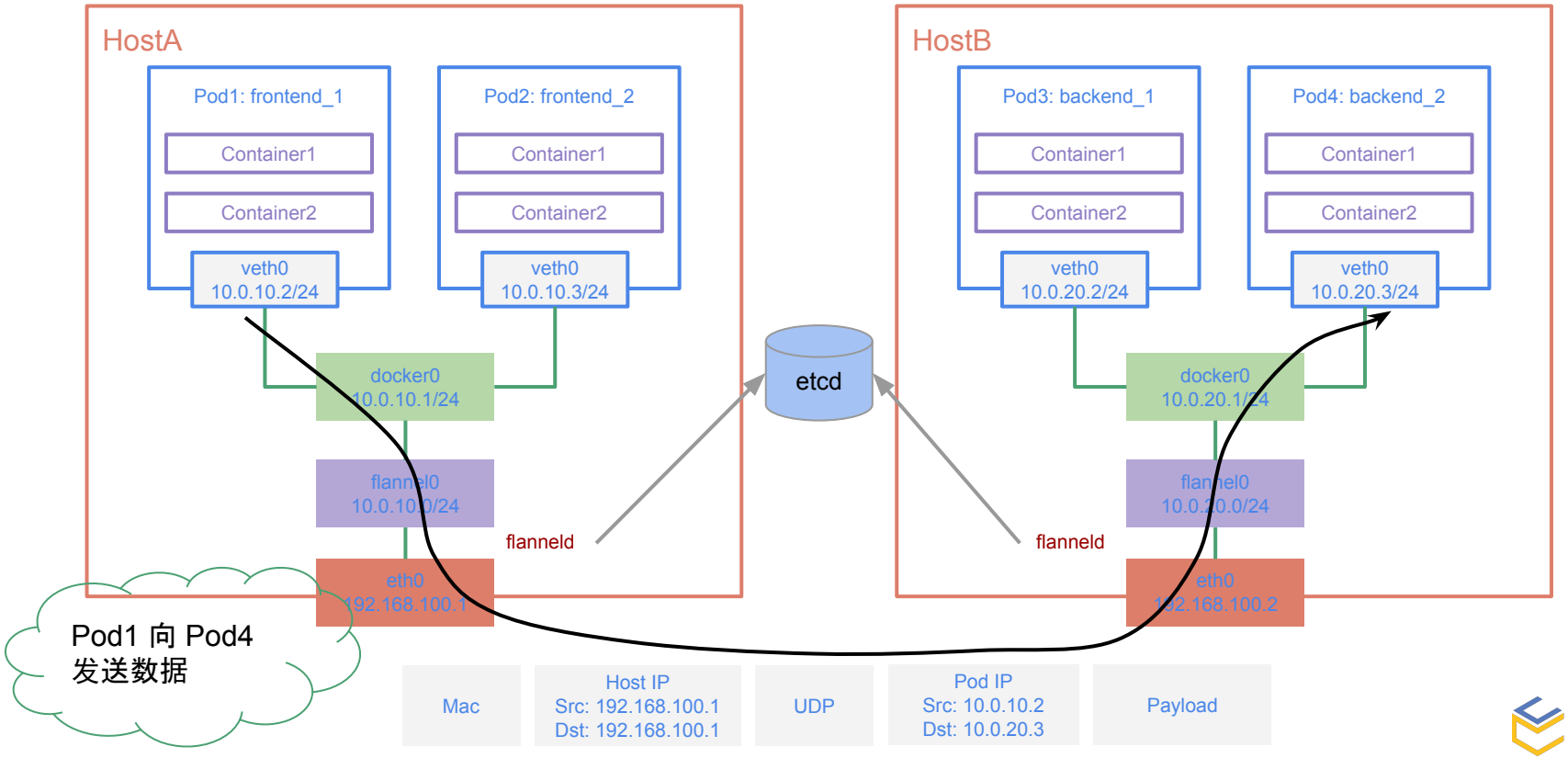
Flannel



Flannel



Flannel

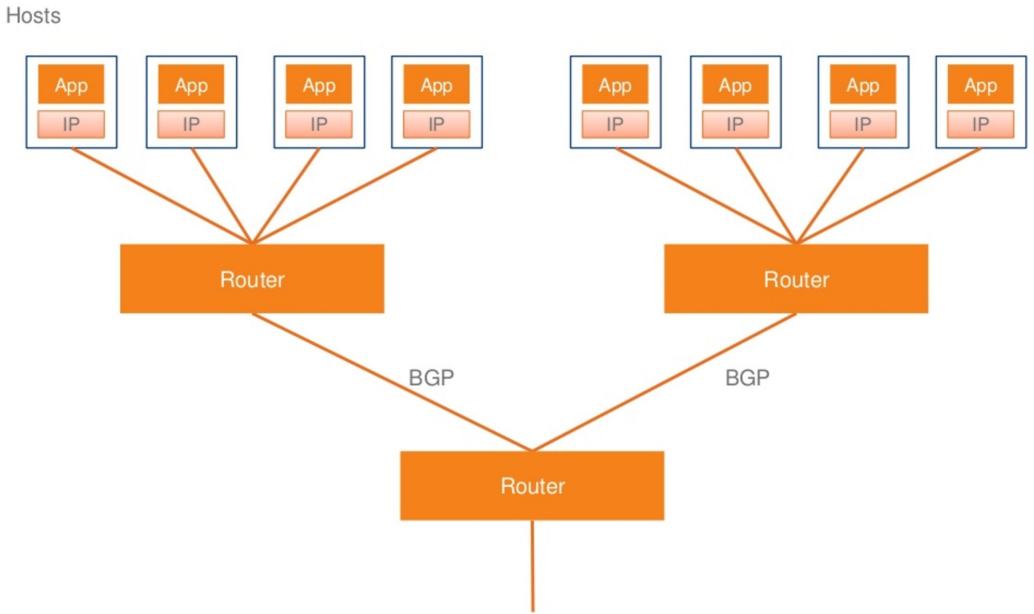


Calico (https://github.com/projectcalico)

Build a container network like the internet?

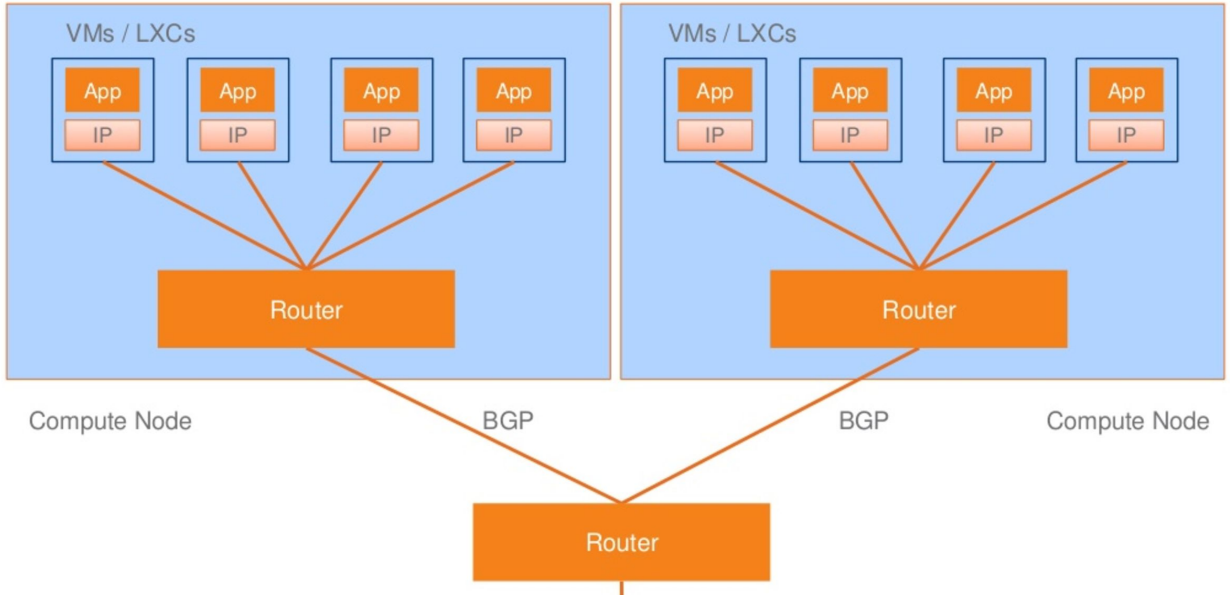
Calico

Build a container network like the internet ?



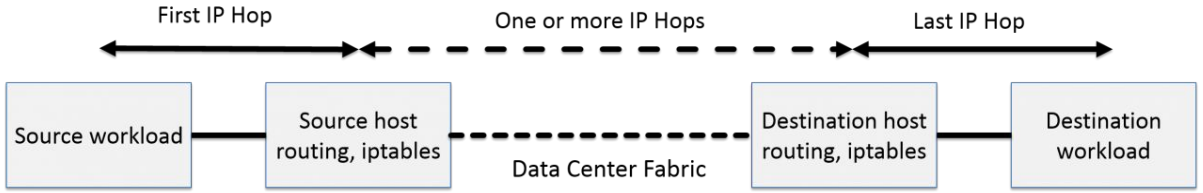
Calico

Build a container network like the internet !

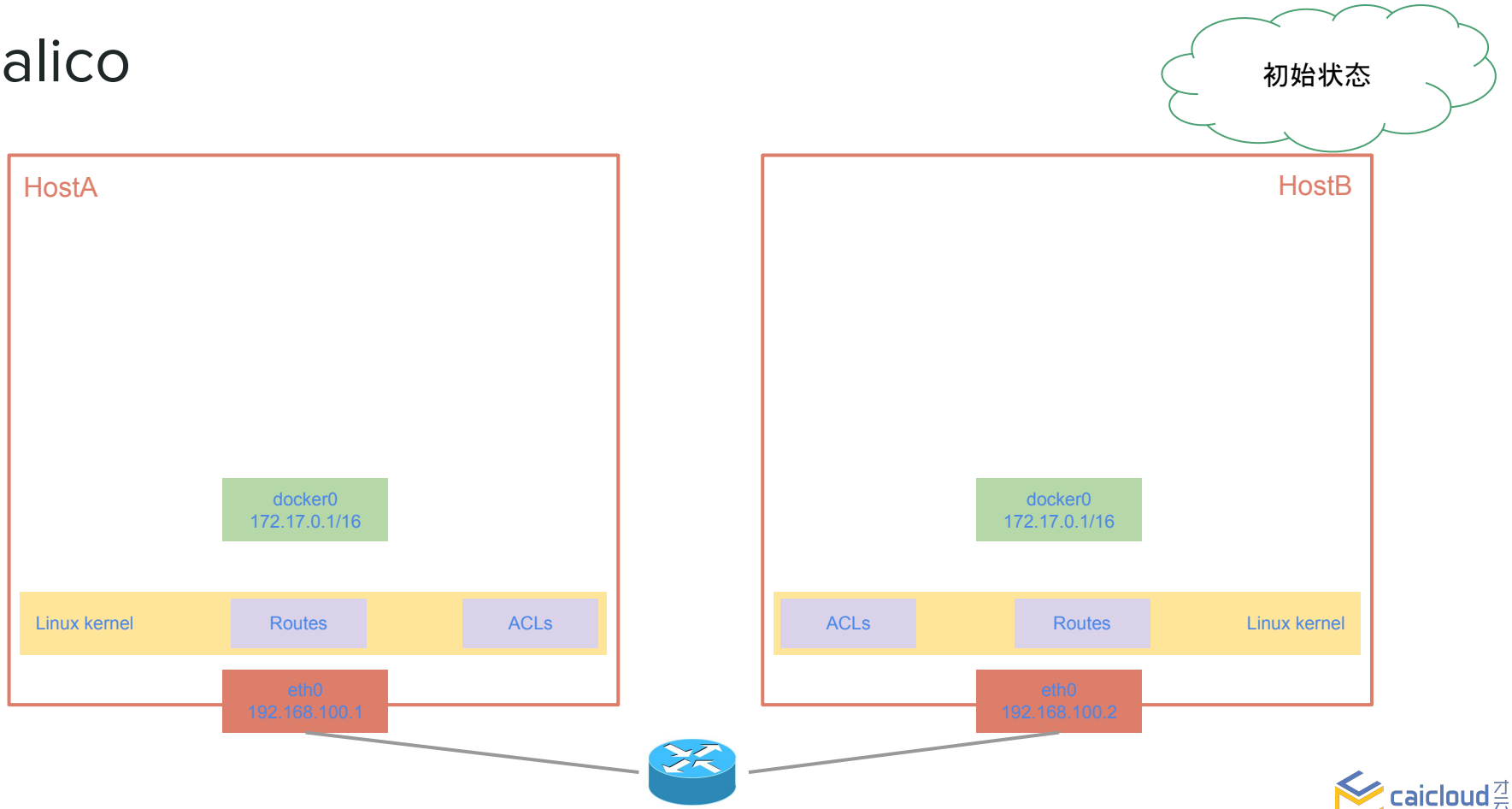


Calico

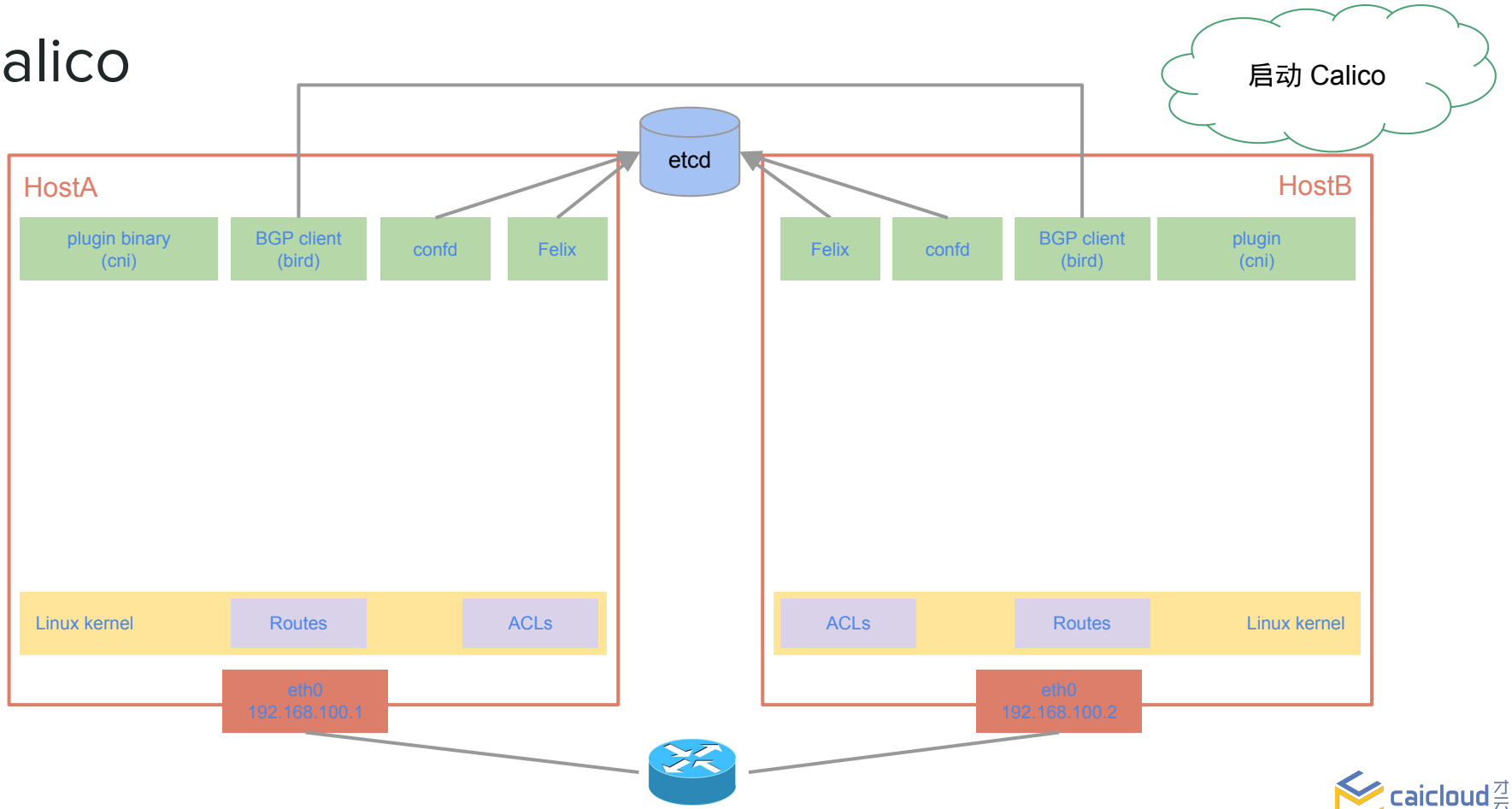
Build a container network like the internet !



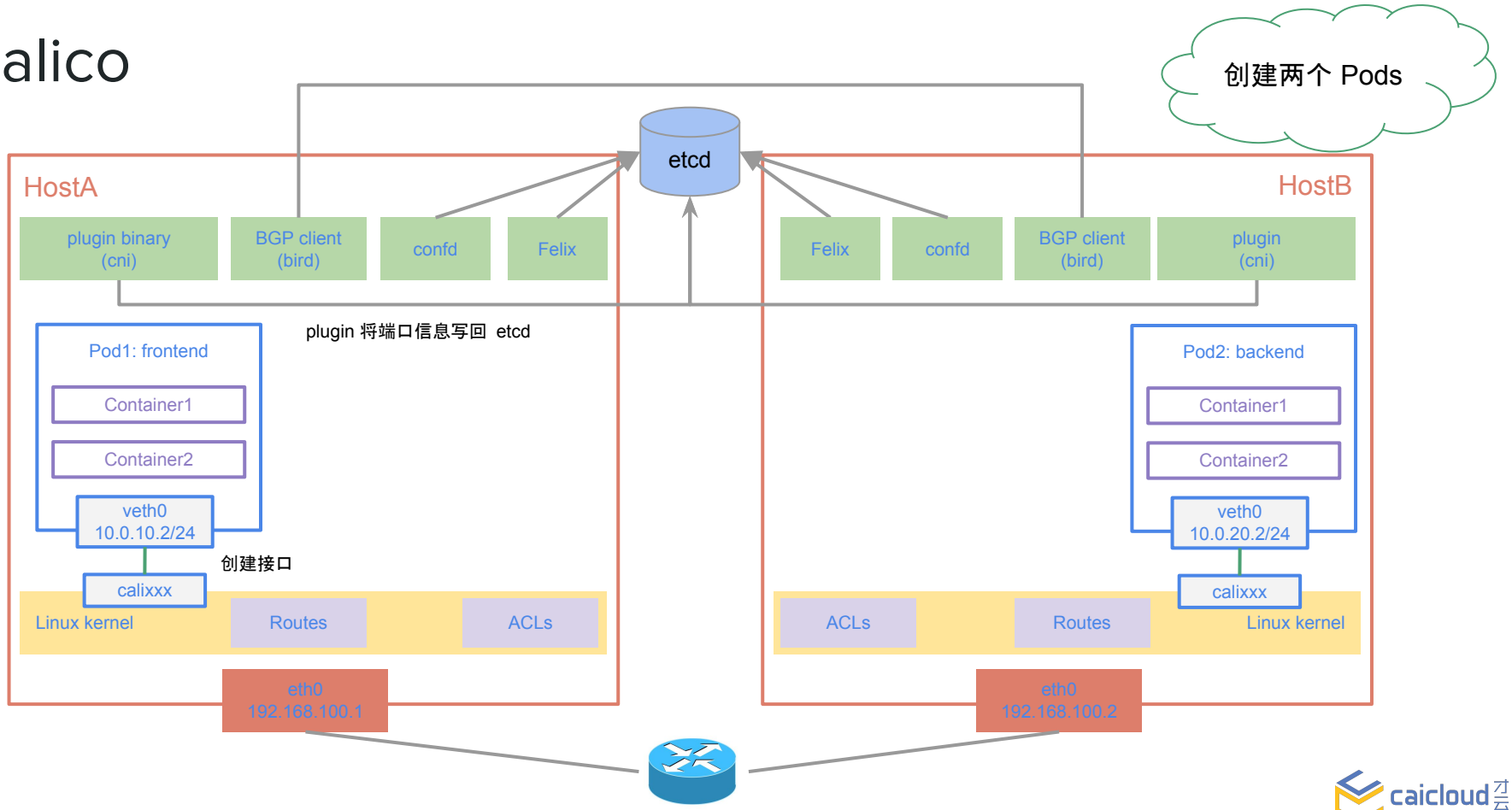
Calico



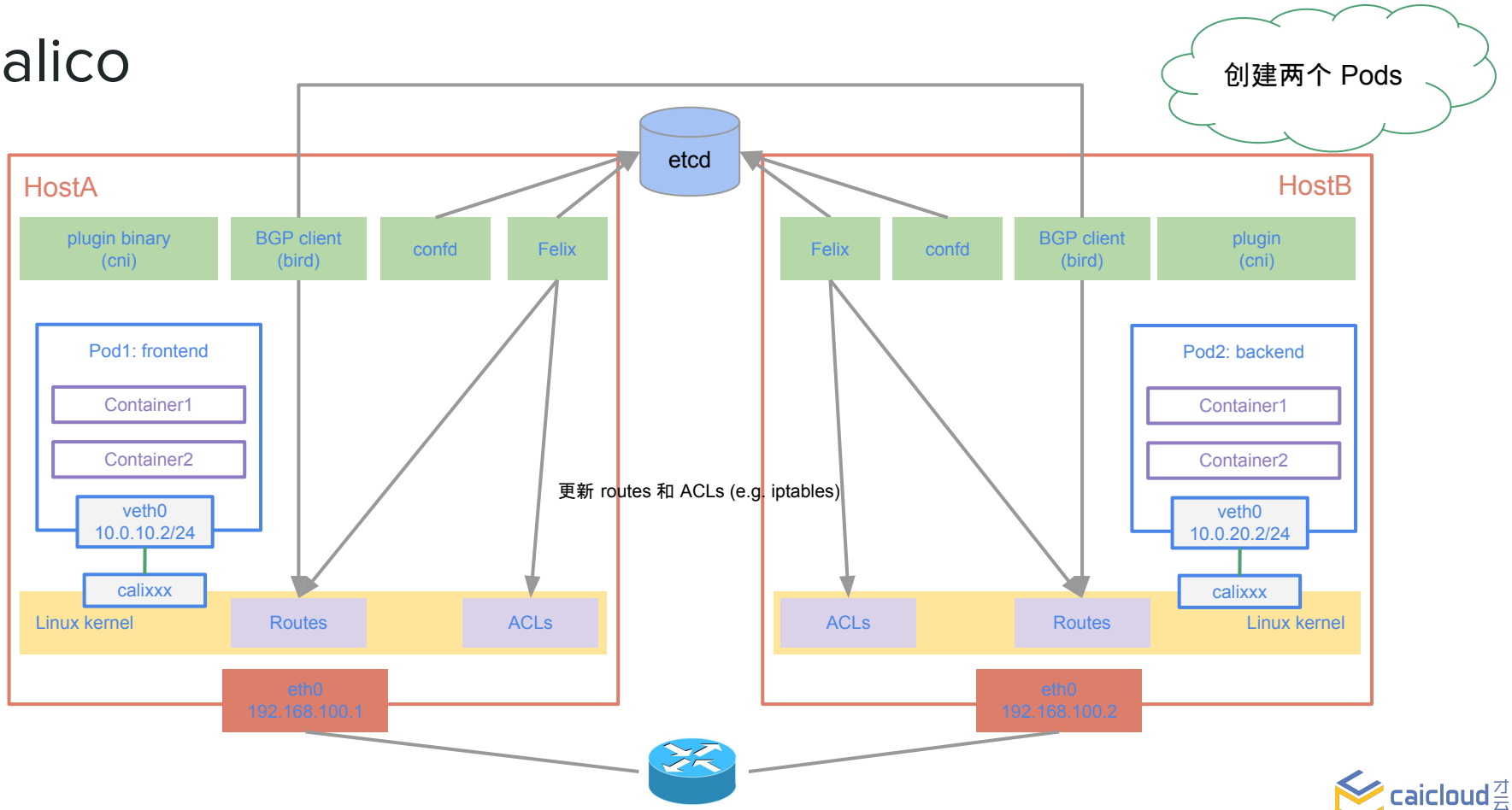
Calico



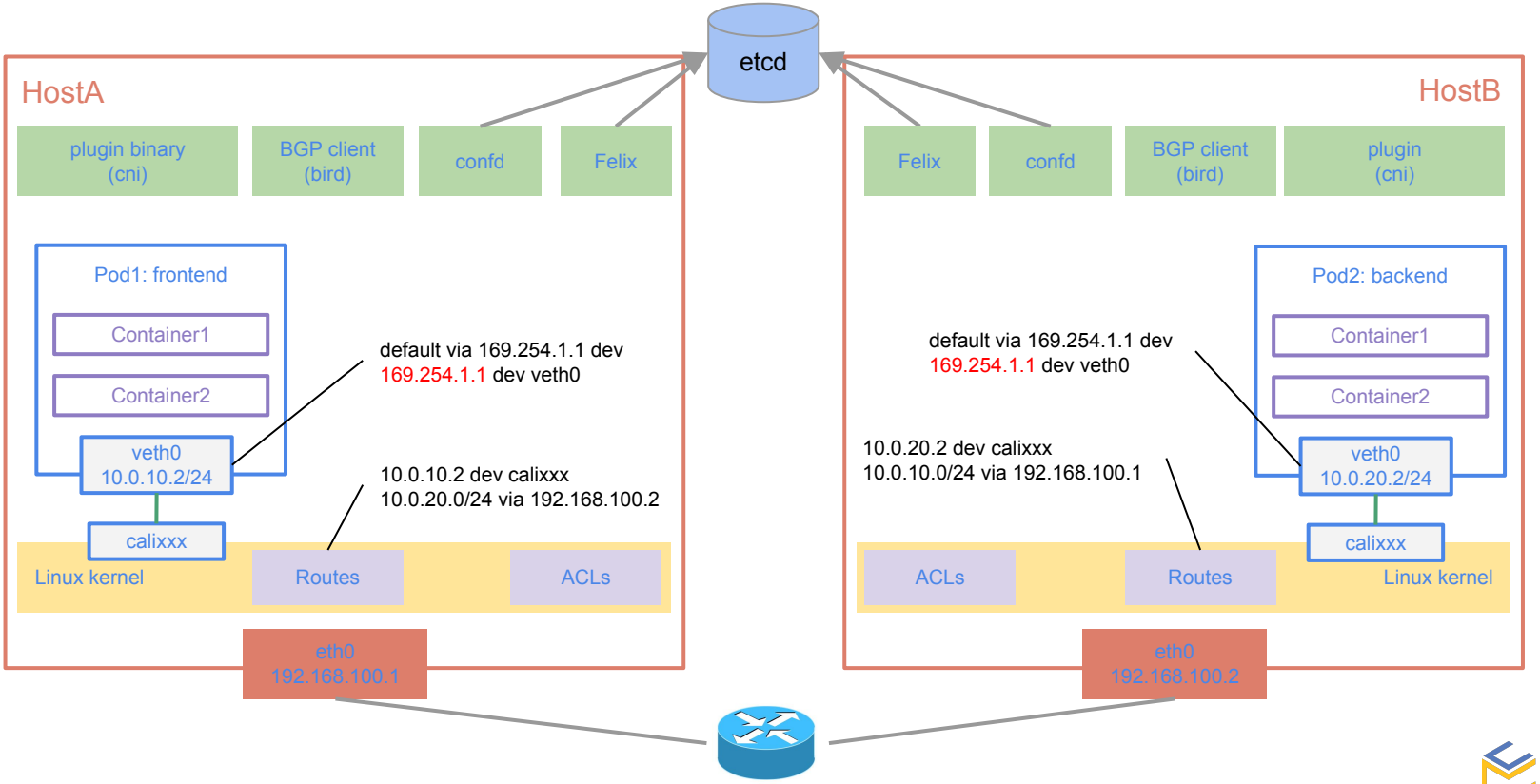
Calico


创建两个 Pods
etcd
HostA
HostB
plugin binary (cni)
BGP client (bird)
confd
Felix
Felix
confd
BGP client (bird)
plugin (cni)
plugin 将端口信息写回 etcd
Pod1: frontend
Container1
Container2
veth0 10.0.10.2/24
创建接口
calixxx
Linux kernel
Routes
ACLs
ACLs
Routes
Linux kernel
calixxx
veth0 10.0.20.2/24
Pod2: backend
Container1
Container2
eth0 192.168.100.1
eth0 192.168.100.2

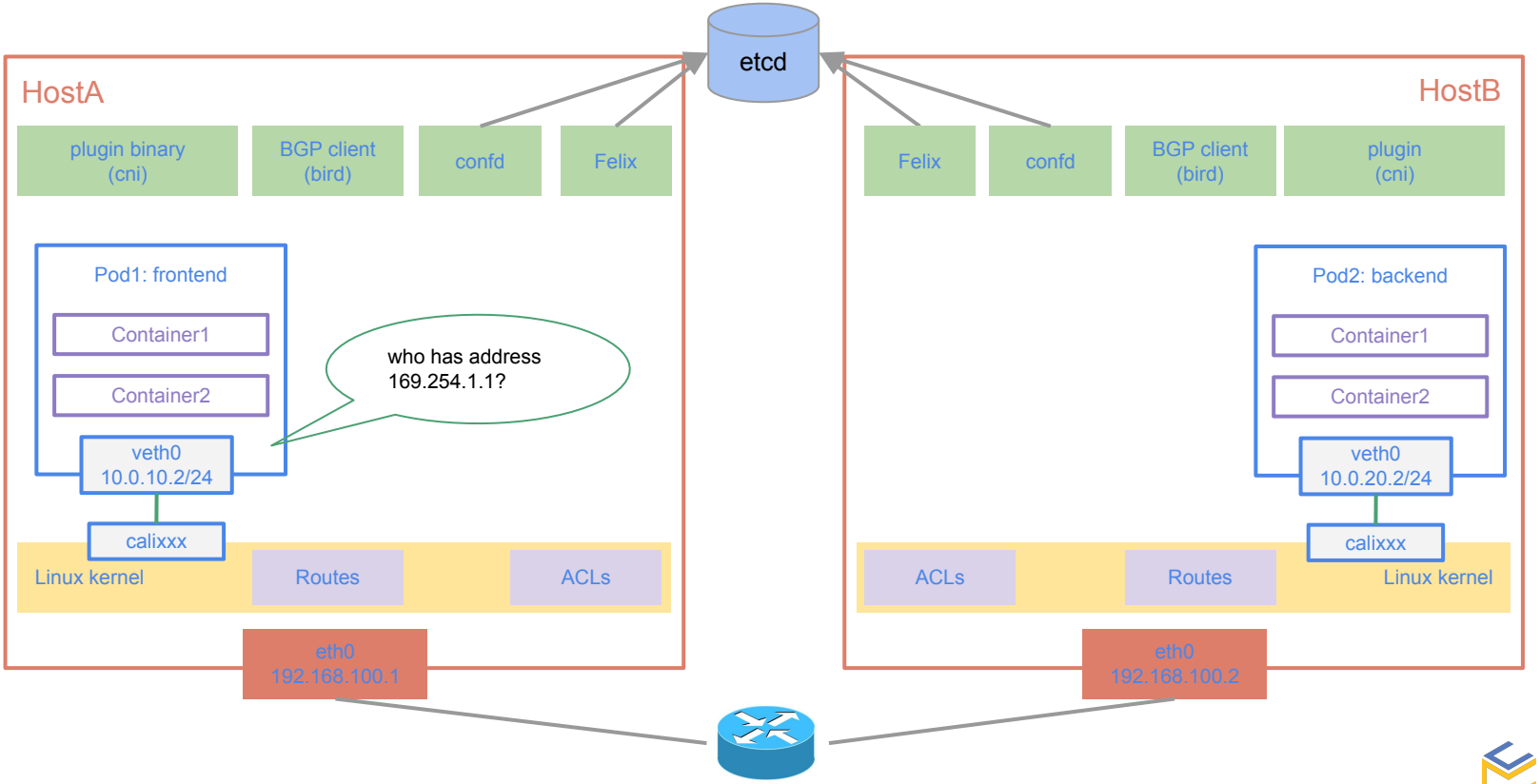
Calico



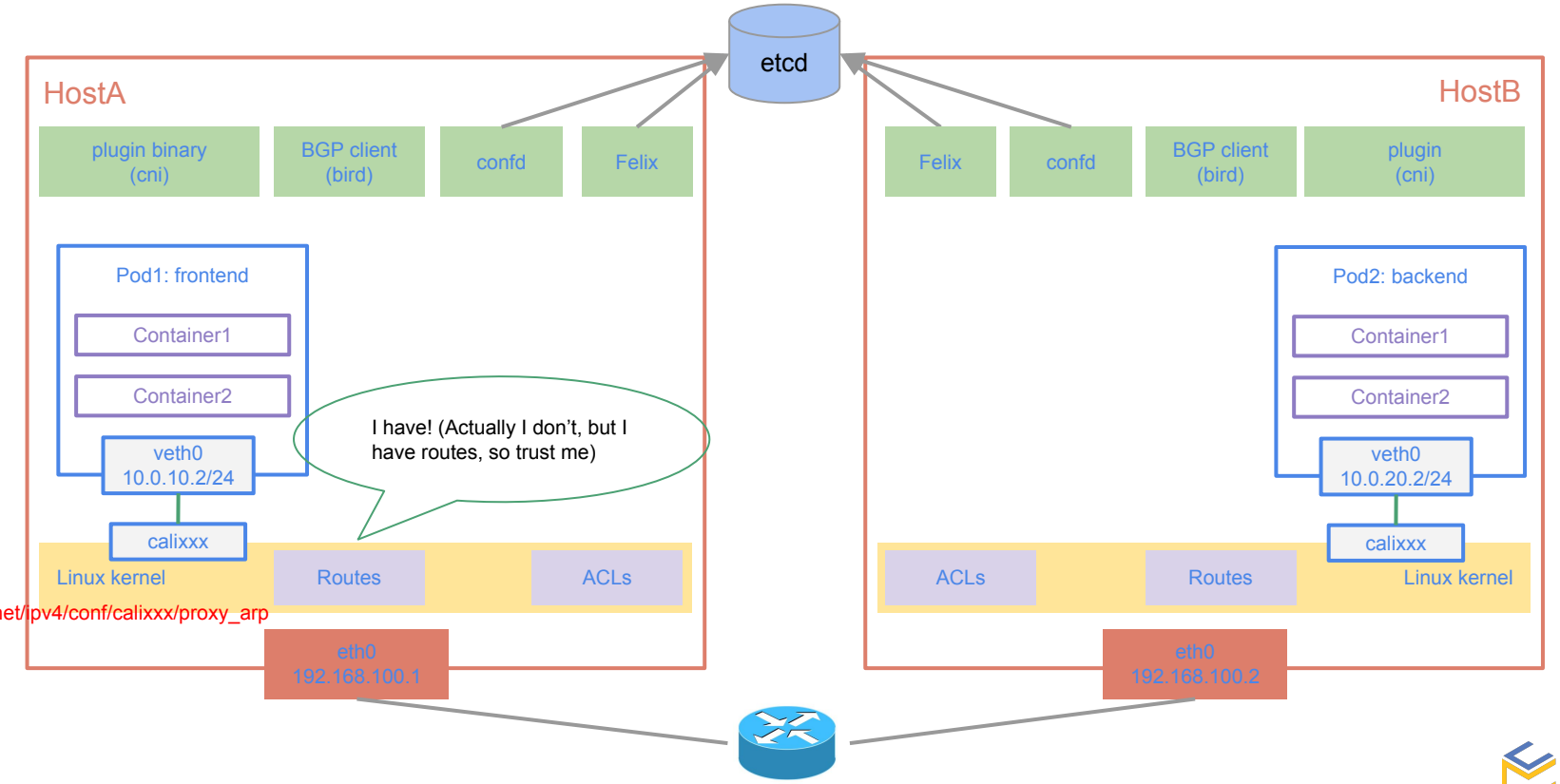
Calico



Calico



Calico

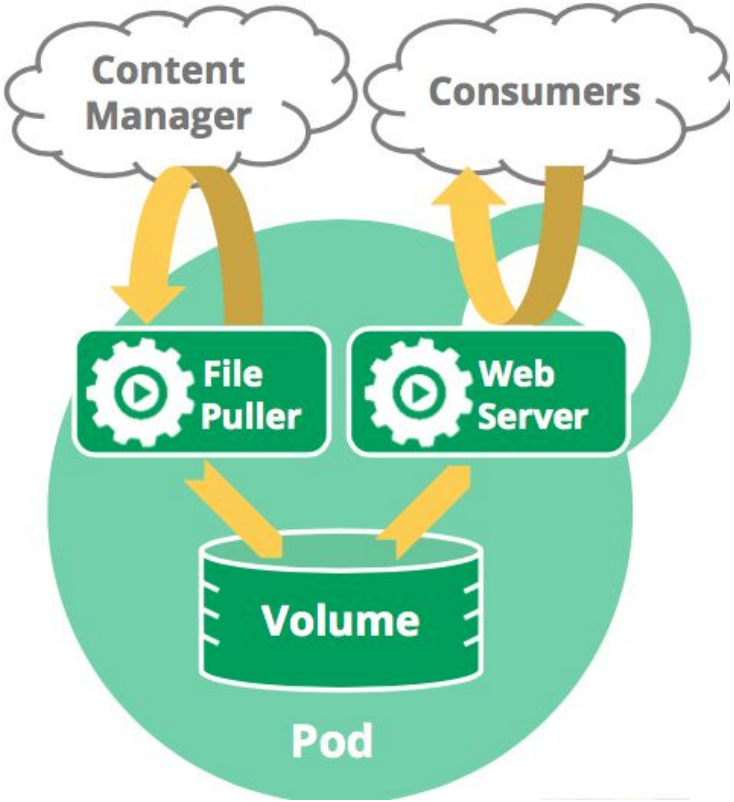

etcd
HostA
HostB
plugin binary (cni)
BGP client (bird)
confd
Felix
Felix
confd
BGP client (bird)
plugin (cni)
Pod1: frontend
Container1
Container2
veth0 10.0.10.2/24
calixxx
I have! (Actually I don't, but I have routes, so trust me)
Linux kernel
Routes
ACLs
ACLs
Routes
Linux kernel
Pod2: backend
Container1
Container2
veth0 10.0.20.2/24
calixxx
/proc/sys/net/ipv4/conf/calixxx/proxy_arp
eth0 192.168.100.1
eth0 192.168.100.2

Kubernetes Networking - Pod

Pod

- Small group of containers & volumes
- Tightly coupled
- The atom of scheduling & placement
- Shared namespace
  - shared IP address & localhost
  - shared IPC, etc
- Managed lifecycle
  - bound to a node, restart in place
  - Can die, cannot be reborn with the same ID

Example: data puller & web server



Pod

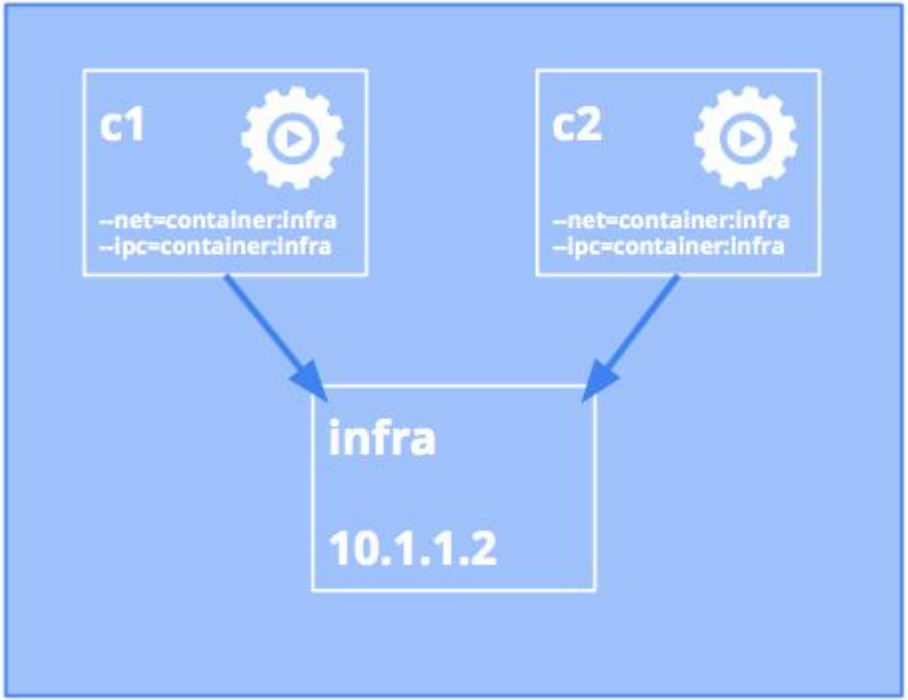
- **Small group of containers & volumes**
- **Tightly coupled**
- The atom of scheduling & placement
- Shared namespace
  - shared IP address & localhost
  - shared IPC, etc
- Managed lifecycle
  - bound to a node, restart in place
  - Can die, cannot be reborn with the same ID

Example: data puller & web server

Pod

- Small group of containers & volumes
- Tightly coupled
- The atom of scheduling & placement
- **Shared namespace**
  - **shared IP address & localhost**
  - **shared IPC, etc**
- Managed lifecycle
  - bound to a node, restart in place
  - Can die, cannot be reborn with the same ID

Example: data puller & web server

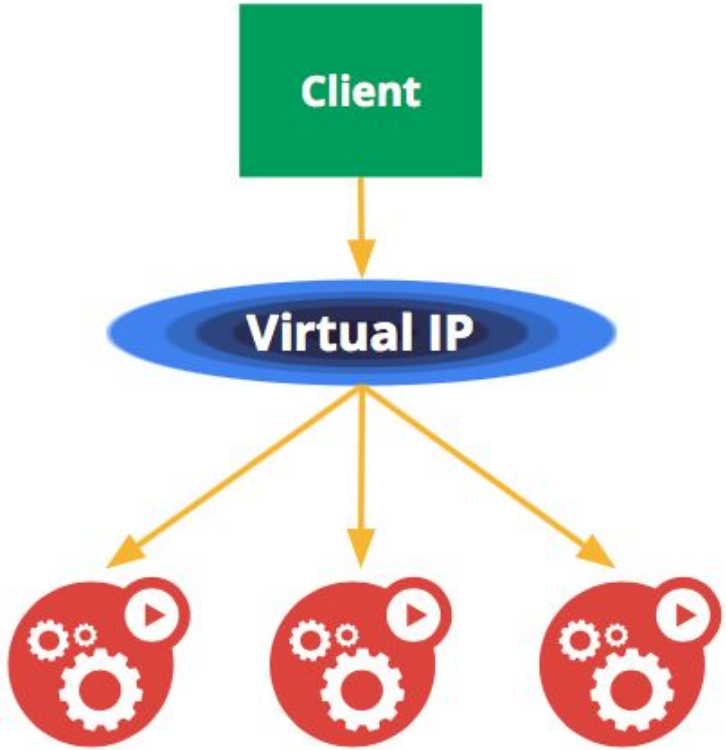


Kubernetes Networking - Service

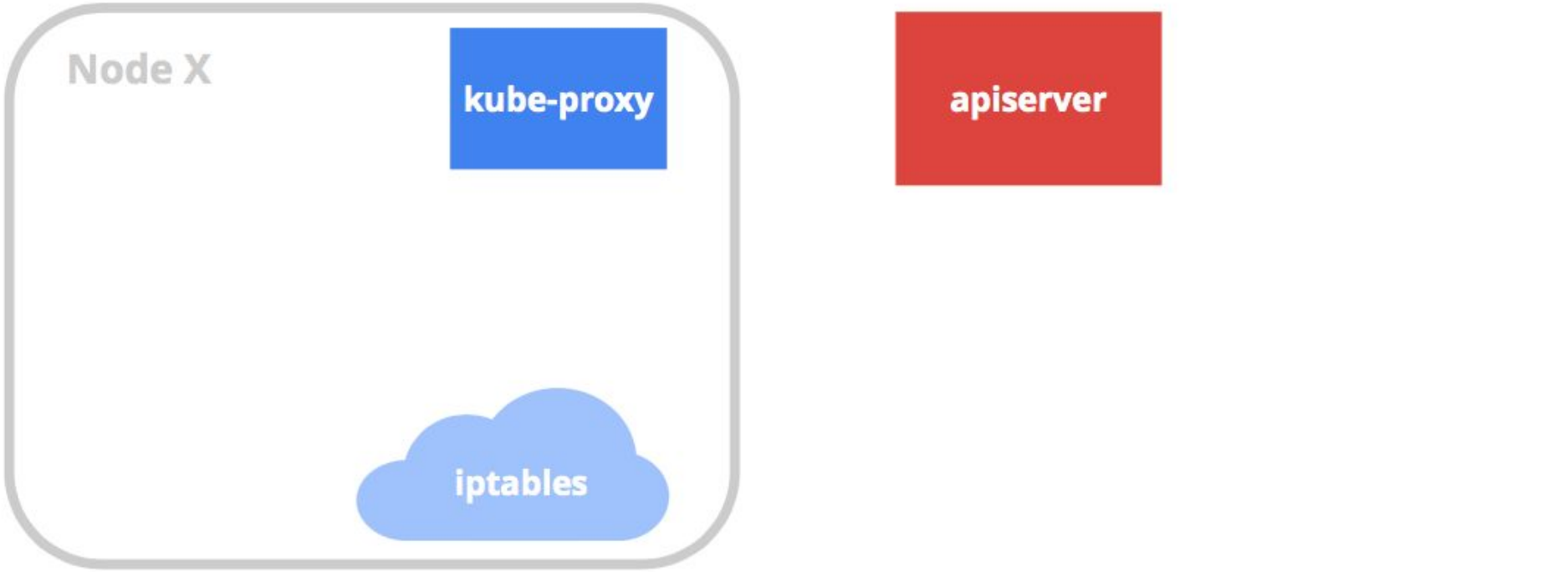
Service

- A group of Pods that **work together**
  - grouped by a selector
- Defines access policy
  - e.g. “load balanced”
- Gets a stable **virtual IP** and port
  - sometimes called the service portal
  - also a DNS name
- VIP is managed by kube-proxy
  - watches all services
  - updates iptables when backends change

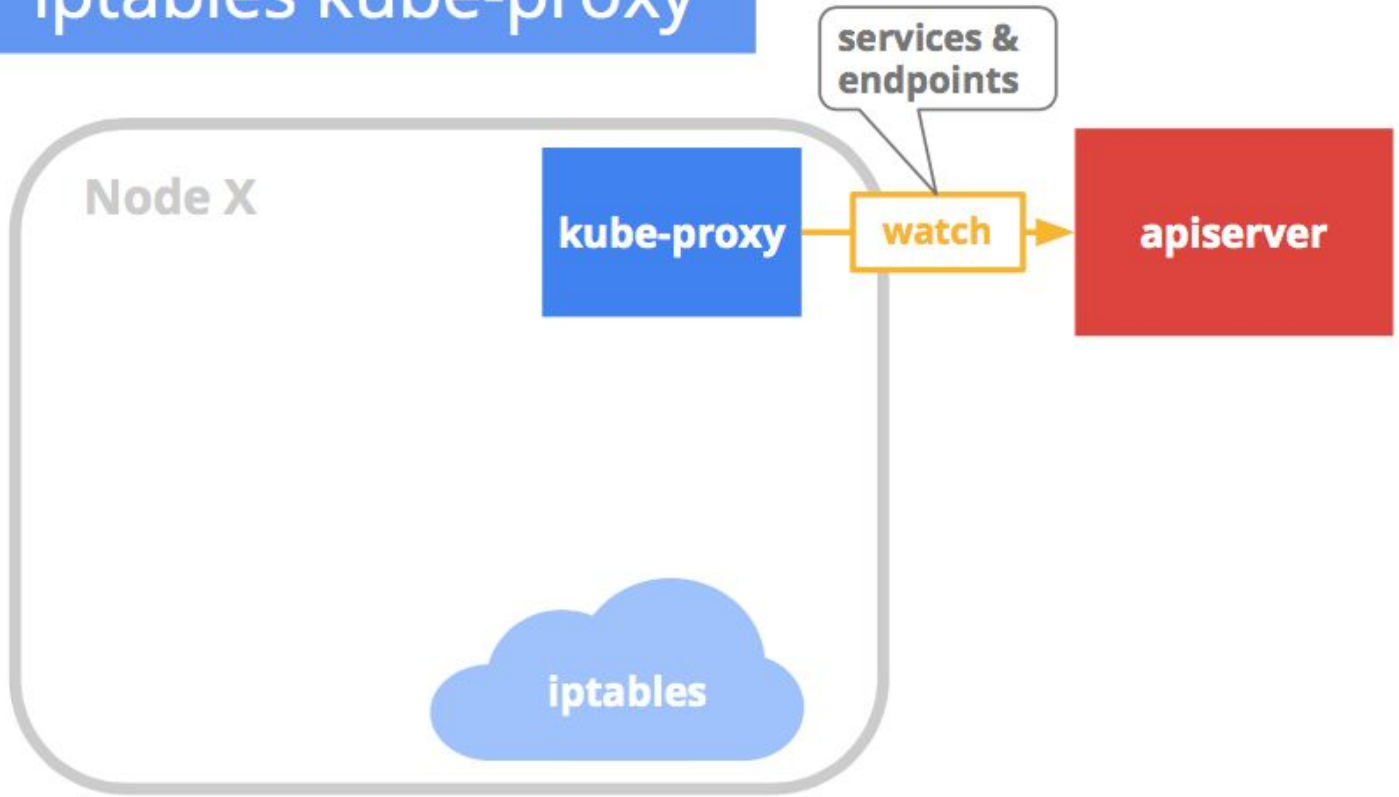
Hides complexity - ideal for non-native apps



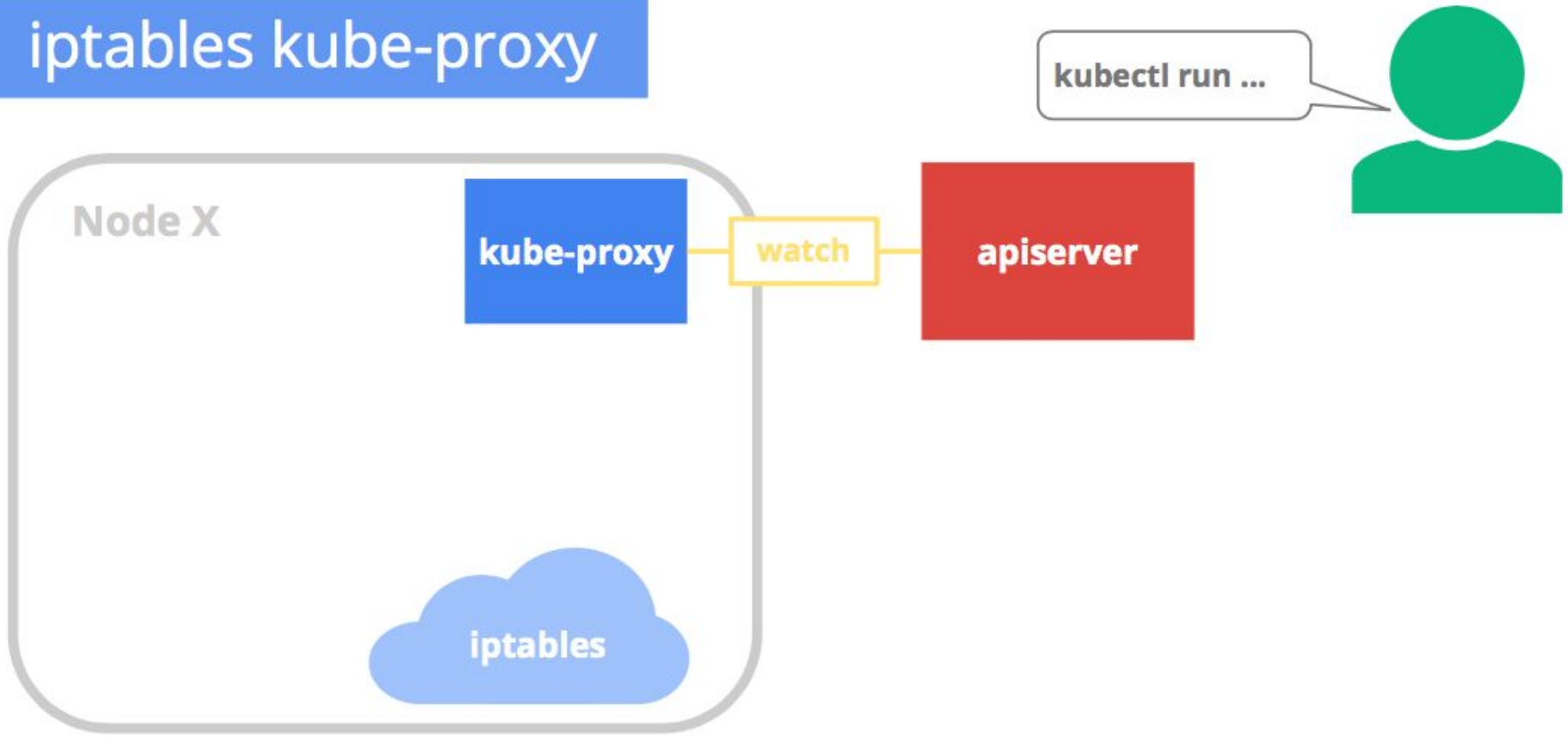
iptables kube-proxy



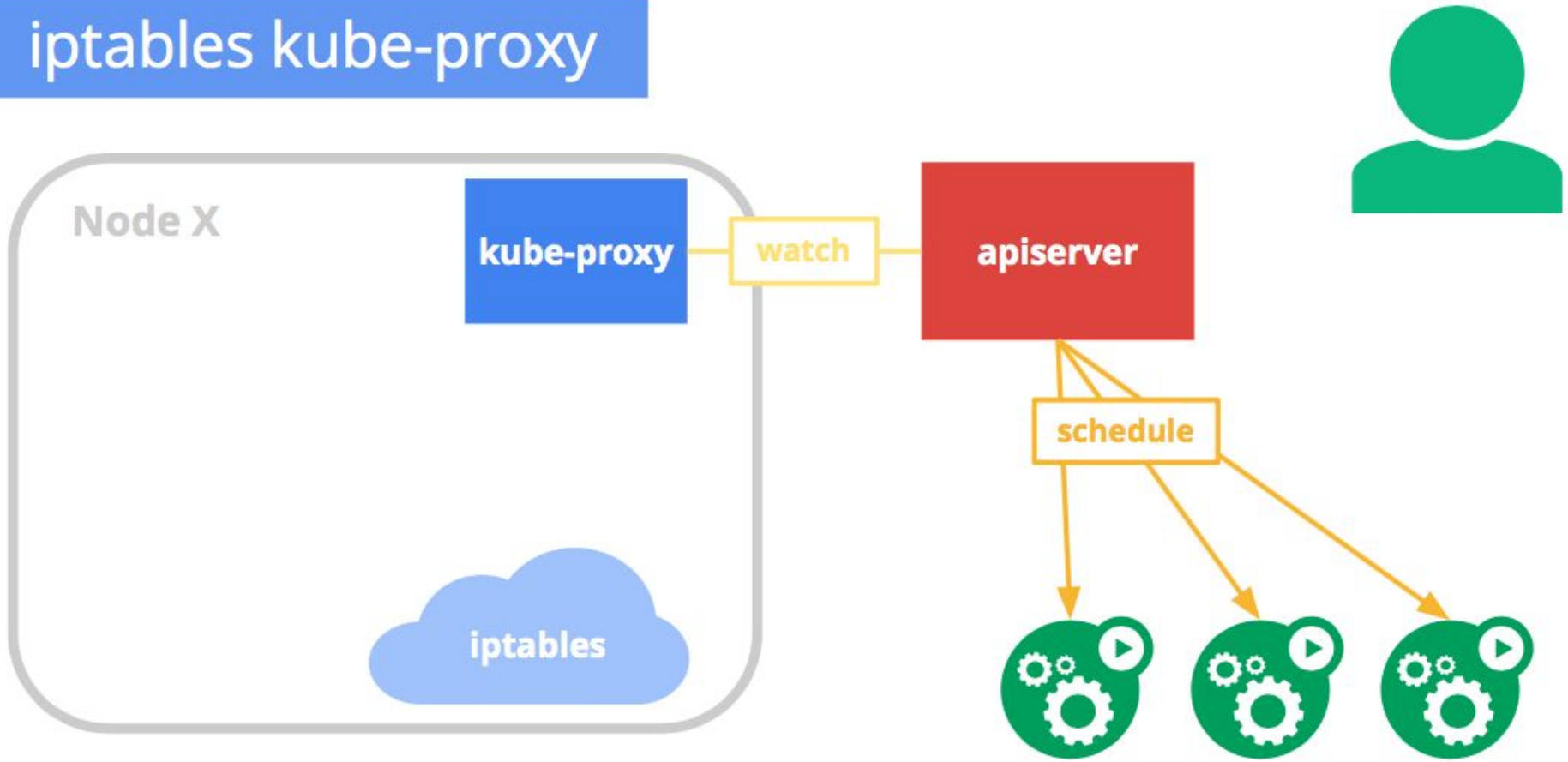
iptables kube-proxy



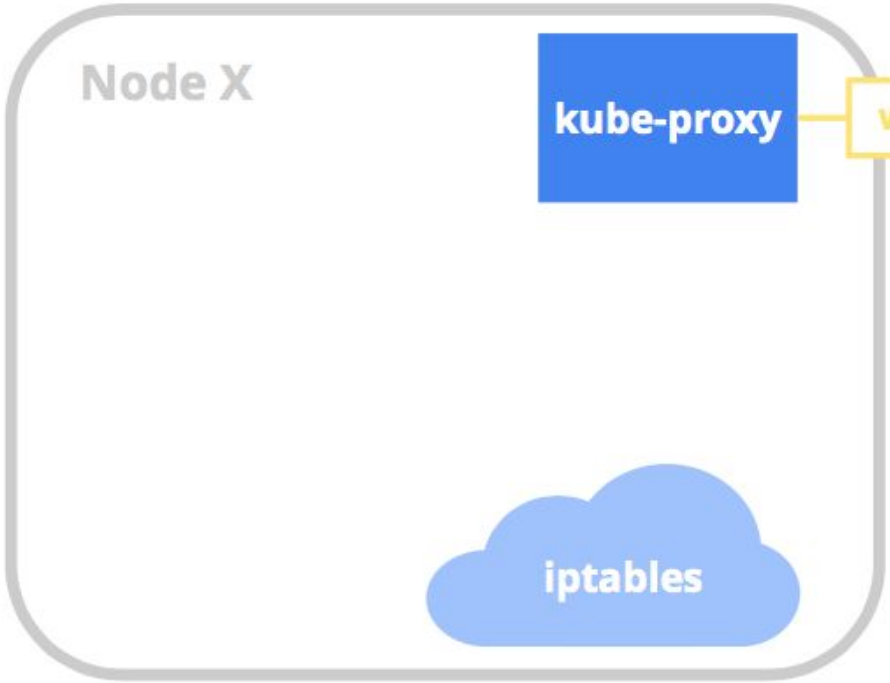
iptables kube-proxy



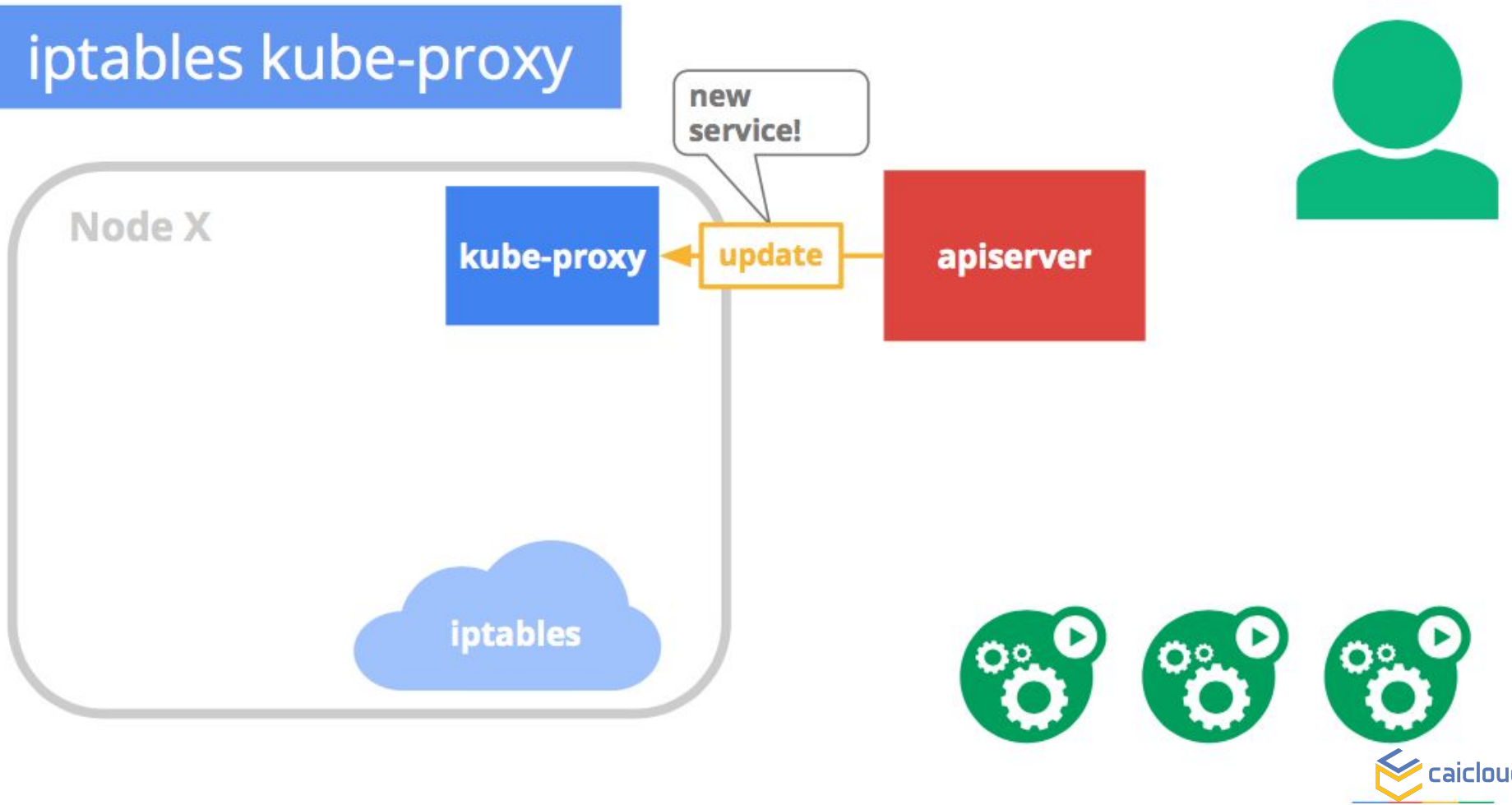
iptables kube-proxy



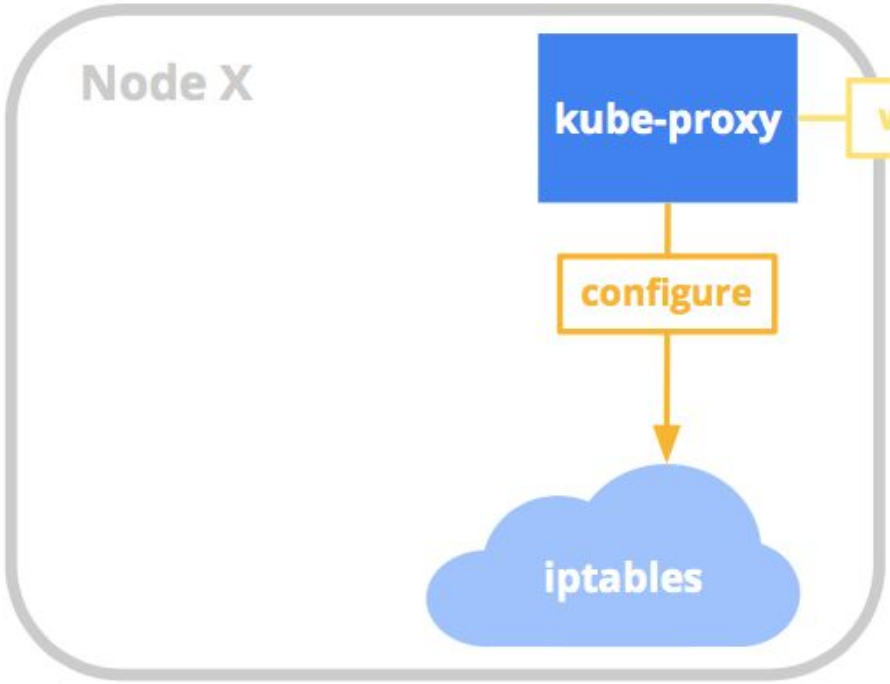
iptables kube-proxy



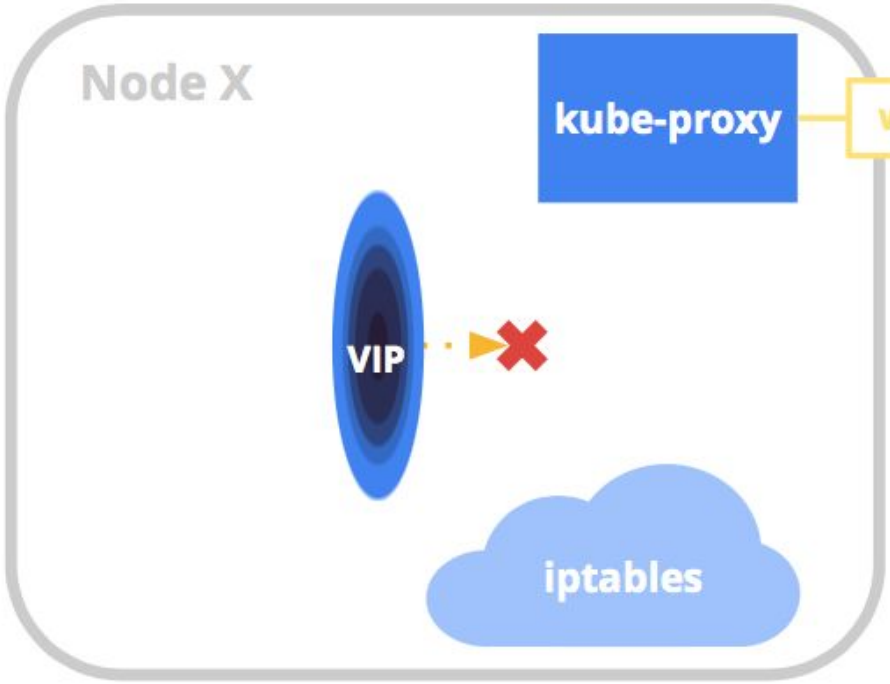
iptables kube-proxy



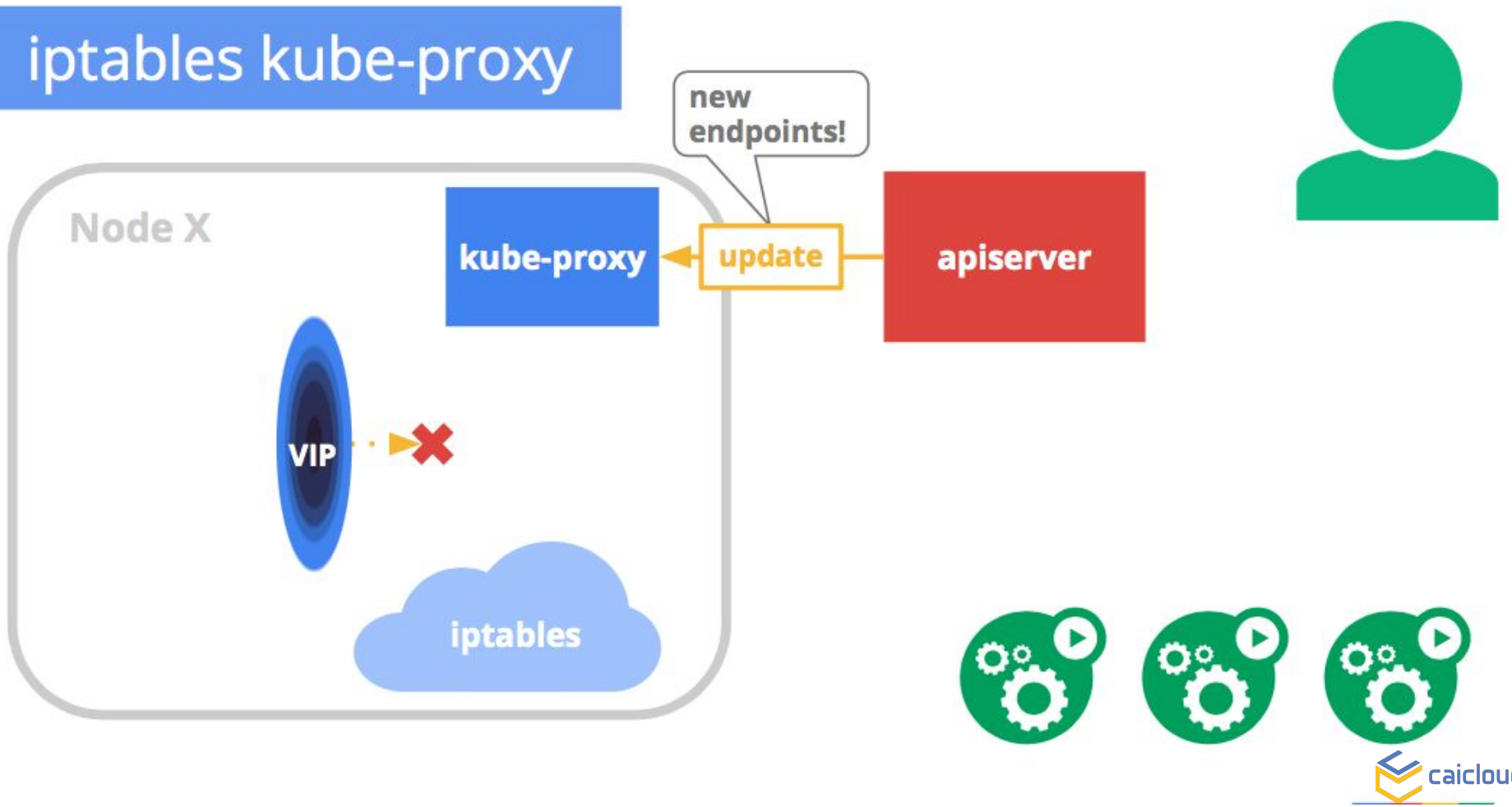
iptables kube-proxy



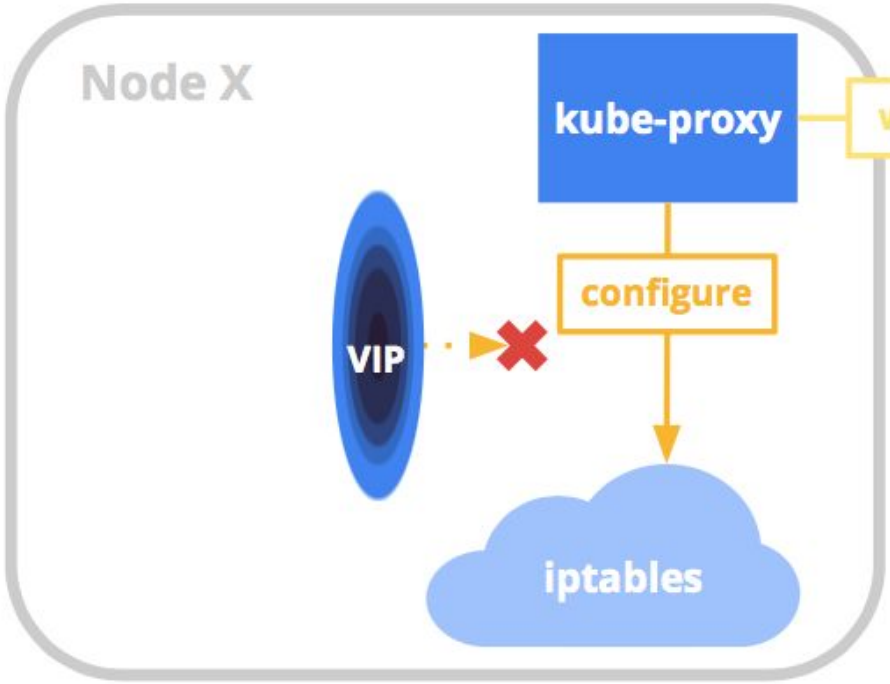
iptables kube-proxy



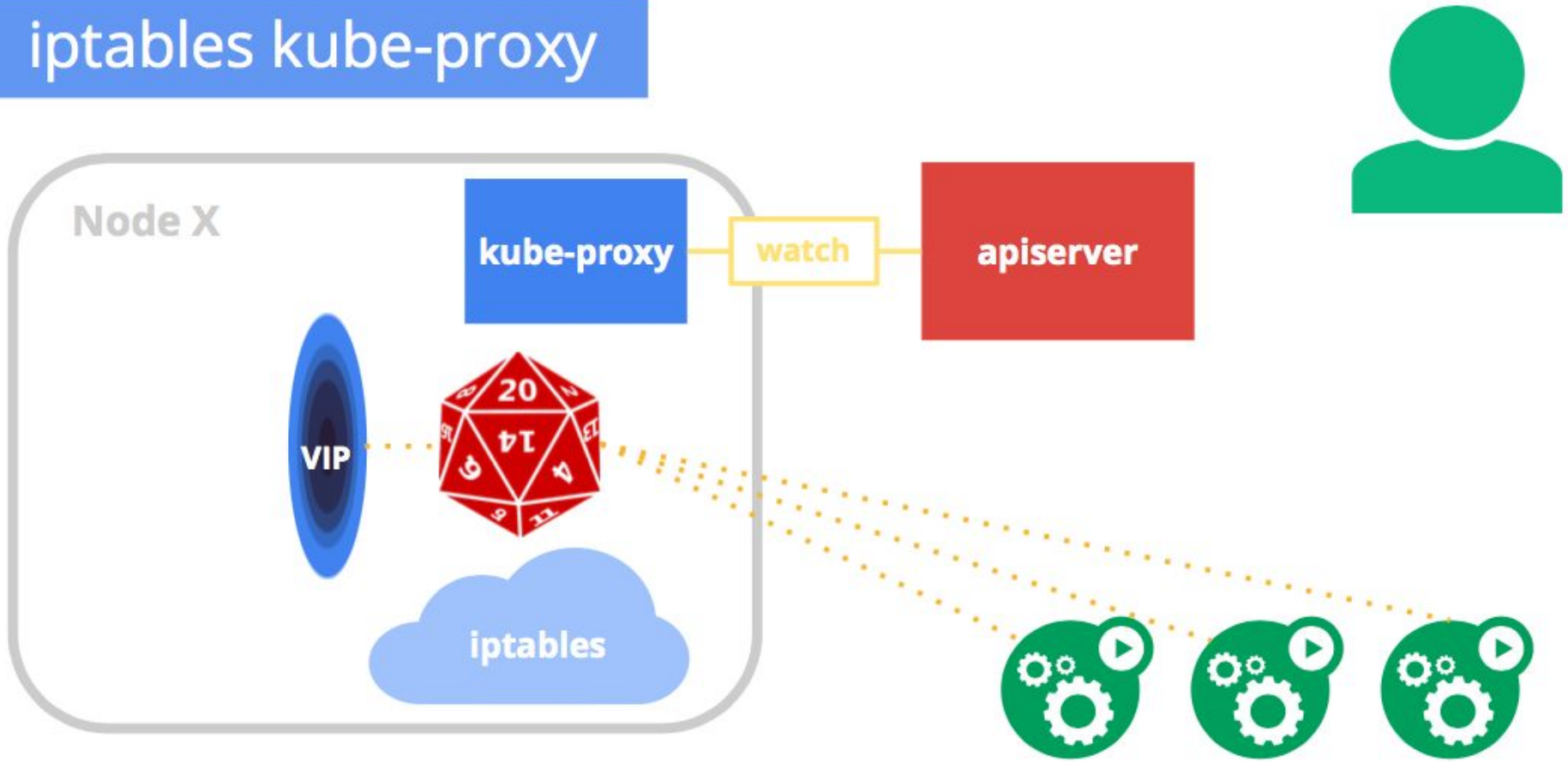
iptables kube-proxy



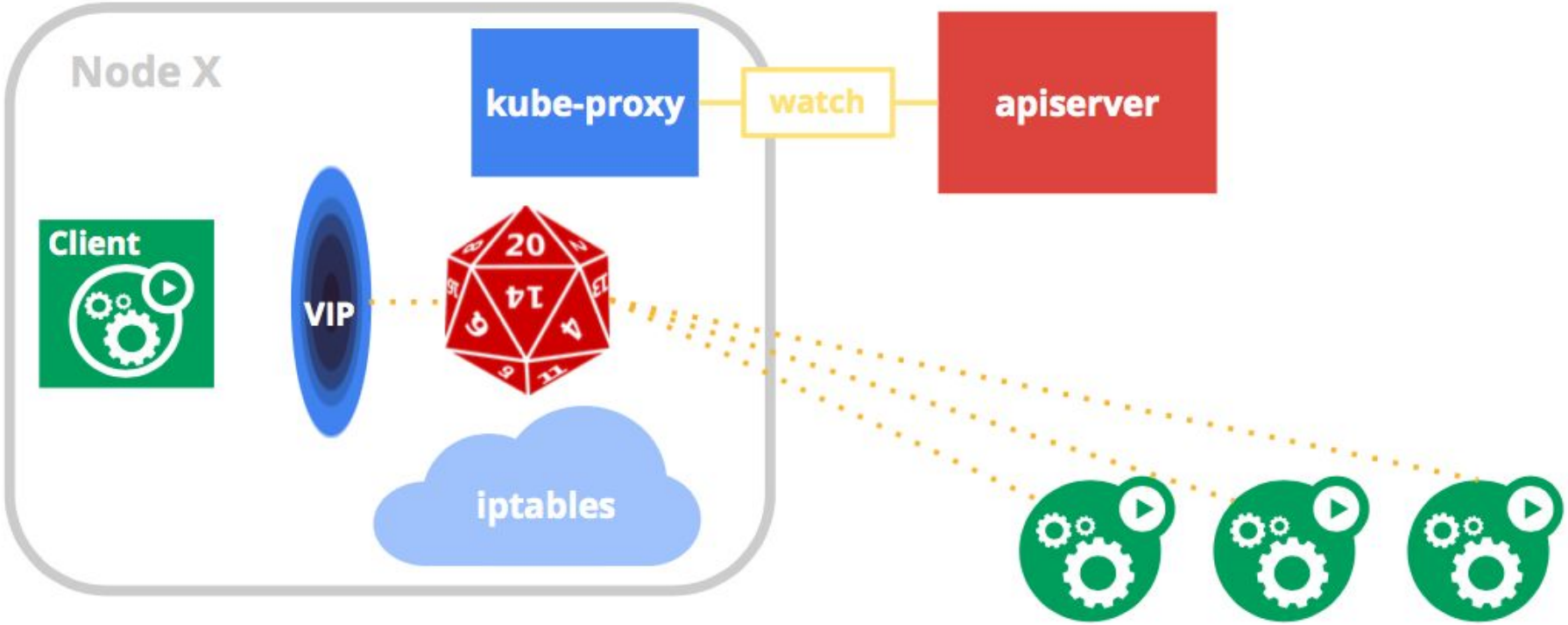
iptables kube-proxy



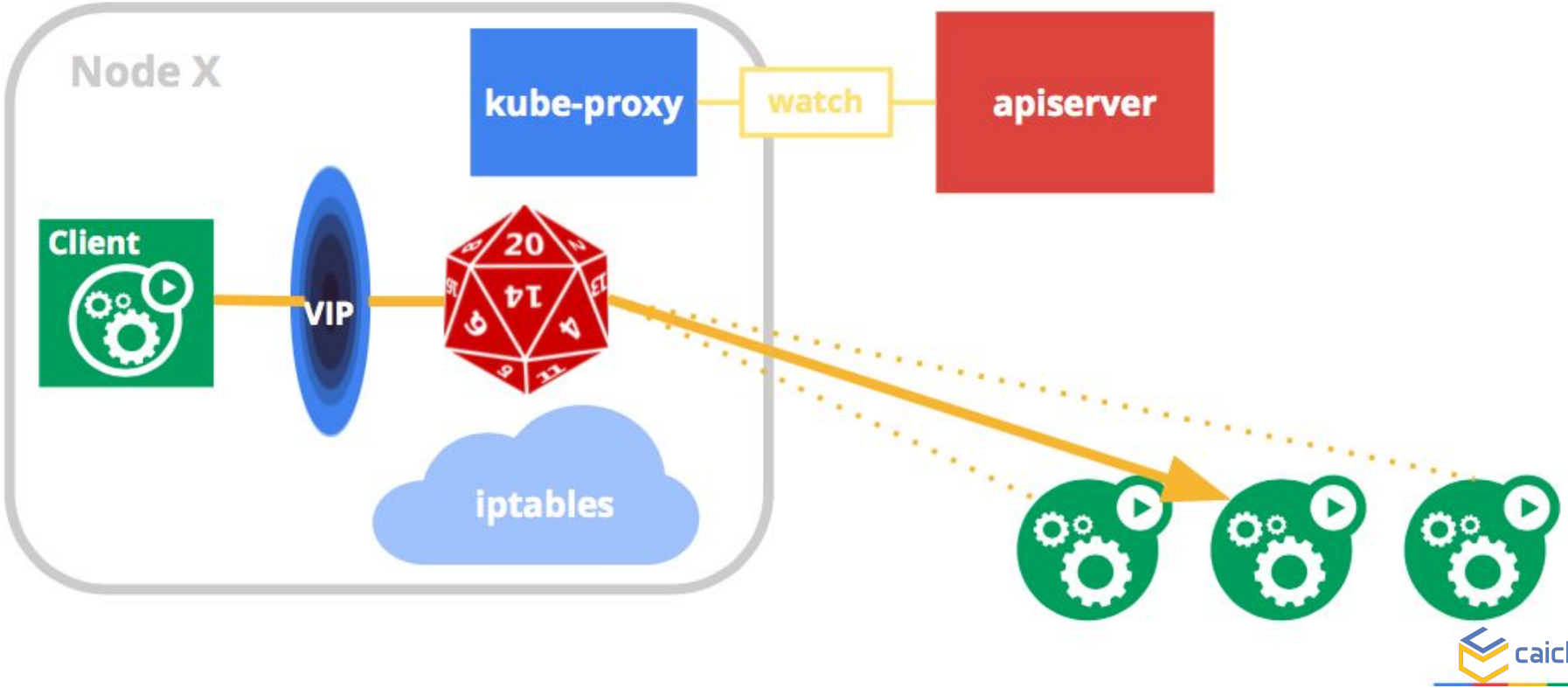
iptables kube-proxy



iptables kube-proxy

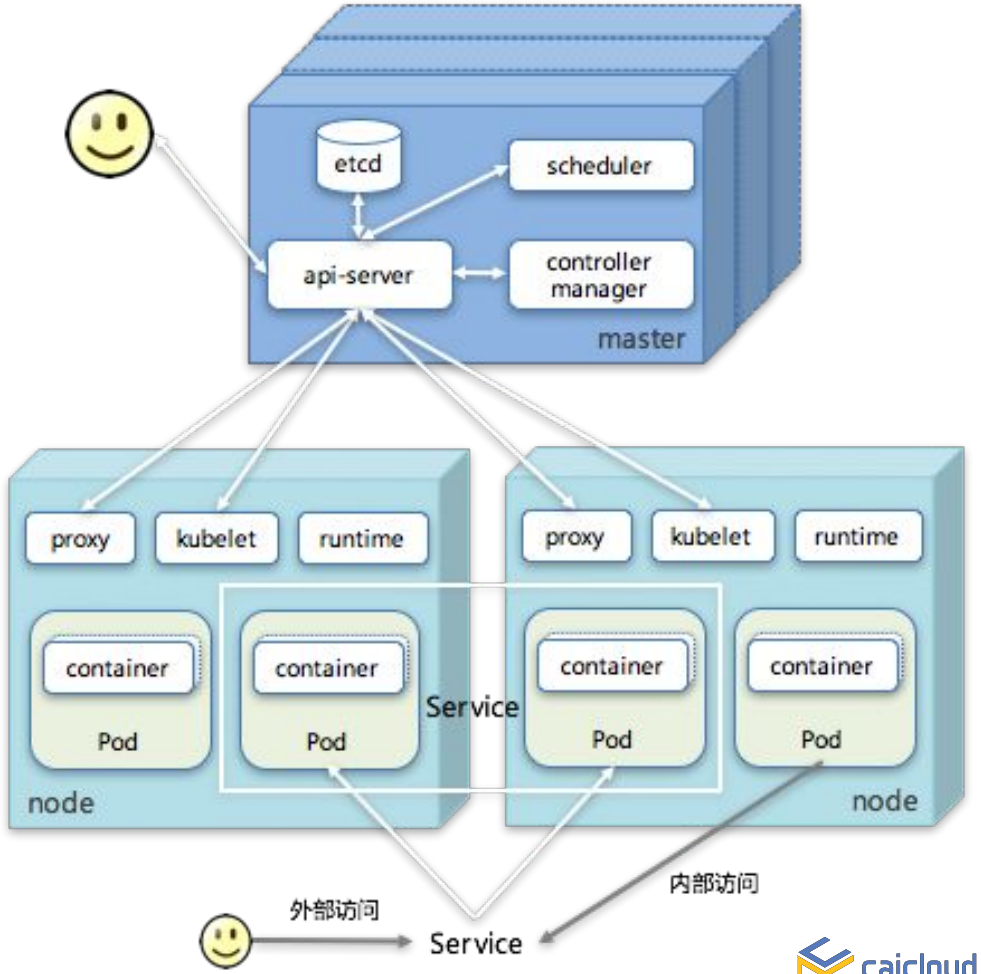


iptables kube-proxy

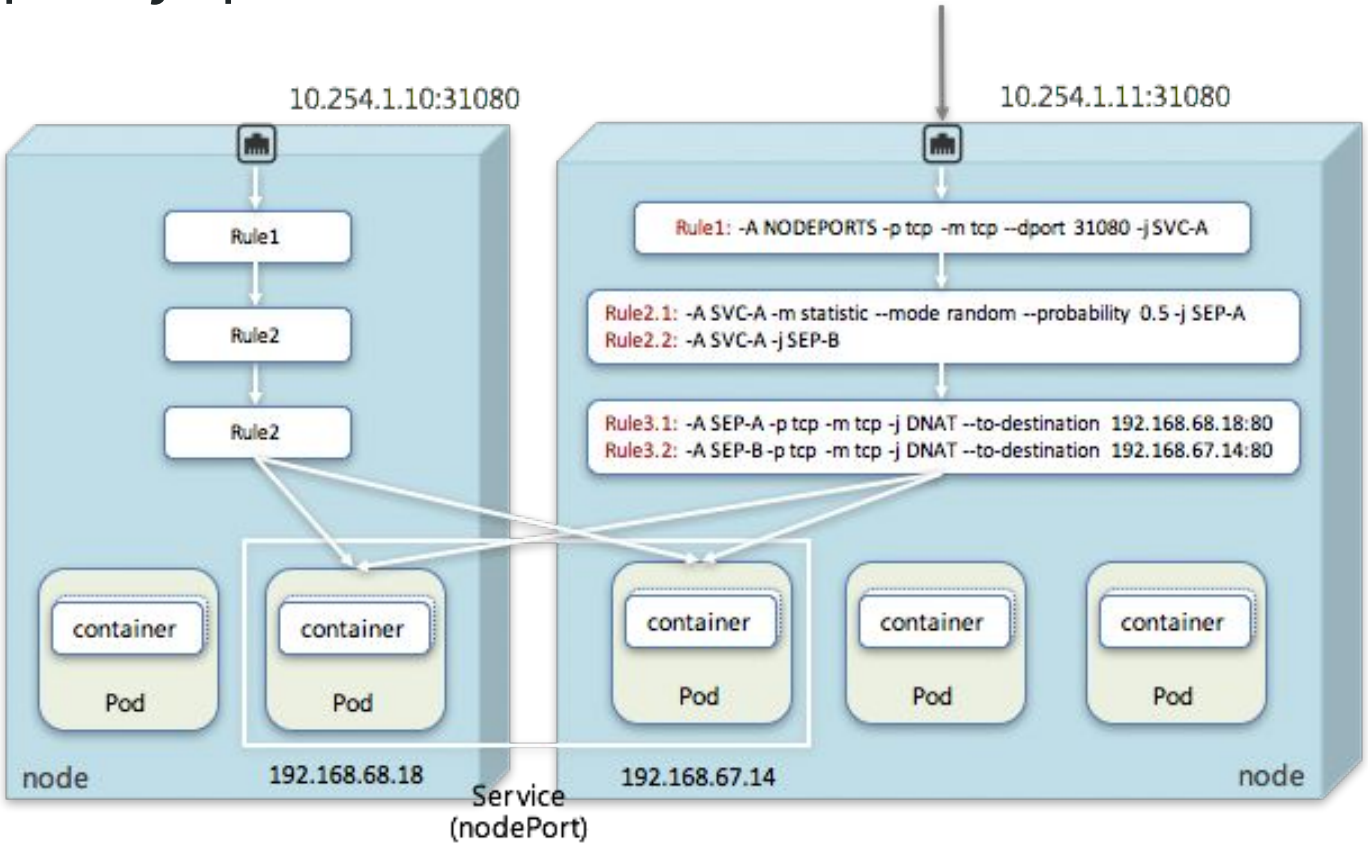


Kube-proxy iptable rules

State of a running cluster



Kube-proxy iptable rules



Service summary

- Services are just an abstraction
  - Only requirement: route (and maybe loadbalance) a virtual IP to a set of backends
- Kube-proxy is an implementation
  - Kube-proxy watches apiserver
  - iptables is re-configured on changes
- Service IPs are only available inside the cluster
  - nodePort: expose on a port on every node
  - loadBalancer: provision a cloud load balancer
  - DiY: haproxy, nginx, etc
- There could be other ways
  - Userspace
  - iptables
  - IP Virtual Server
  - Haproxy
  - etc

Kubernetes Networking - DNS

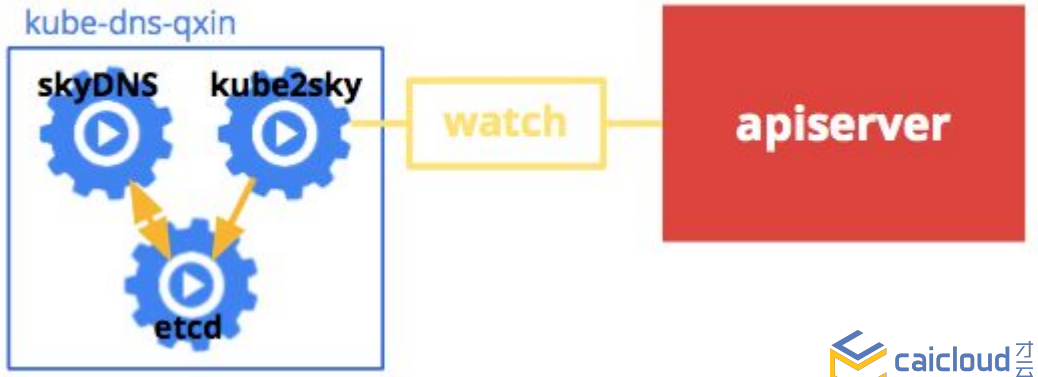
DNS

- Run SkyDNS as a pod in the cluster
  - kube2sky bridges Kubernetes API -> skyDNS (legacy)
  - skyDNS supports Kubernetes service discovery (recent)
  - Tell kubelets about it (static service IP)
- Strictly optional, but practically required
  - LOTS of things depend on it
  - Probably will become more integrated


```
kubernetes

kubernetes.default

kubernetes.default.svc.cluster.local

foo.my-namespace.svc.cluster.local
```

DNS

- Run SkyDNS as a pod in the cluster
  - kube2sky bridges Kubernetes API -> skyDNS (legacy)
  - skyDNS supports Kubernetes service discovery (recent)
  - Tell kubelets about it (static service IP)
- Strictly optional, but practically required
  - LOTS of things depend on it
  - Probably will become more integrated

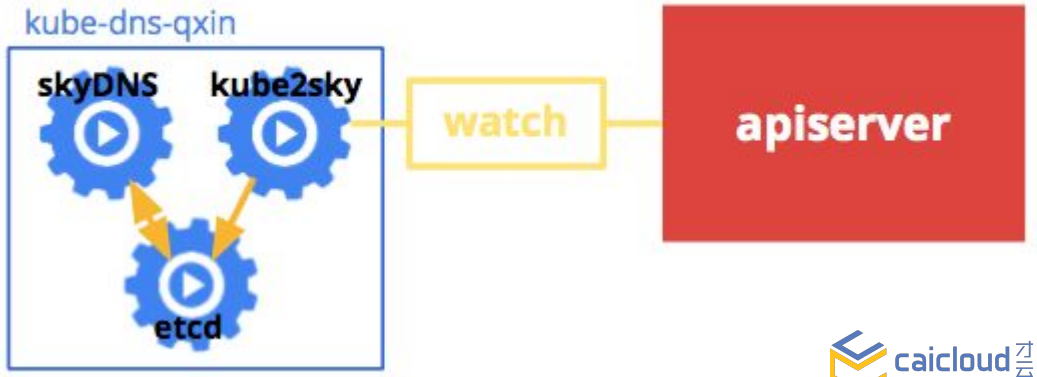


DNS

- Run SkyDNS as a pod in the cluster
  - kube2sky bridges Kubernetes API -> skyDNS (legacy)
  - skyDNS supports Kubernetes service discovery (recent)
  - Tell kubelets about it (static service IP)
- Strictly optional, but practically required
  - LOTS of things depend on it
  - Probably will become more integrated

/etc/resolv.conf

```
nameserver 10.0.0.10
...
```



DNS

- Run SkyDNS as a pod in the cluster
  - kube2sky bridges Kubernetes API -> skyDNS (legacy)
  - skyDNS supports Kubernetes service discovery (recent)
  - Tell kubelets about it (static service IP)
- Strictly optional, but practically required
  - LOTS of things depend on it
  - Probably will become more integrated

/etc/resolv.conf

```
nameserver 10.0.0.10
...
```