Kubernetes 101/201

邓德源 才云科技

Kubernetes API design

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

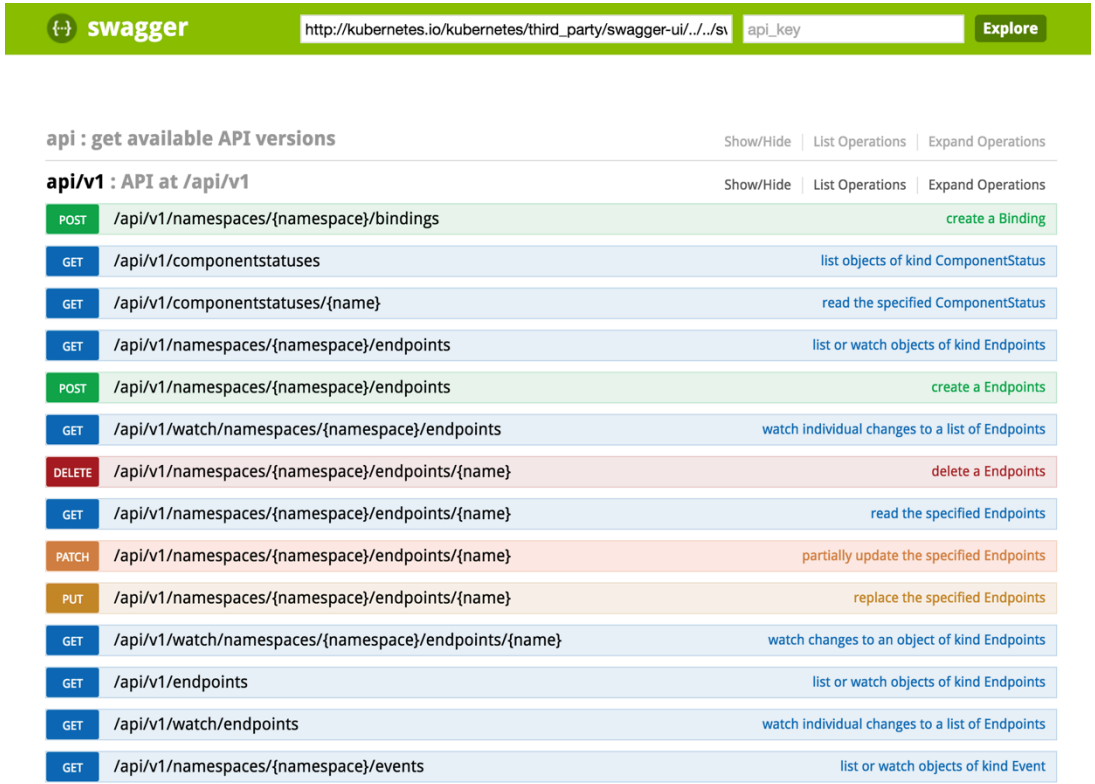
```
apiVersion: v1
kind: Pod
metadata:
    namespace: default
spec:
    containers:
    dnsPolicy: ClusterFirst
    nodeName: i-2zea47skez7ye2xr438v
    restartPolicy: Always
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    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

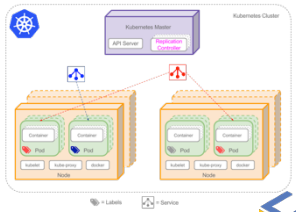
- 每个 API group 中的资源可以单独的 enable/disable
- 每个 API group 有不同的版本，单独开发维护


```
{
  "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

```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”

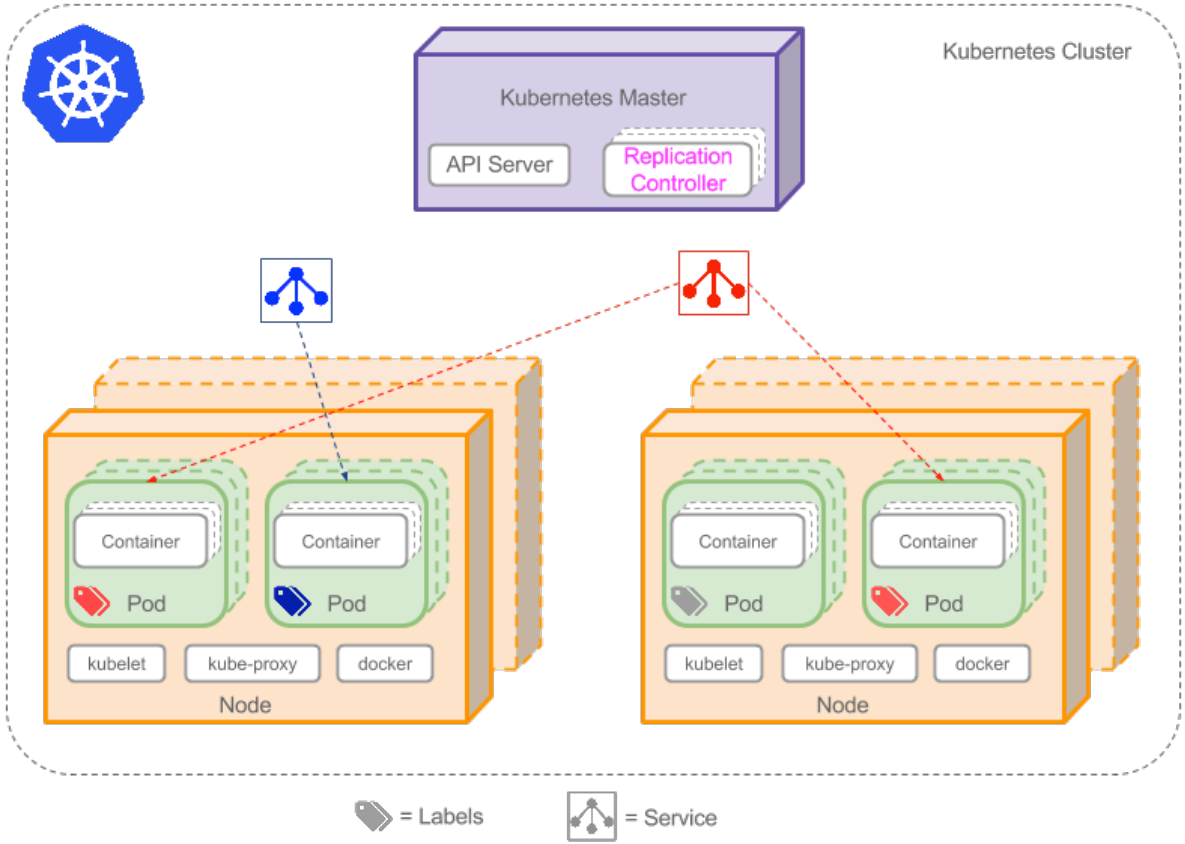


Kubernetes 架构与工作流程

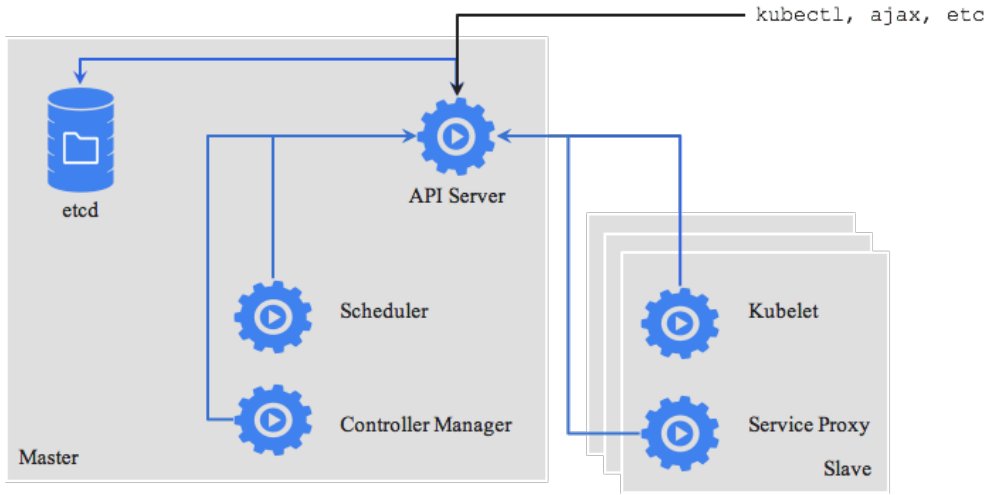
Kubernetes provides rich functionality set that can serve as building blocks to construct end to end solutions, and sky is the limit.

-- Xin Zhang

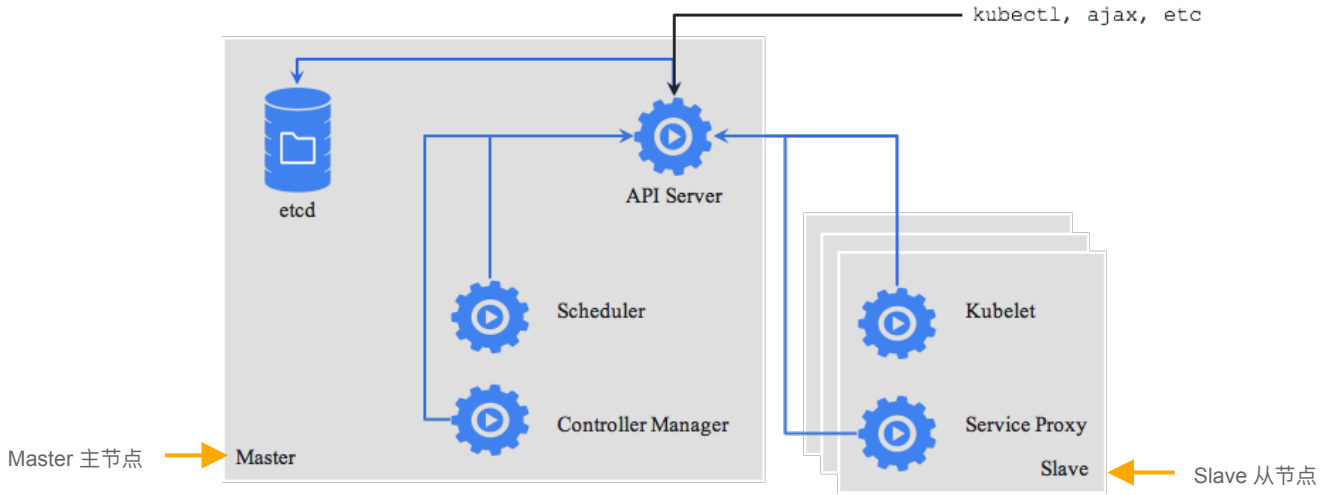
Kubernetes 架构



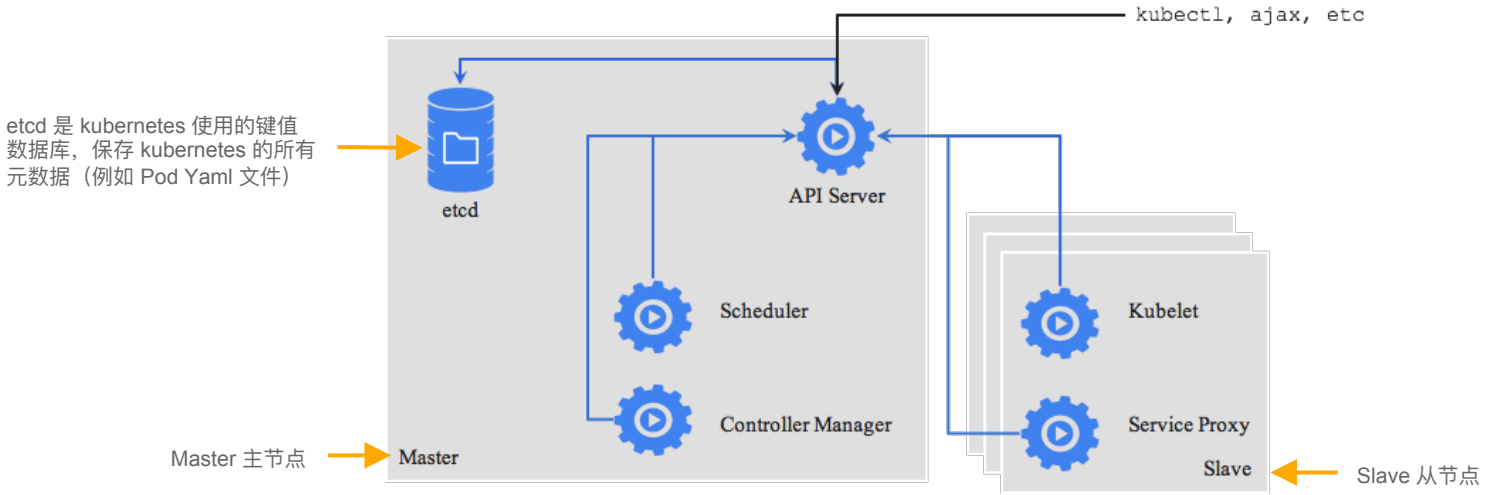
Kubernetes 架构



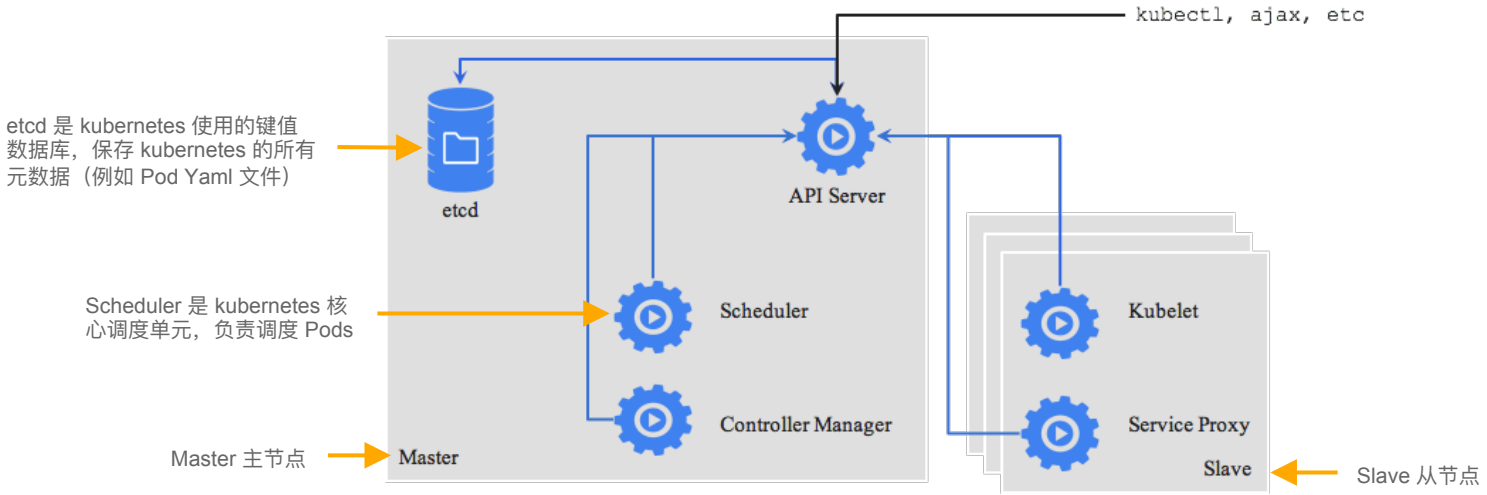
Kubernetes 架构



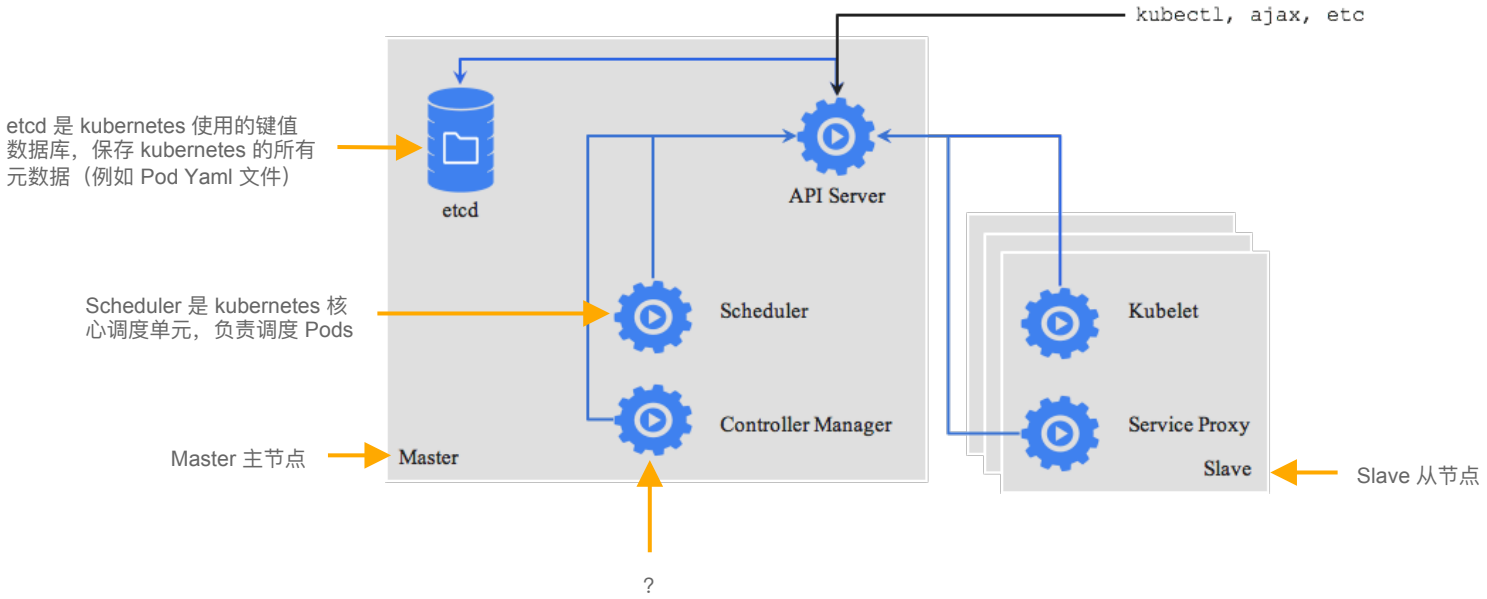
Kubernetes 架构



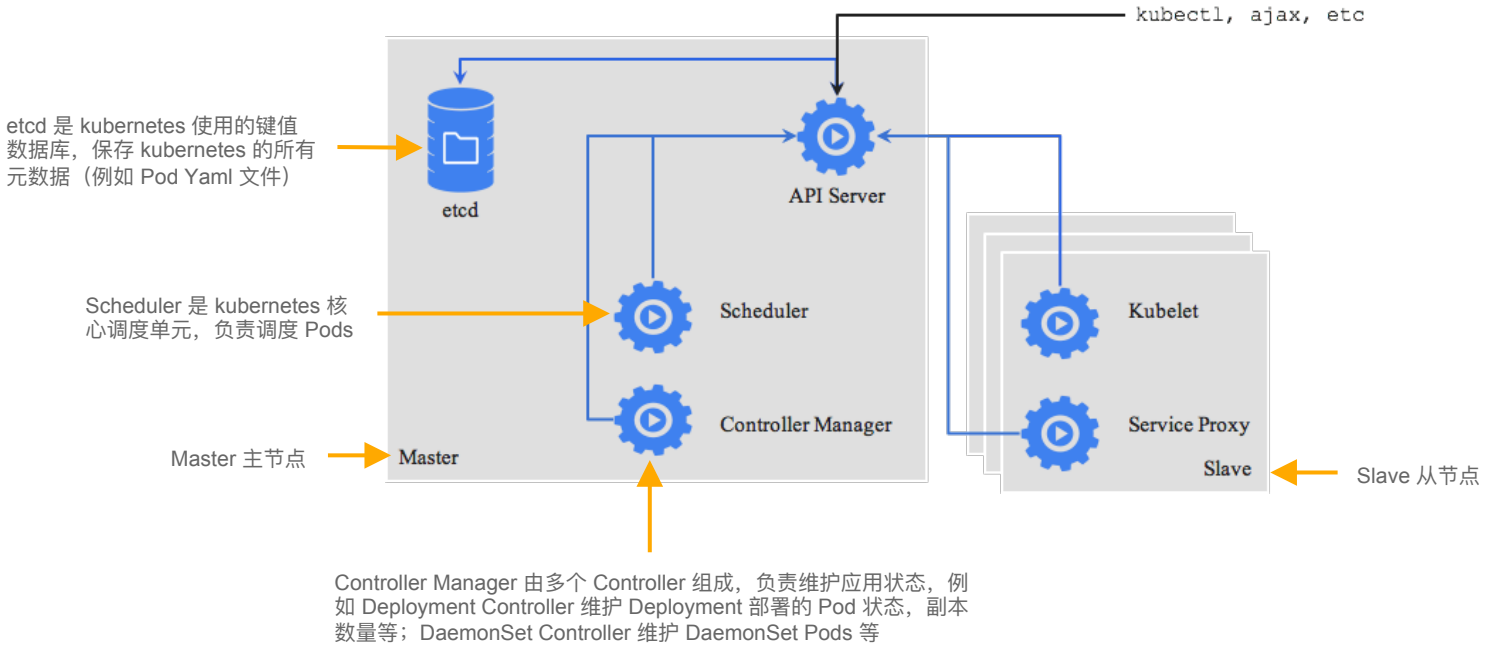
Kubernetes 架构



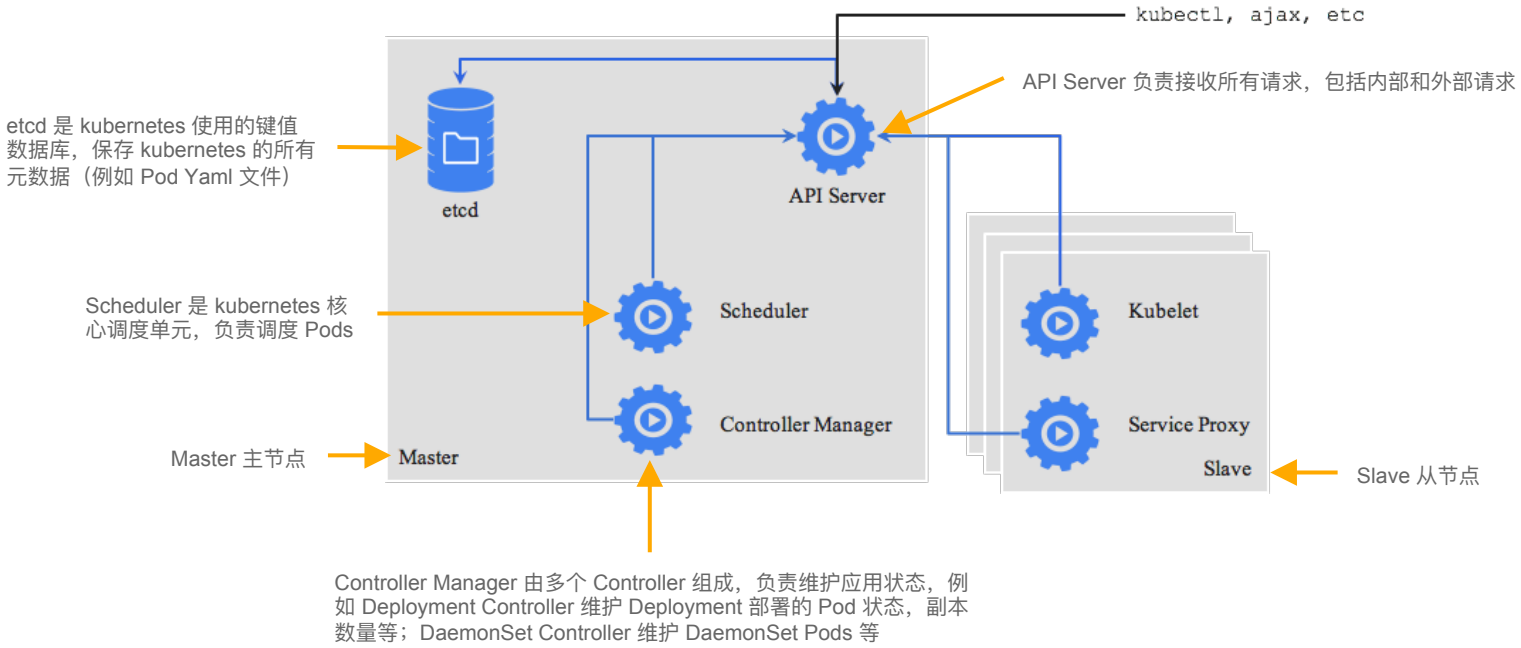
Kubernetes 架构



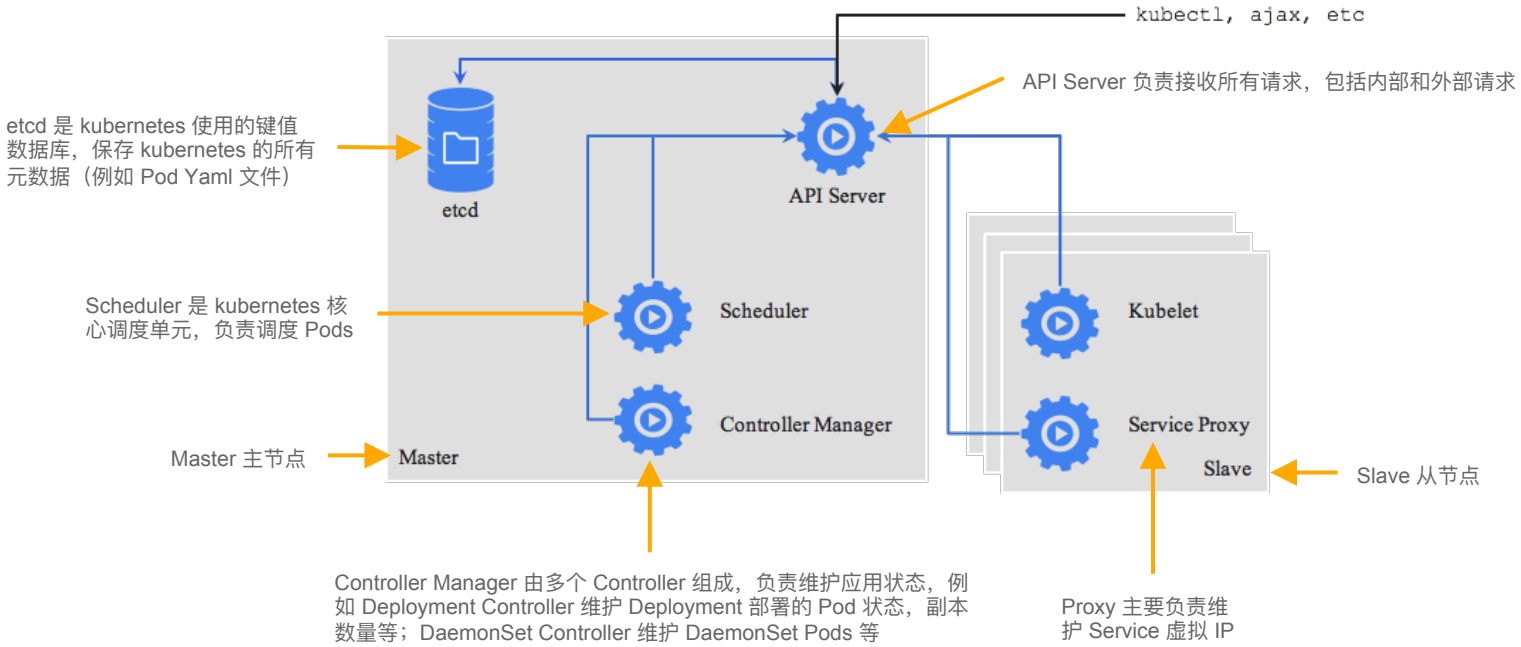
Kubernetes 架构



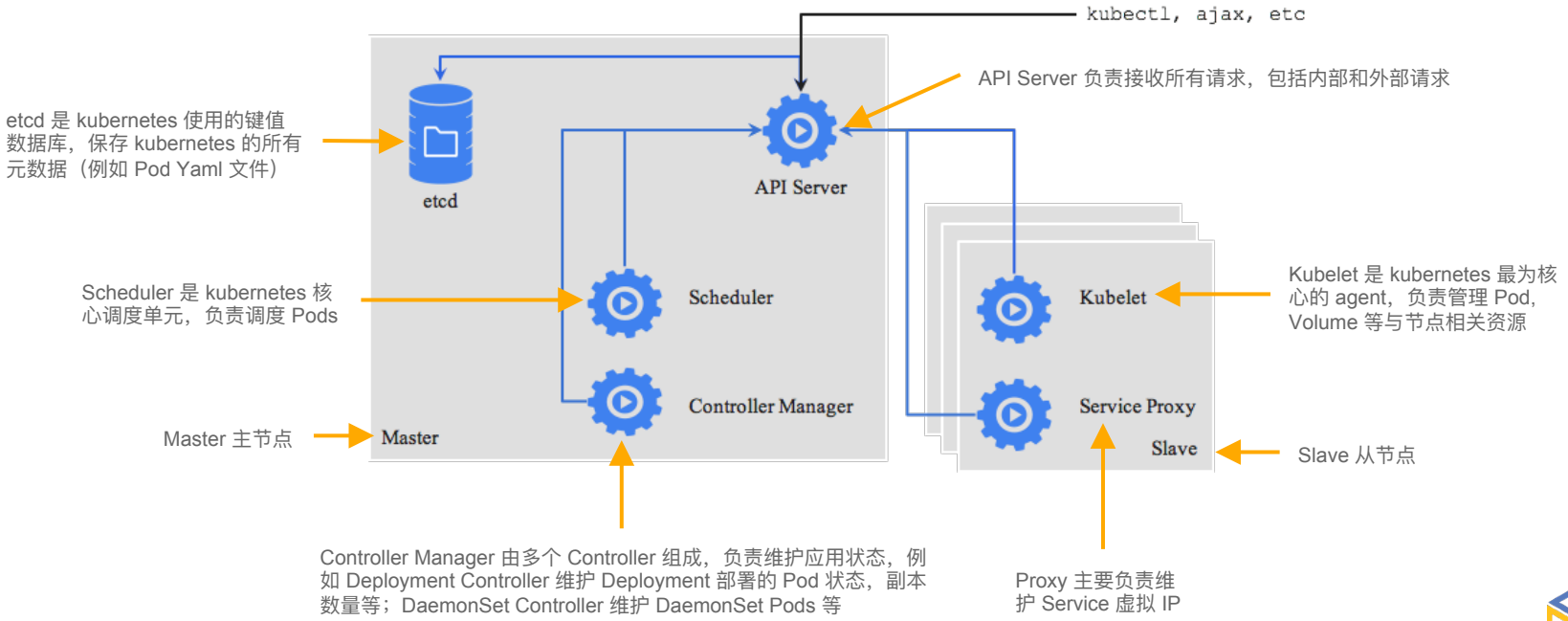
Kubernetes 架构



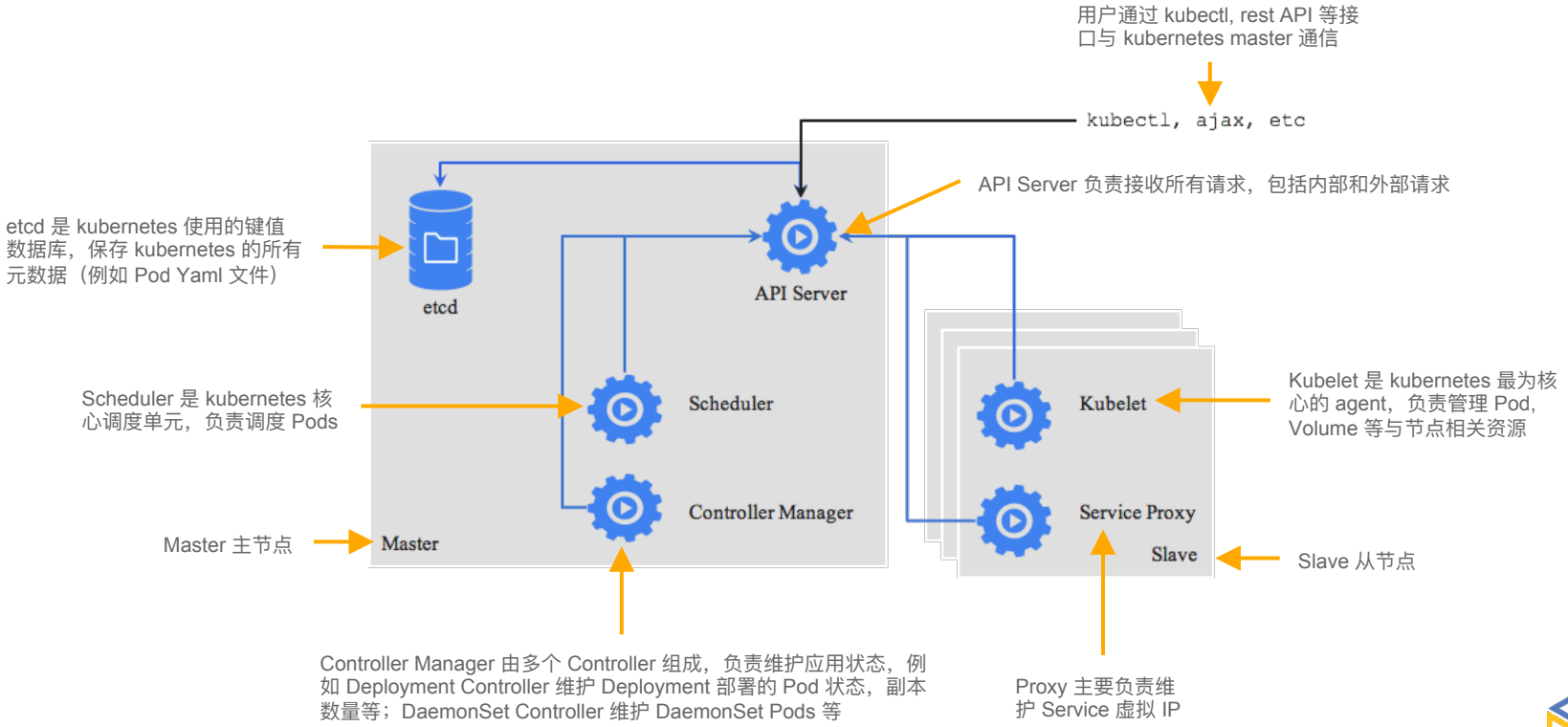
Kubernetes 架构



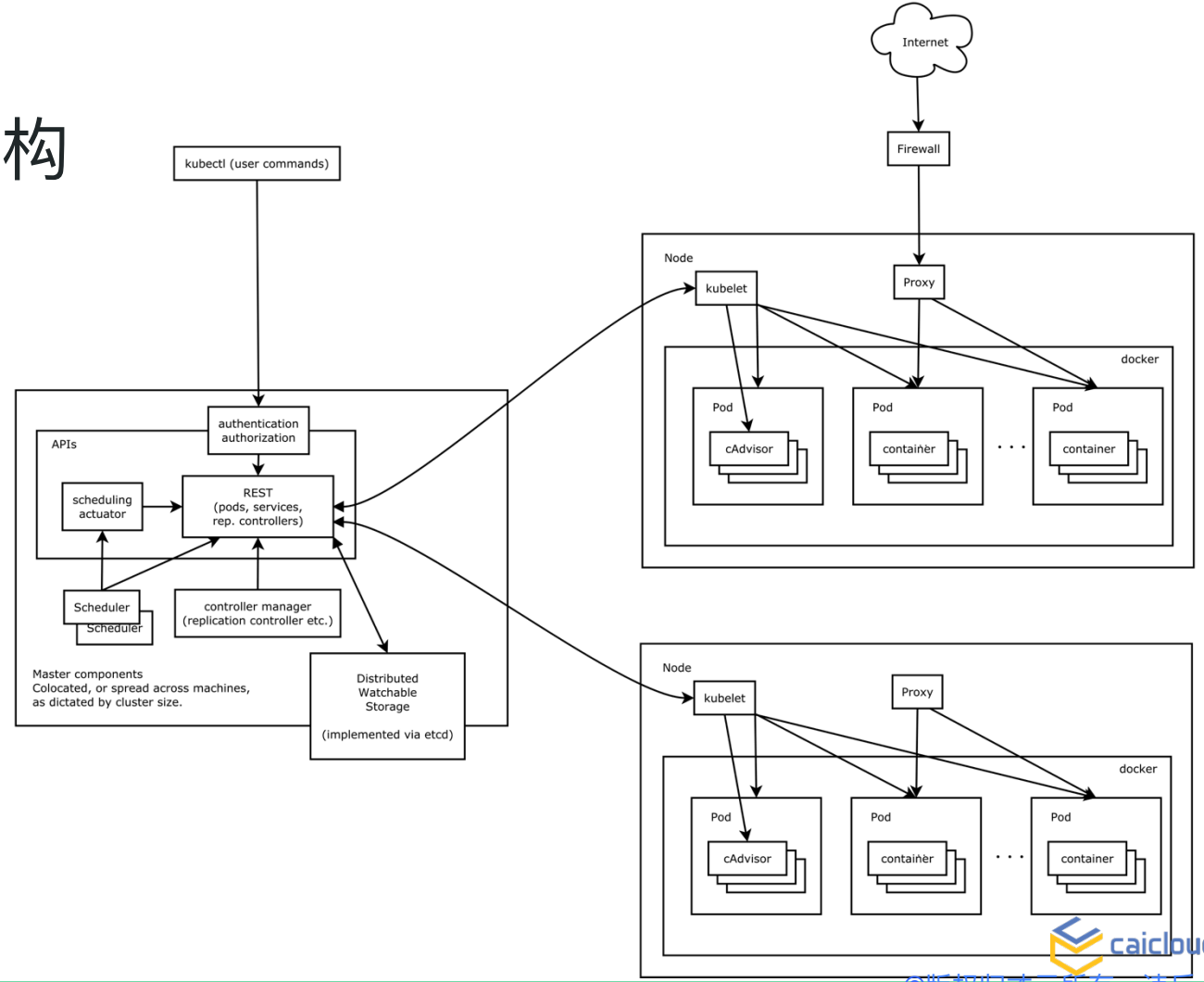
Kubernetes 架构



Kubernetes 架构



Kubernetes 架构



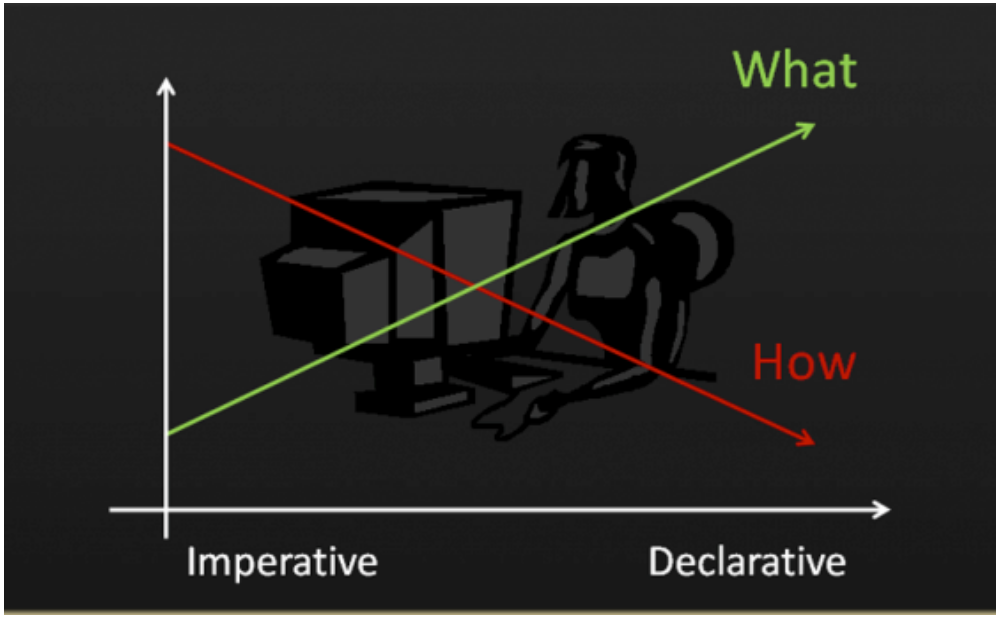
官方 Official 架构图

声明式 VS 命令式

声明式：告诉“机器”你想要的是什么(what)，让机器想出如何去做(how)。

命令式：命令“机器”如何去做(how)，这样不管你想要的是什么(what)，它都会按照你的命令实现。

- Example
  - Map 函数：将直接遍历整个数组的过程归纳抽离出来，让我们专注于描述我们想要的是什么(what)
  - Reduce 函数：抽离如何遍历数组和状态管理部分的实现，提供一个通用的方式把一个 list 合并成一个值。我们需要做的只是指明我们想要的是什么(what)

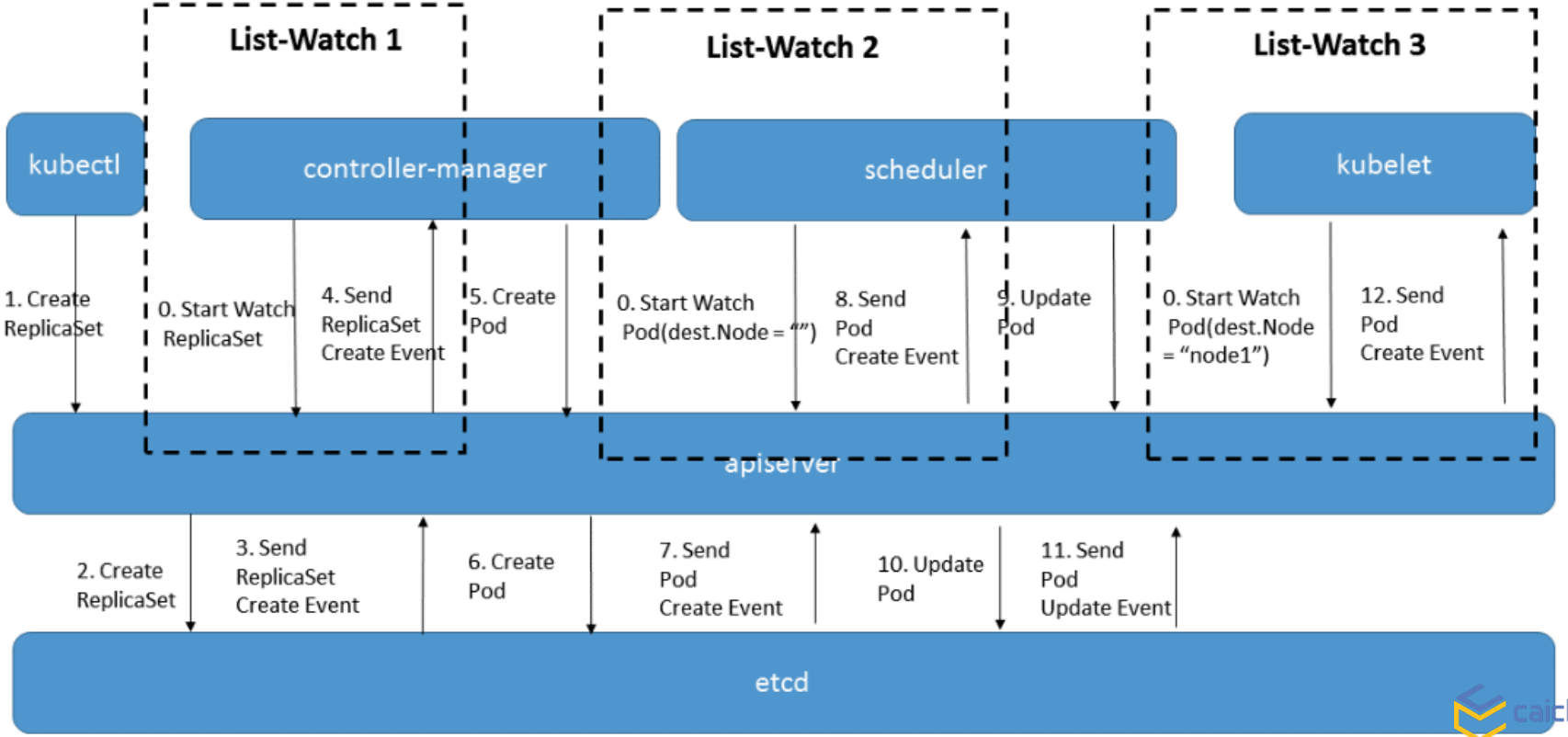


声明式操作在分布式系统中的好处：稳定，不怕丢操作或运行多次

Kubernetes 架构解析

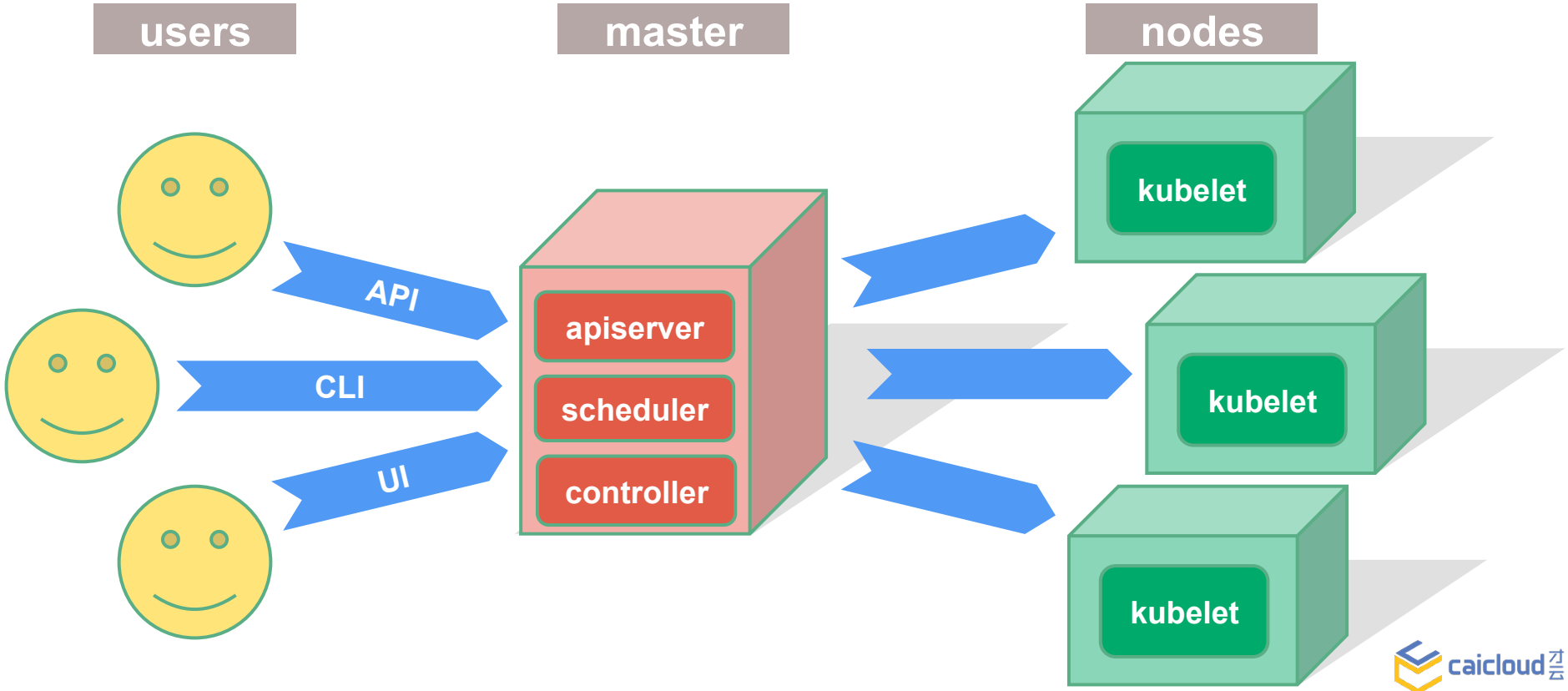
- K8S 集群组成
  - 分布式存储（Etcd）
  - 控制节点（Master）
  - 工作节点（Node）
- 只有 apiserver 与存储通信
  - 用户直接访问 apiserver
  - 内部进程，包括 kubelet, controller 均通过 apiserver 访问存储
  - 出于安全考虑
- 配置管理操作声明式而非命令式

Apiserver list-watch

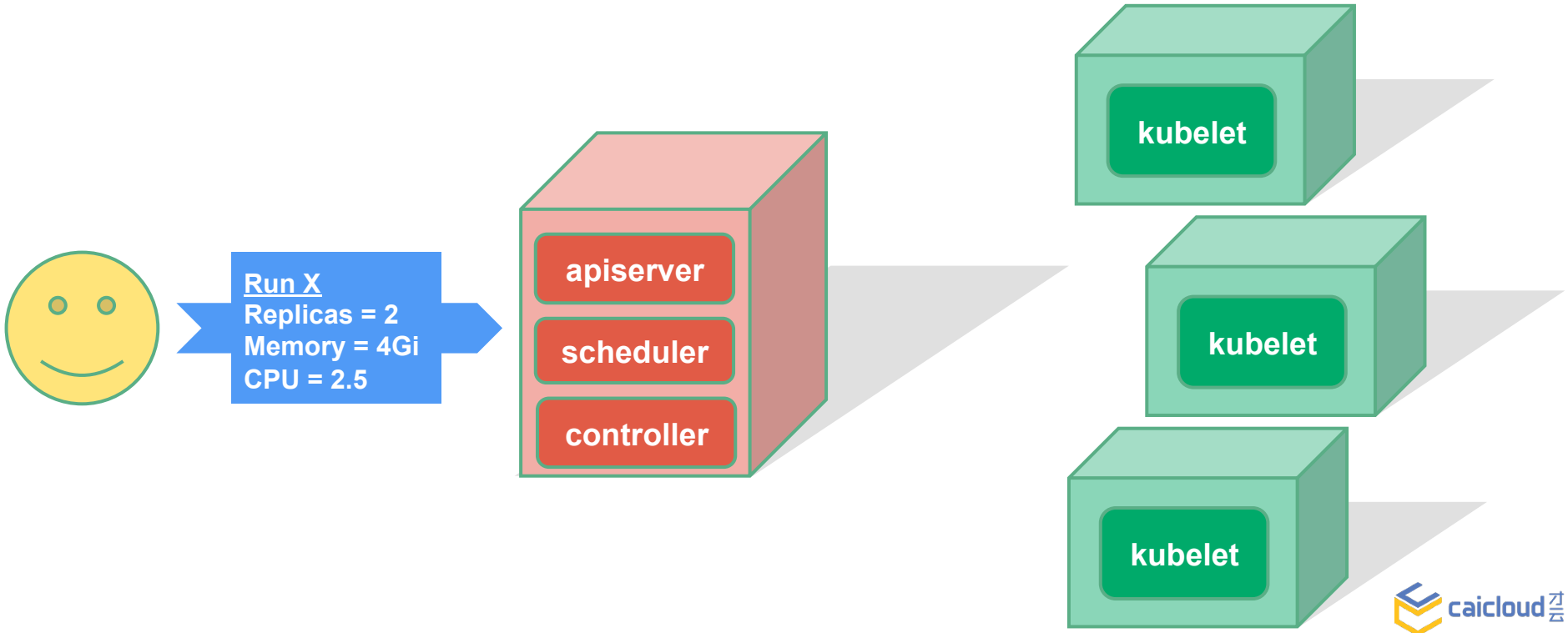


Kubernetes 工作流程

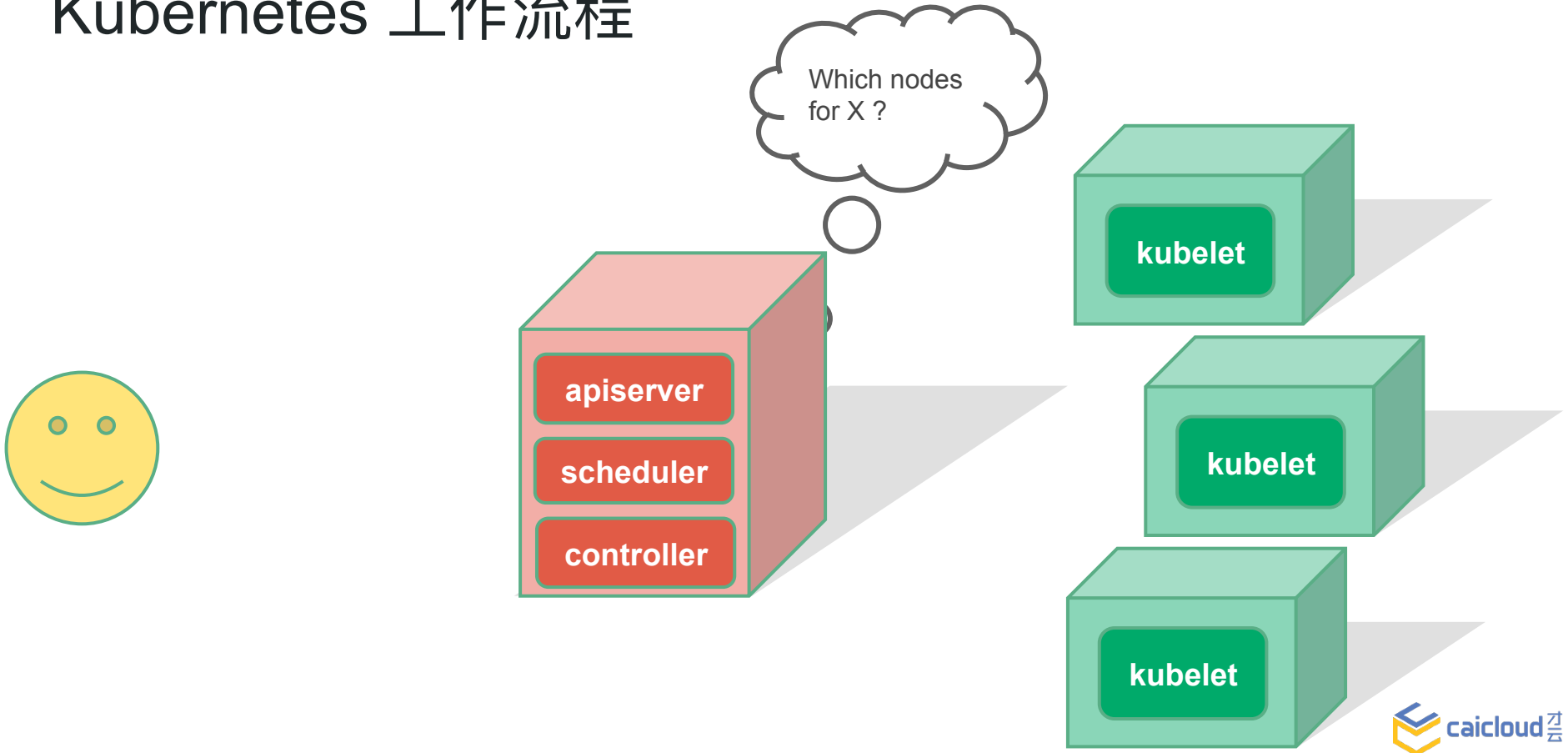
Kubernetes 工作流程



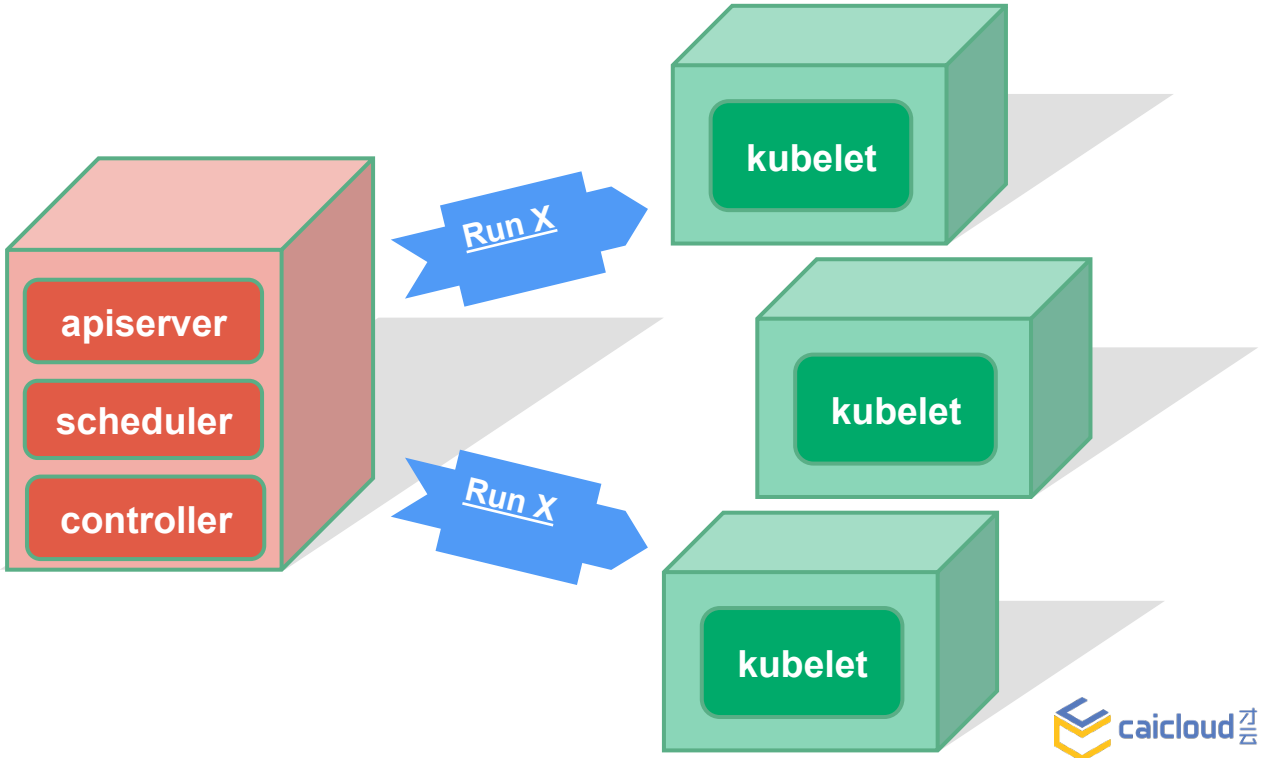
Kubernetes 工作流程



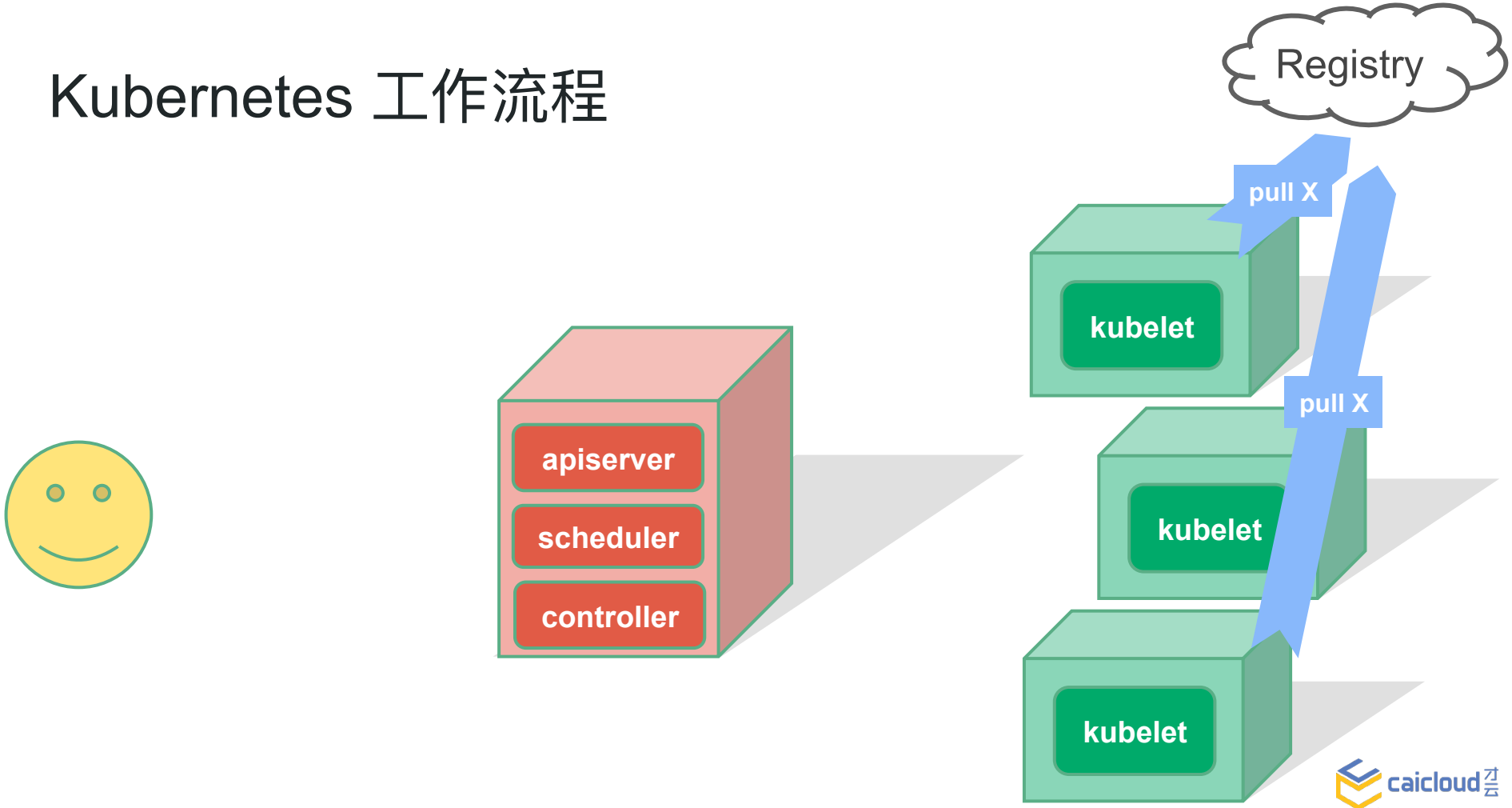
Kubernetes 工作流程



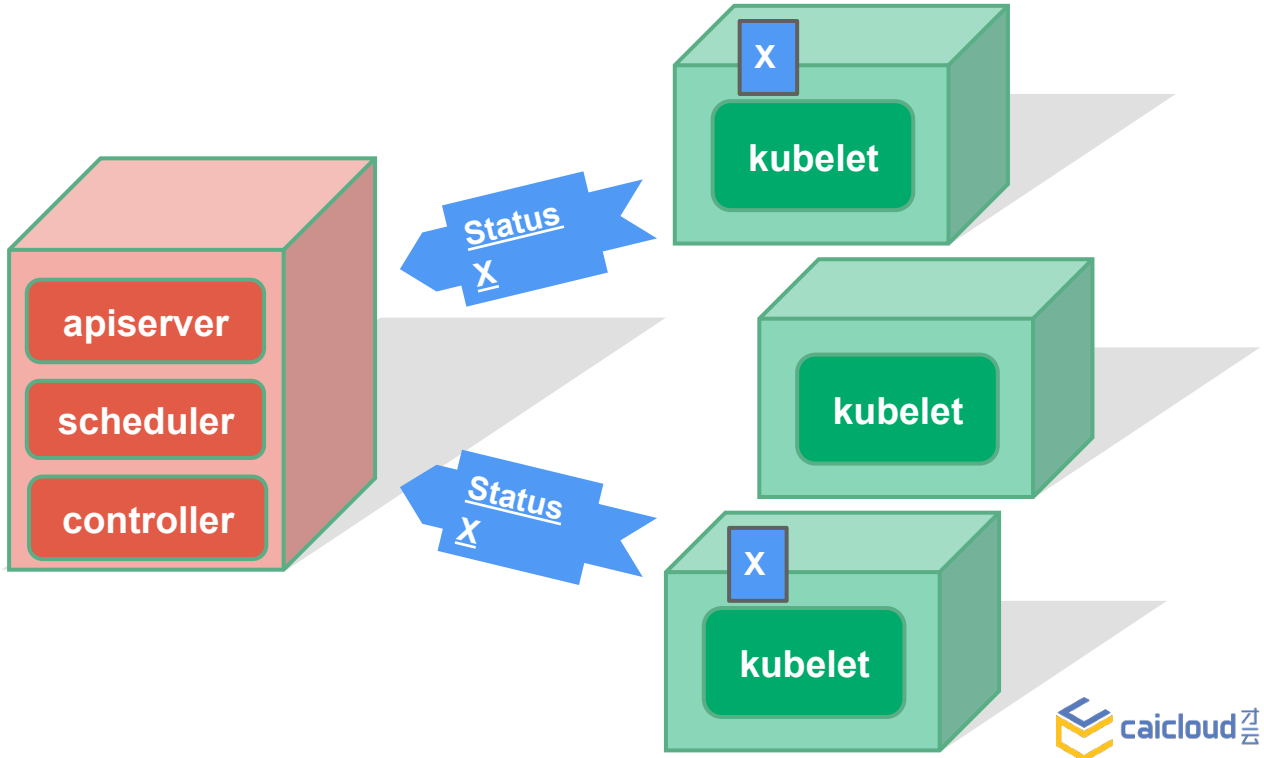
Kubernetes 工作流程



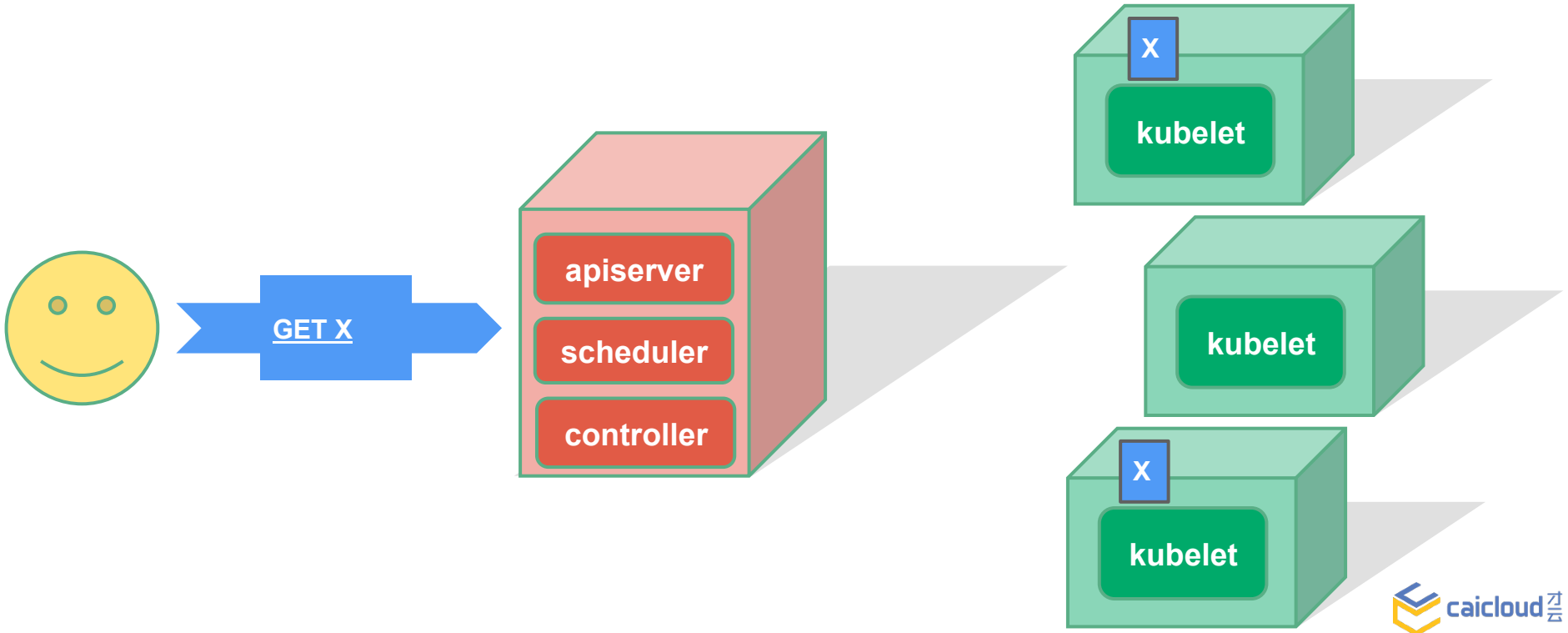
Kubernetes 工作流程



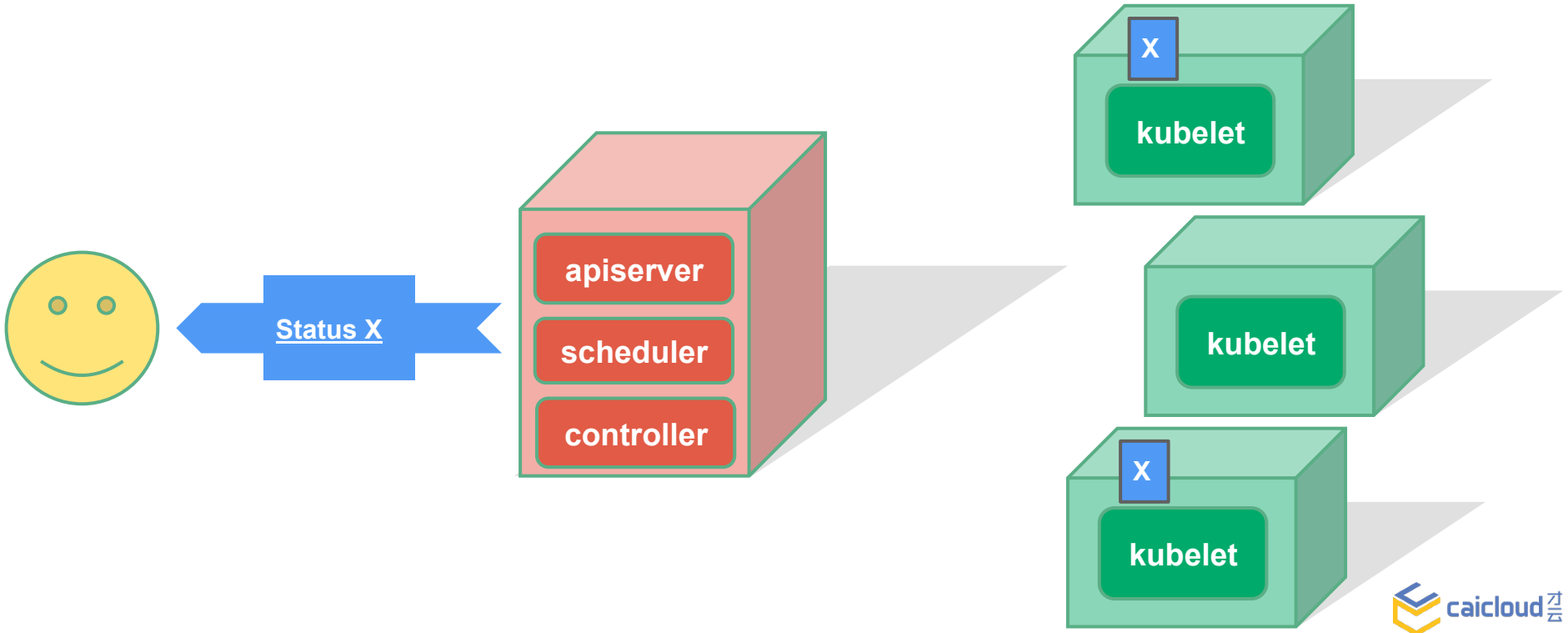
Kubernetes 工作流程



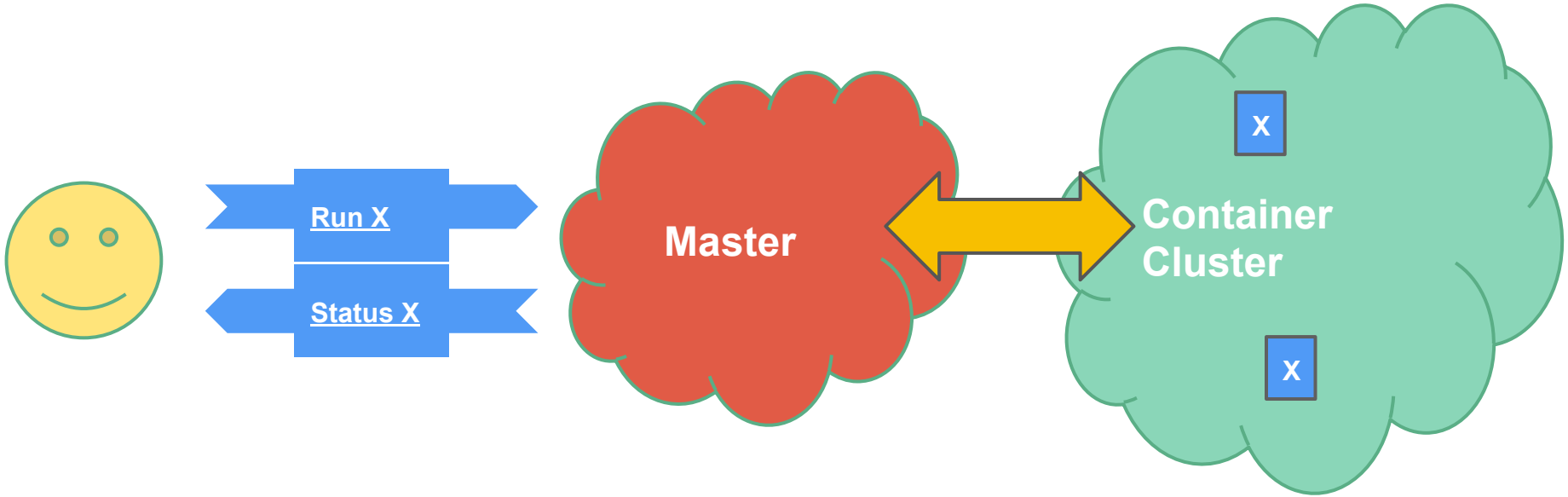
Kubernetes 工作流程



Kubernetes 工作流程



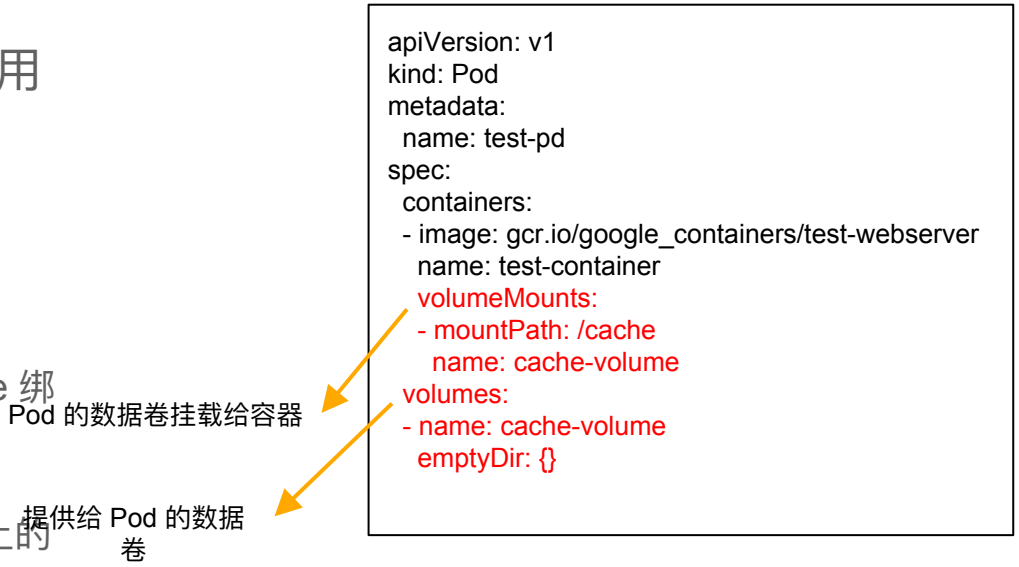
Kubernetes 工作流程 - 用户角度



Kubernetes advanced concepts (optional)

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



Namespace, 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，MEM
  - k8s 元素数量配额

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

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
```

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
```

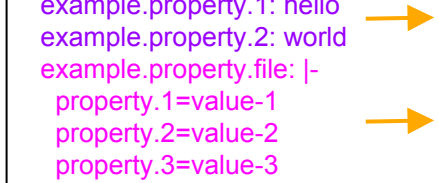
还可以通过 kubectl 设置 HPA：

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

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

```yaml
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

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

2

```yaml
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

```yaml
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
```

From ReplicationController to ReplicaSet to Deployment

- From Replication Controller to Replica Set
  - Changes in label selector matching
  - From equality matching to set matching
- From Replica Set to Deployment
  - Declarative update
  - Rollout version control & history
  - Quick roll back to multiple revisions
  - Multiple tunable parameters
  - Server0side

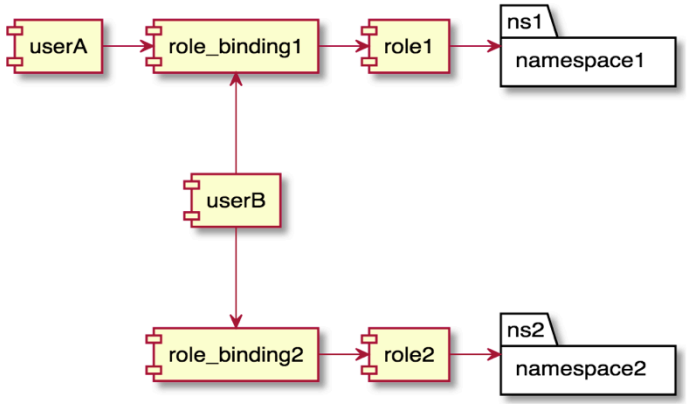
Replica Set

- 下一代的 Replication Controller
  - RC 支持 “kubectl rolling-update”
  - RC 只支持 Equality-based 选择器
  - RS 支持 Set-based 和 Equality-based 选择器
- 如何使用
  - 一般不单独使用
  - Deployment
  - HPA（Horizontal Pod Autoscaler）

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

    kube-apiserver --authorization-mode=ABAC --authorization-policy-file=/etc/kubernetes/tokens/abac.json ...
  - 访问策略只能跟用户直接关联

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

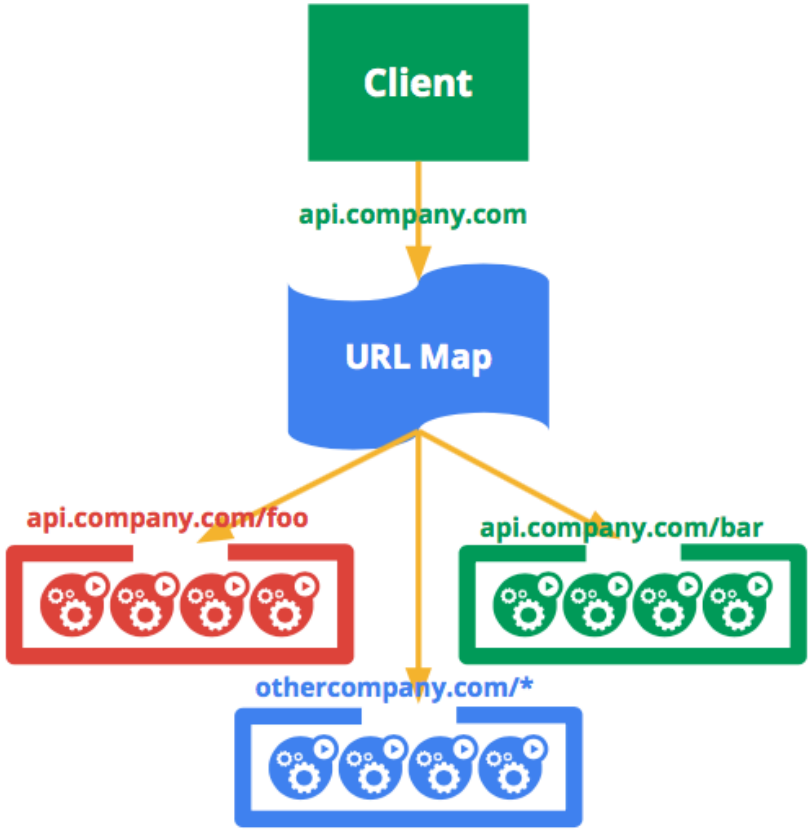
```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: nfs
spec:
 capacity:
   storage: 1Mi
 accessModes:
   - ReadWriteMany
 nfs:
   # FIXME: use the right IP
   server: 10.244.1.4
   path: "/exports"
```

```
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
```

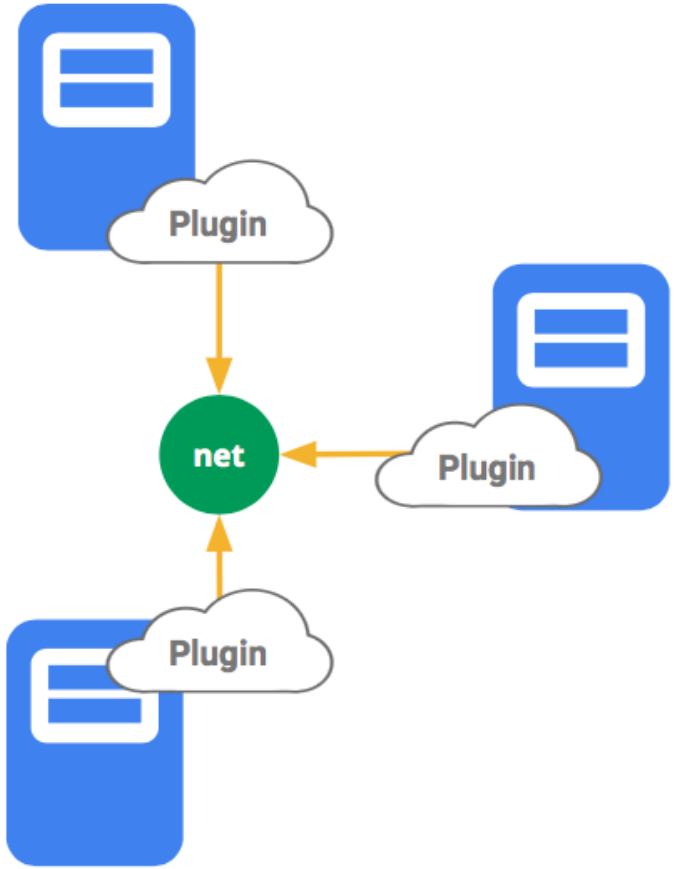
Ingress (L7)

- 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



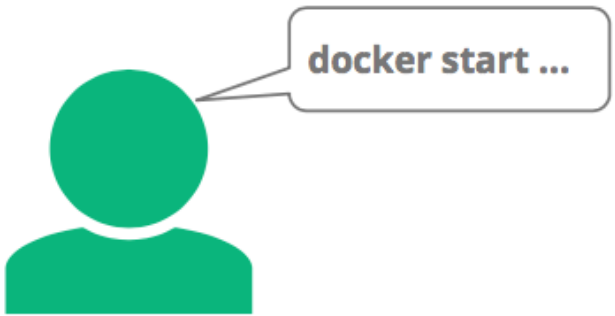
Network Plugins

- 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

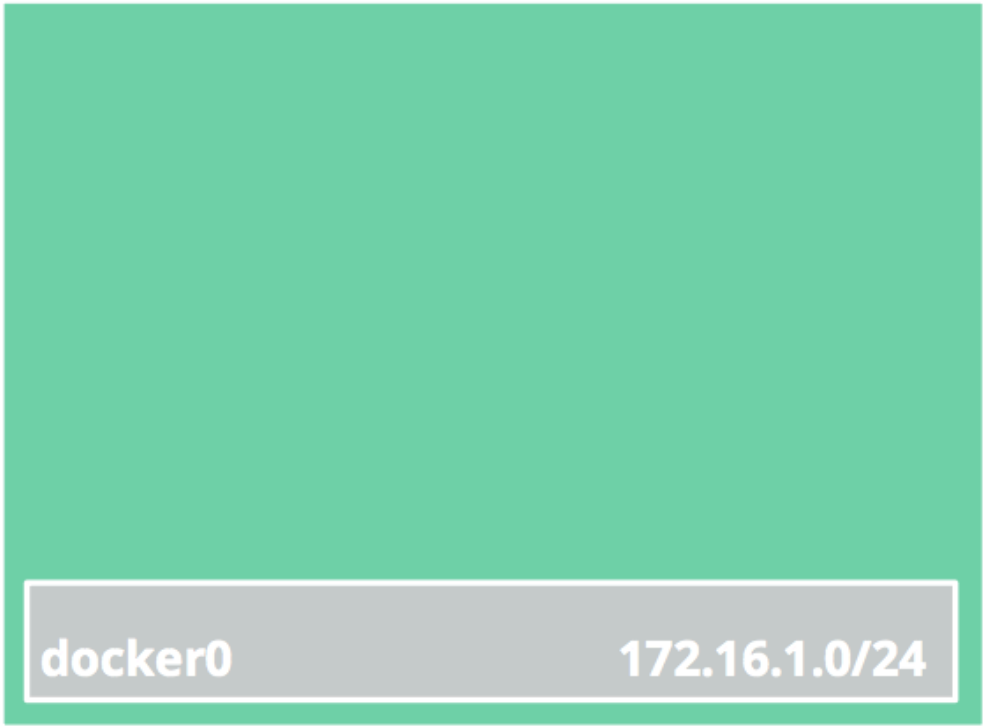


Kubernetes 网络模型

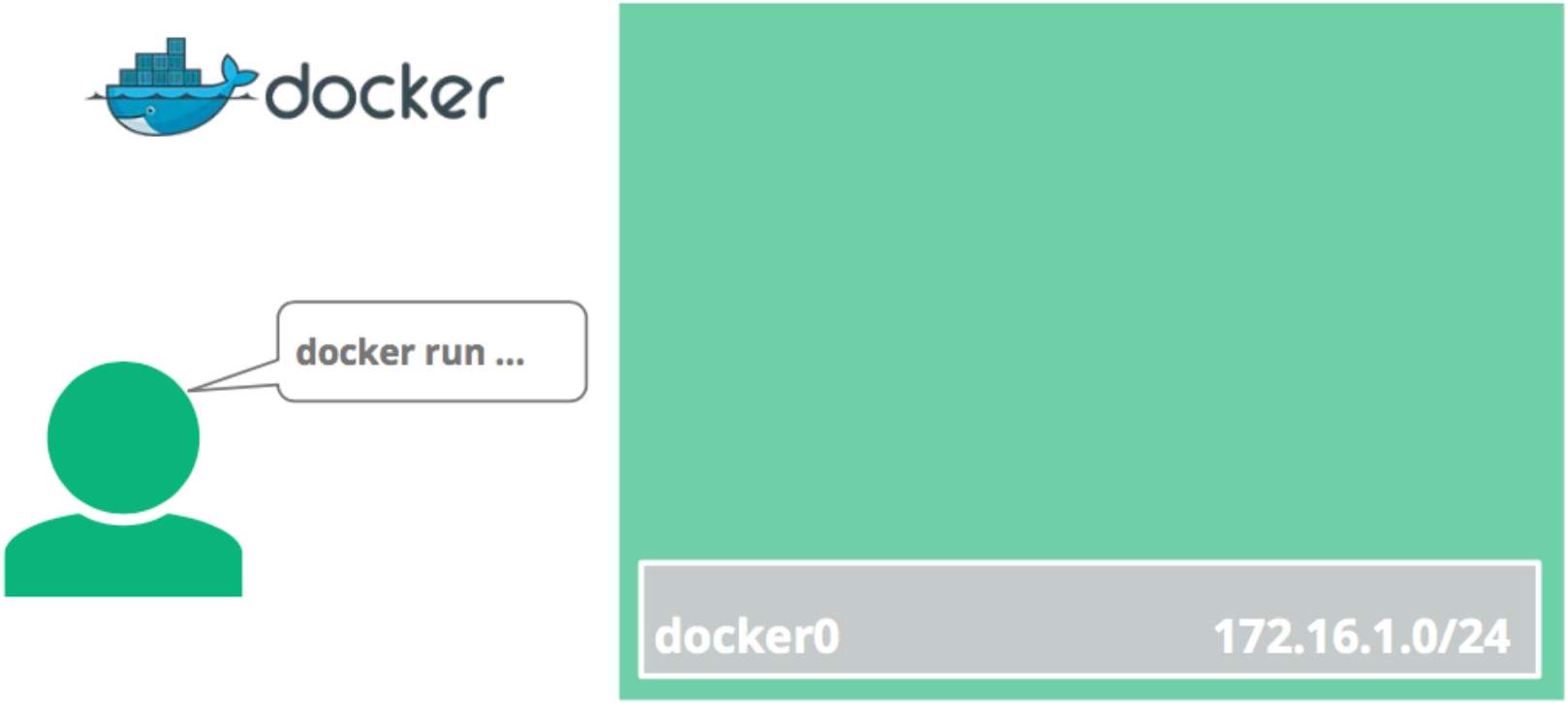
Docker Networking



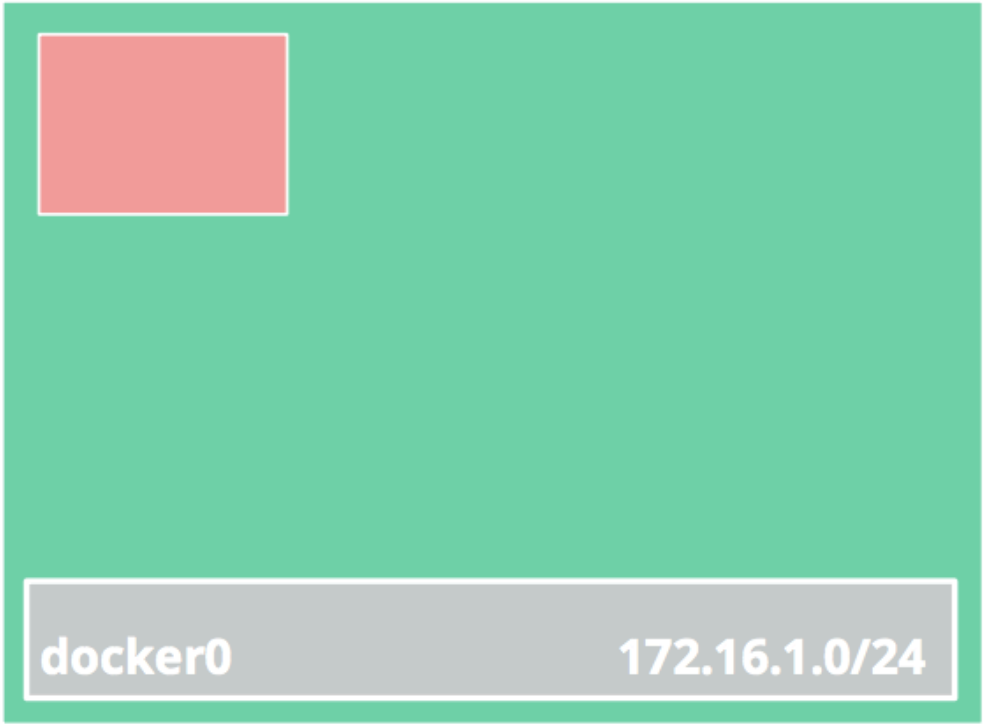
Docker Networking



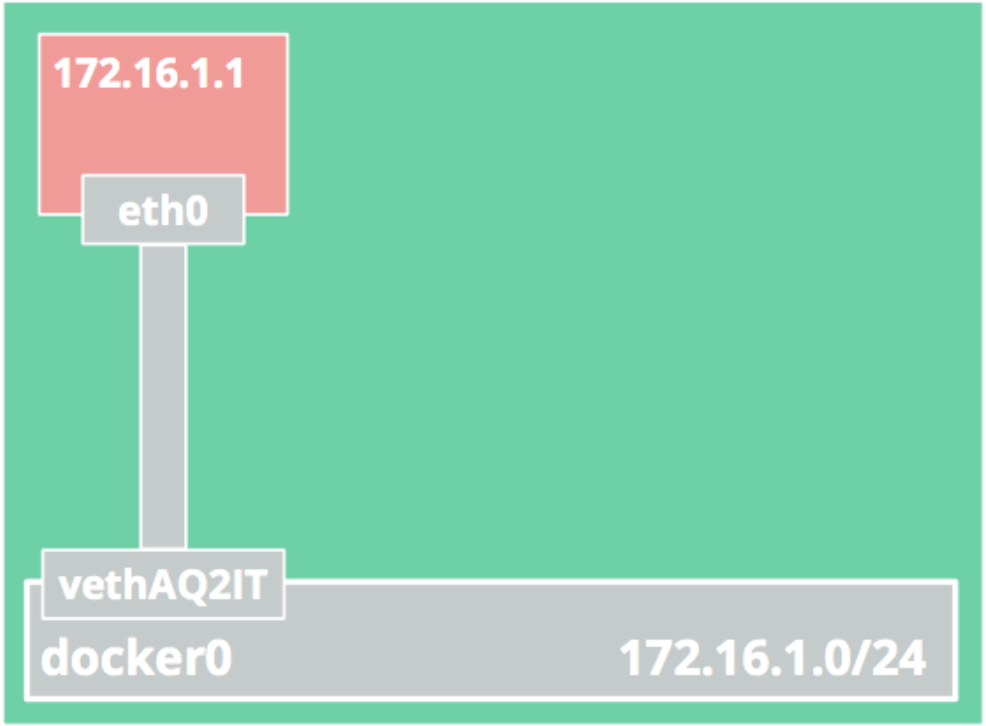
Docker Networking



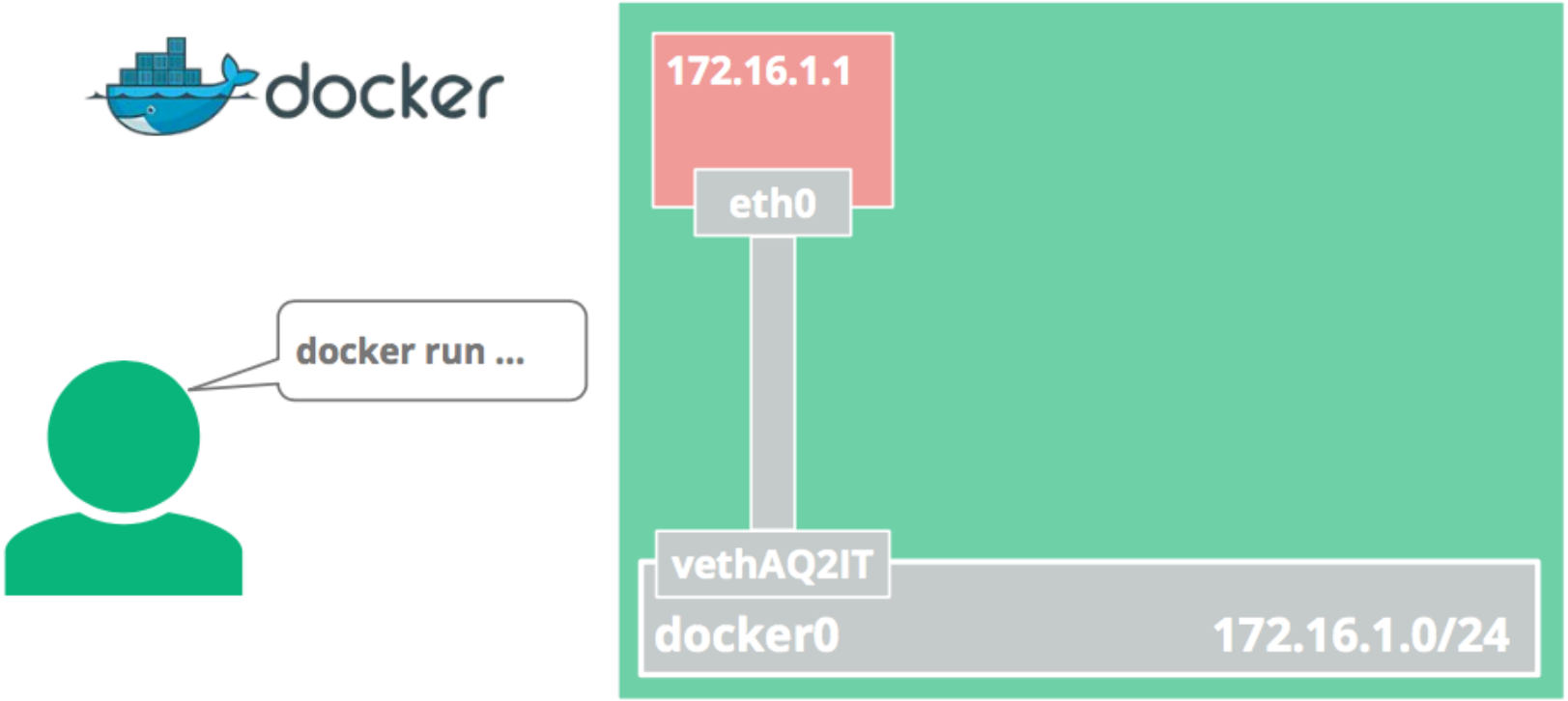
Docker Networking



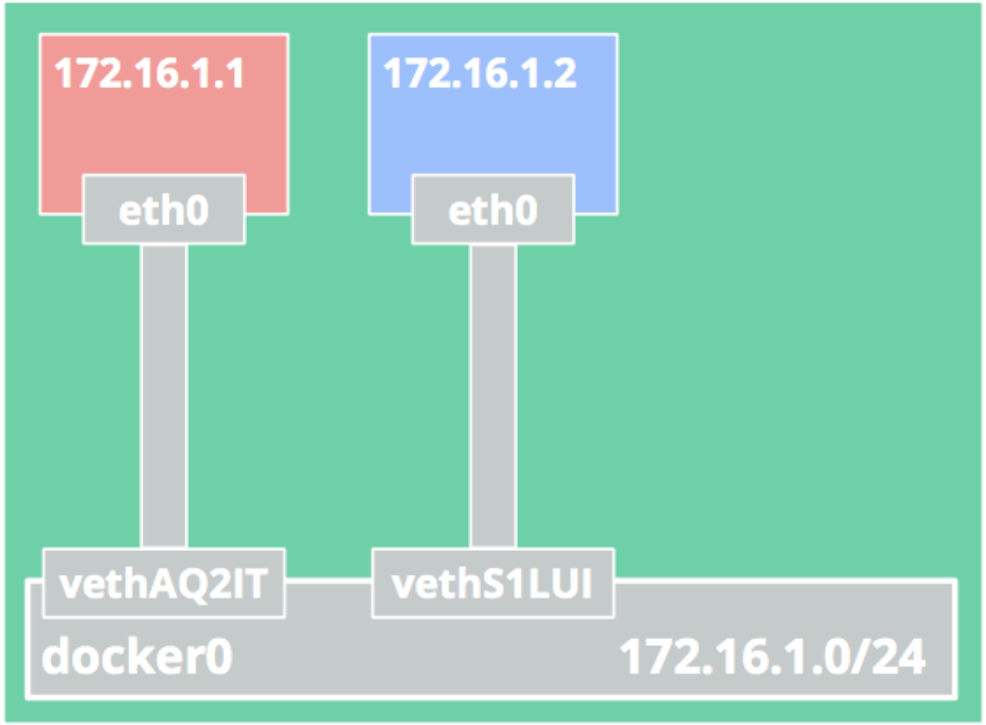
Docker Networking



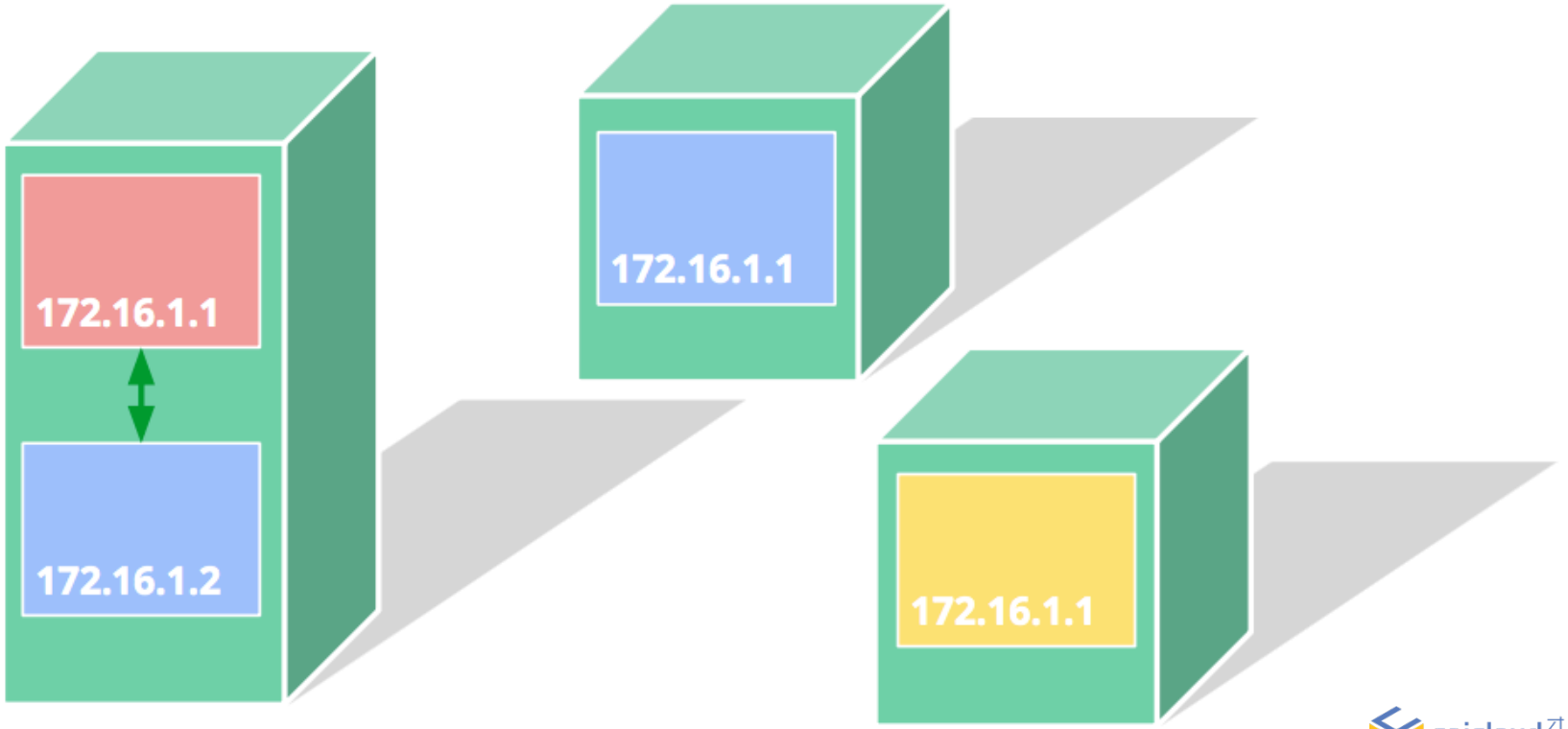
Docker Networking



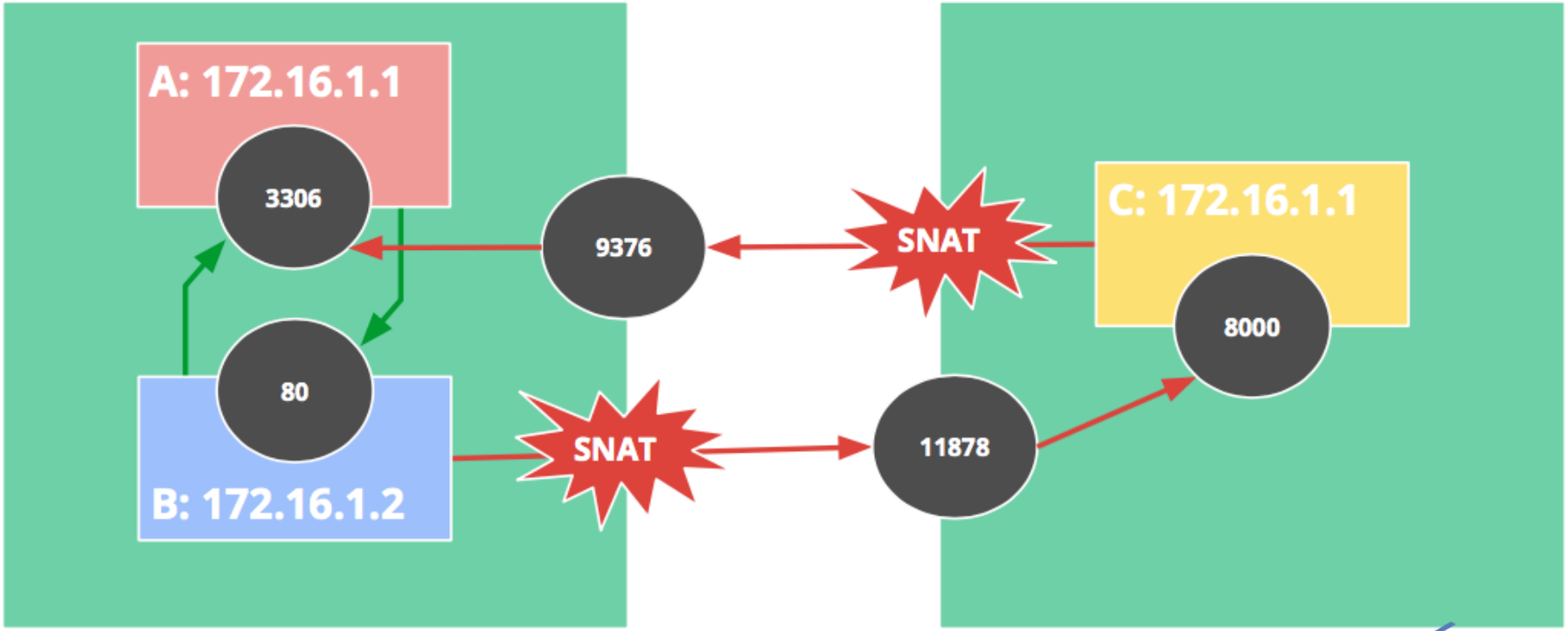
Docker Networking



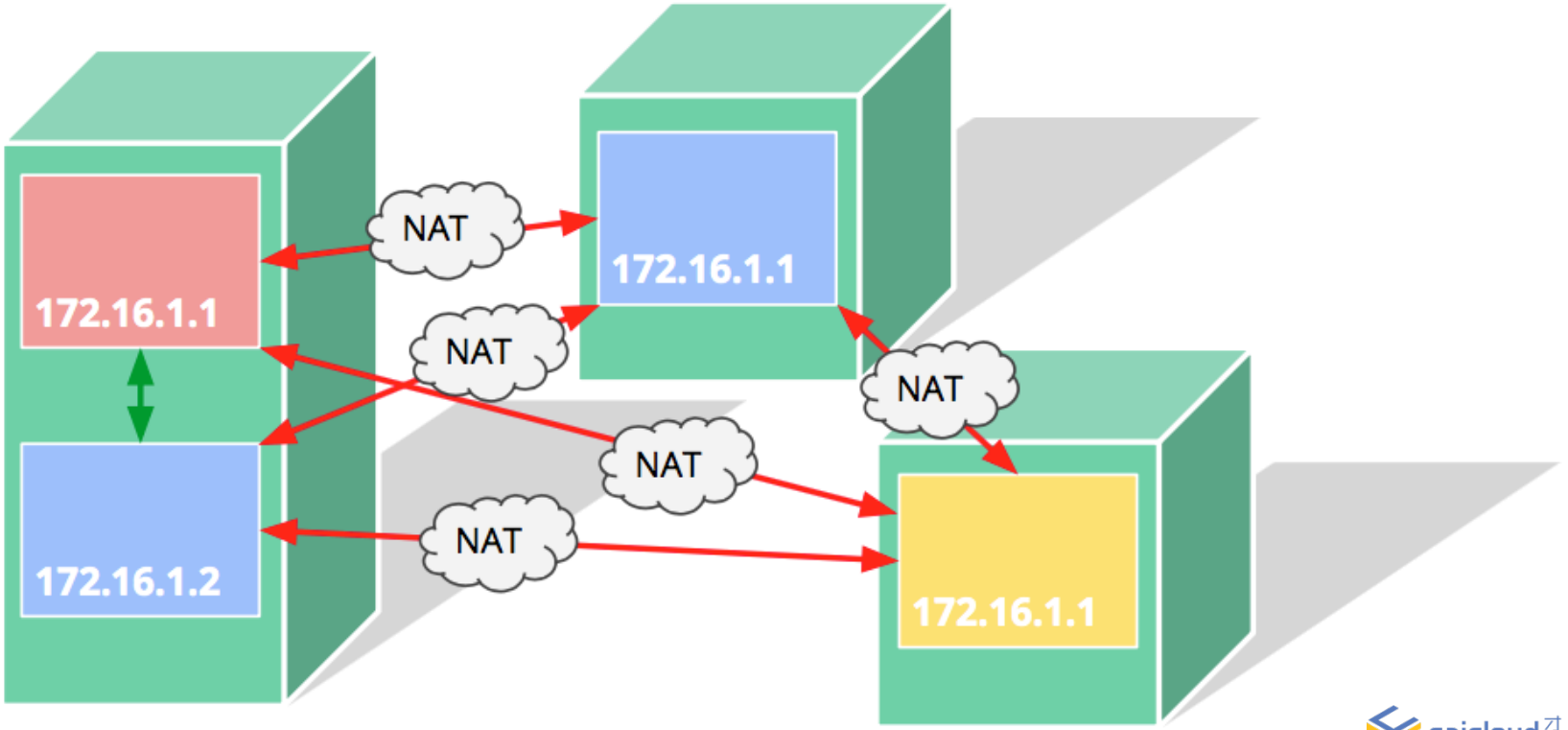
Docker Networking



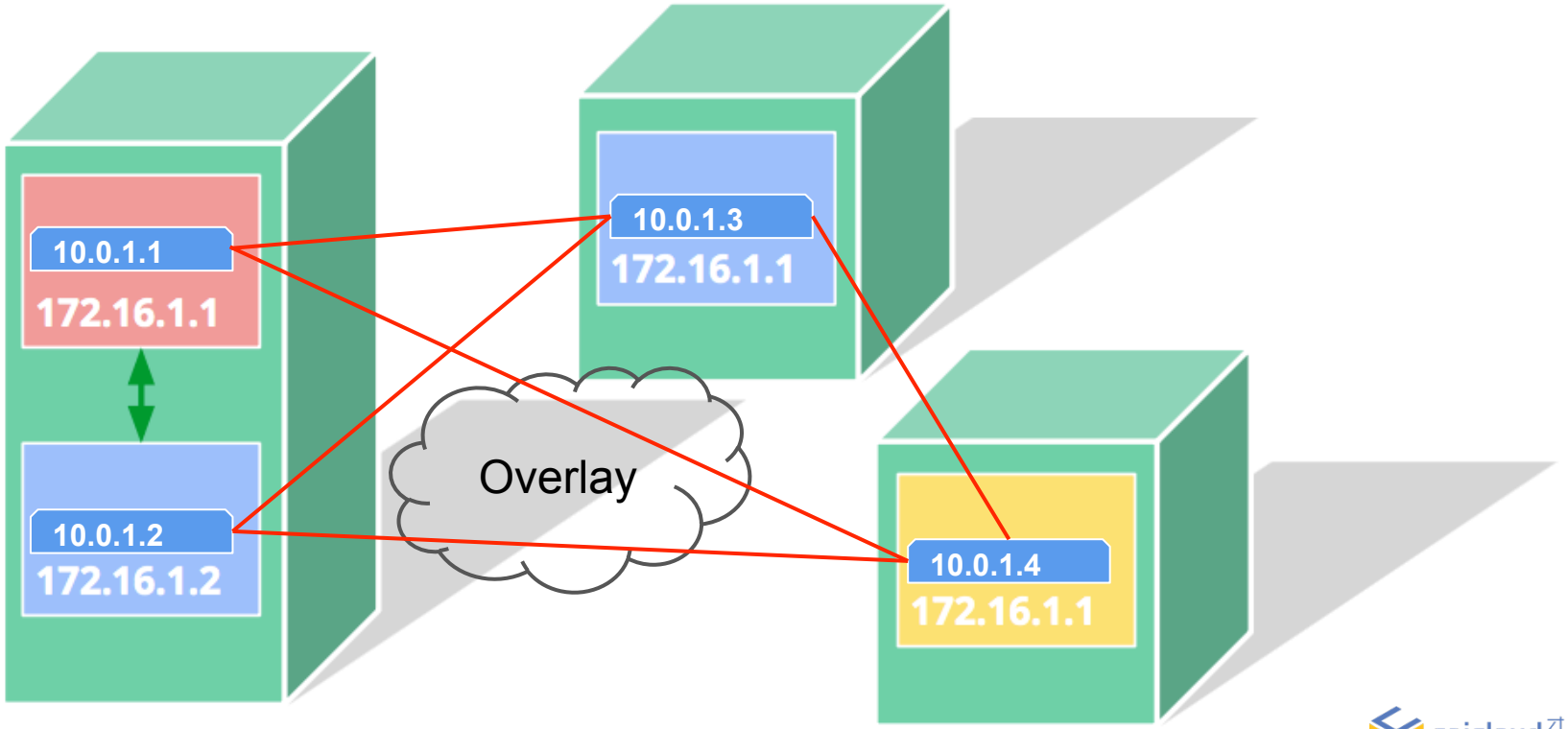
Docker Networking - HostPort



Docker Networking

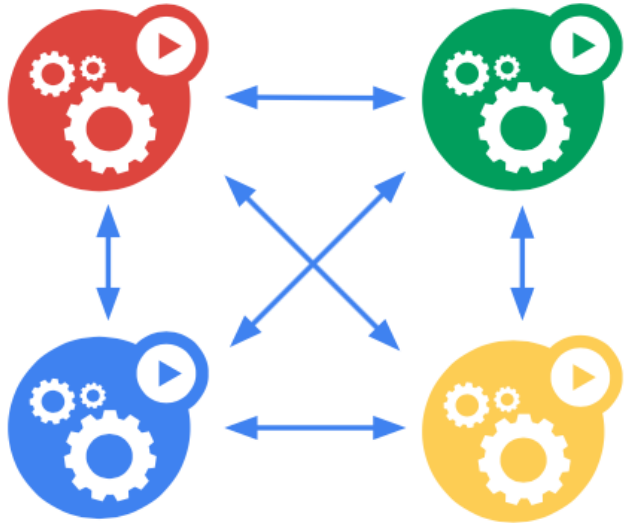


Docker Networking - Overlay

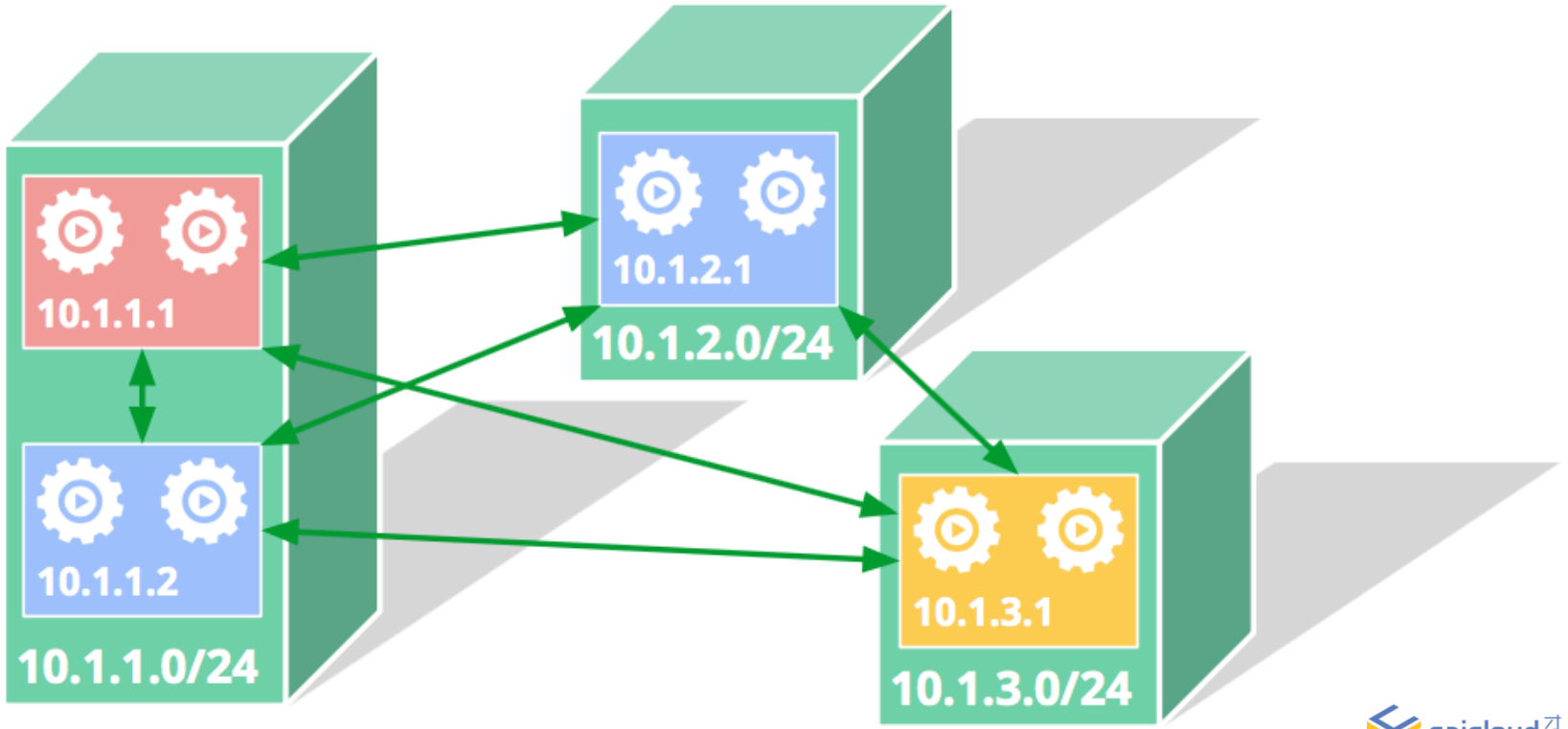


Kubernetes Networking

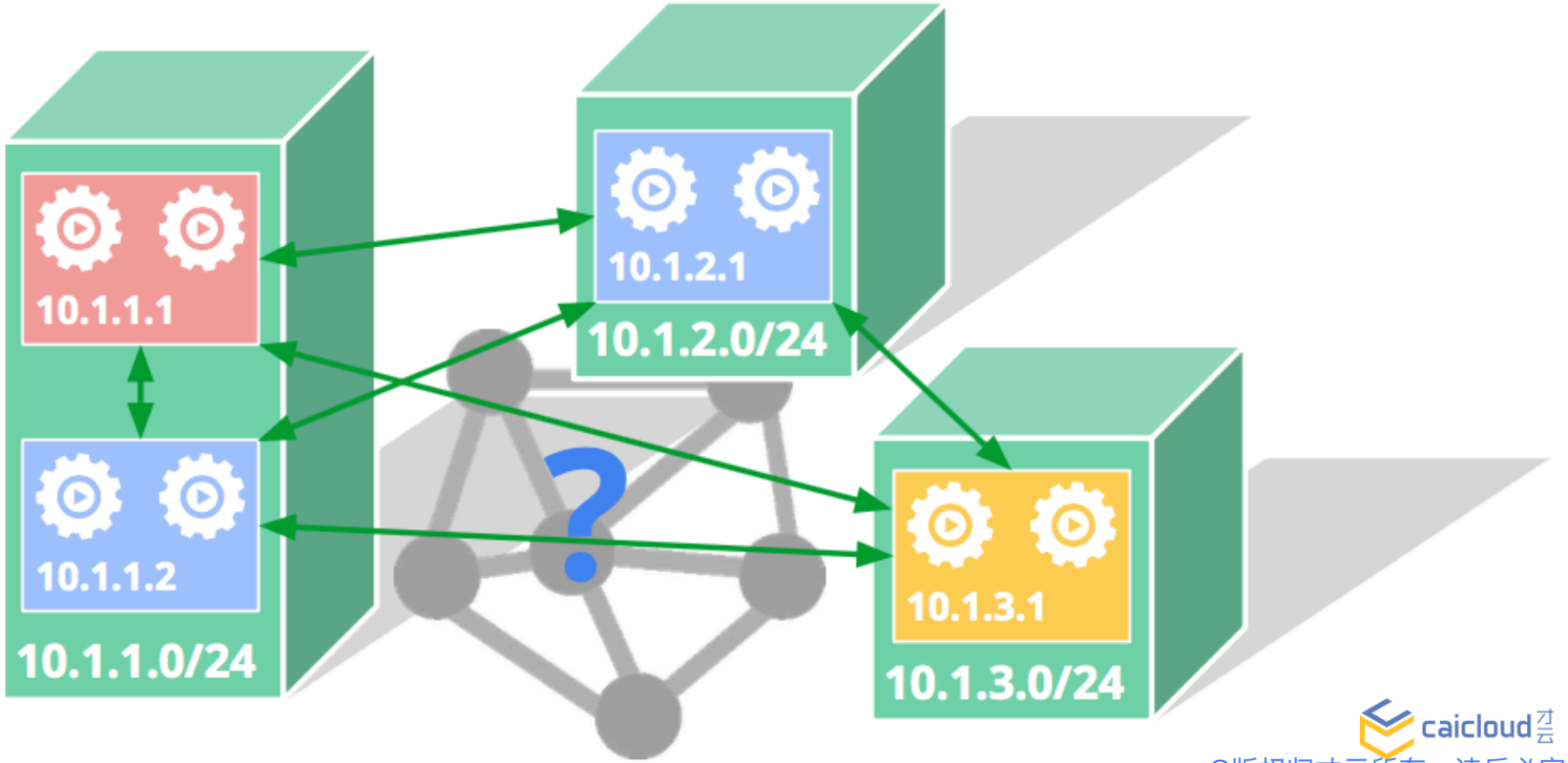
- IPs are routable
  - vs docker default private IP
- Pods can reach each other without NAT
  - even across nodes
- No brokering of port numbers
  - Too complex, why bother?
- Every host is assigned a subnet
- This is a fundamental requirement



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