容器集群管理云平台

Cluster as a service

由美国Google+AWS原生容器集群cluster management团队打造

Kubernetes Master High Availability 高级实践

才云科技 唐继元 <tangjiyuan@caicloud.io>

About Me

- Current：杭州才云科技 云开源高级工程师
- 曾华为工作5年:
    1. linux内核相关的研发 – 2年
    2. 众核OS、LibOS的相关研发 – 2年
    3. 容器相关研发 – 1年

Goals

1. 从Kubernetes架构设计看高可用
2. 社区高可用版本演进
3. HA Master整体架构
4. 核心技术点和难点
5. 实践中的遇到的那些坑
6. 社区关于HA Master的未来发展

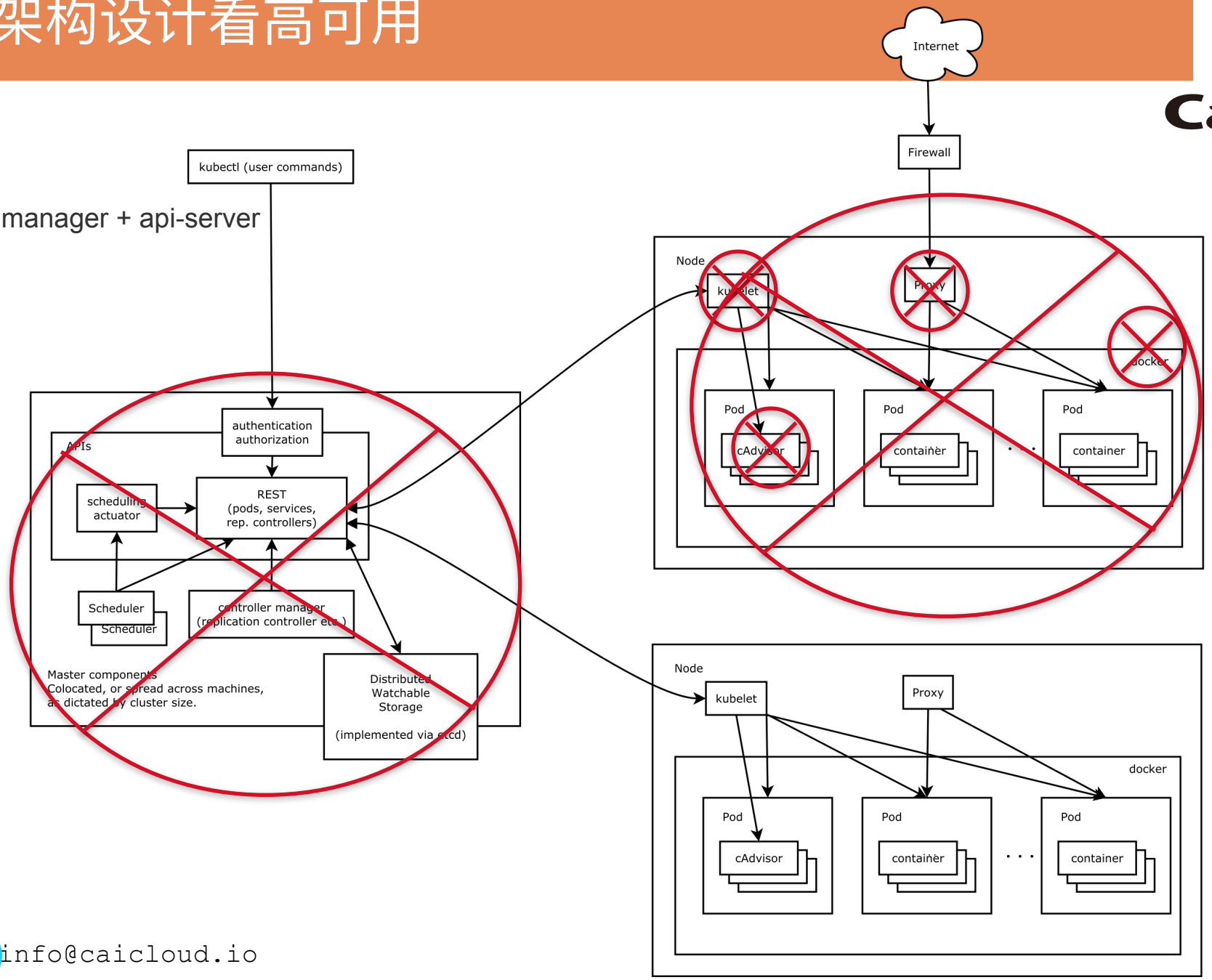
从Kubernetes架构设计看高可用

Node：kubelet + kube-proxy

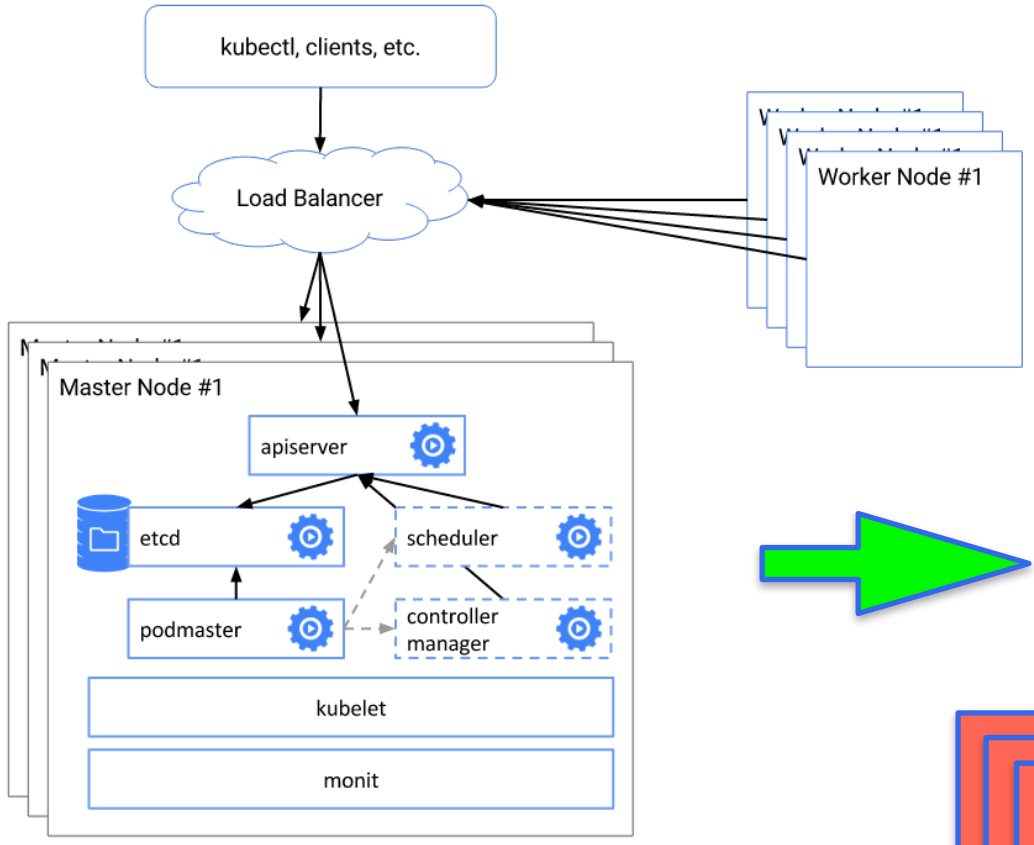
Master：scheduler + controller manager + api-server

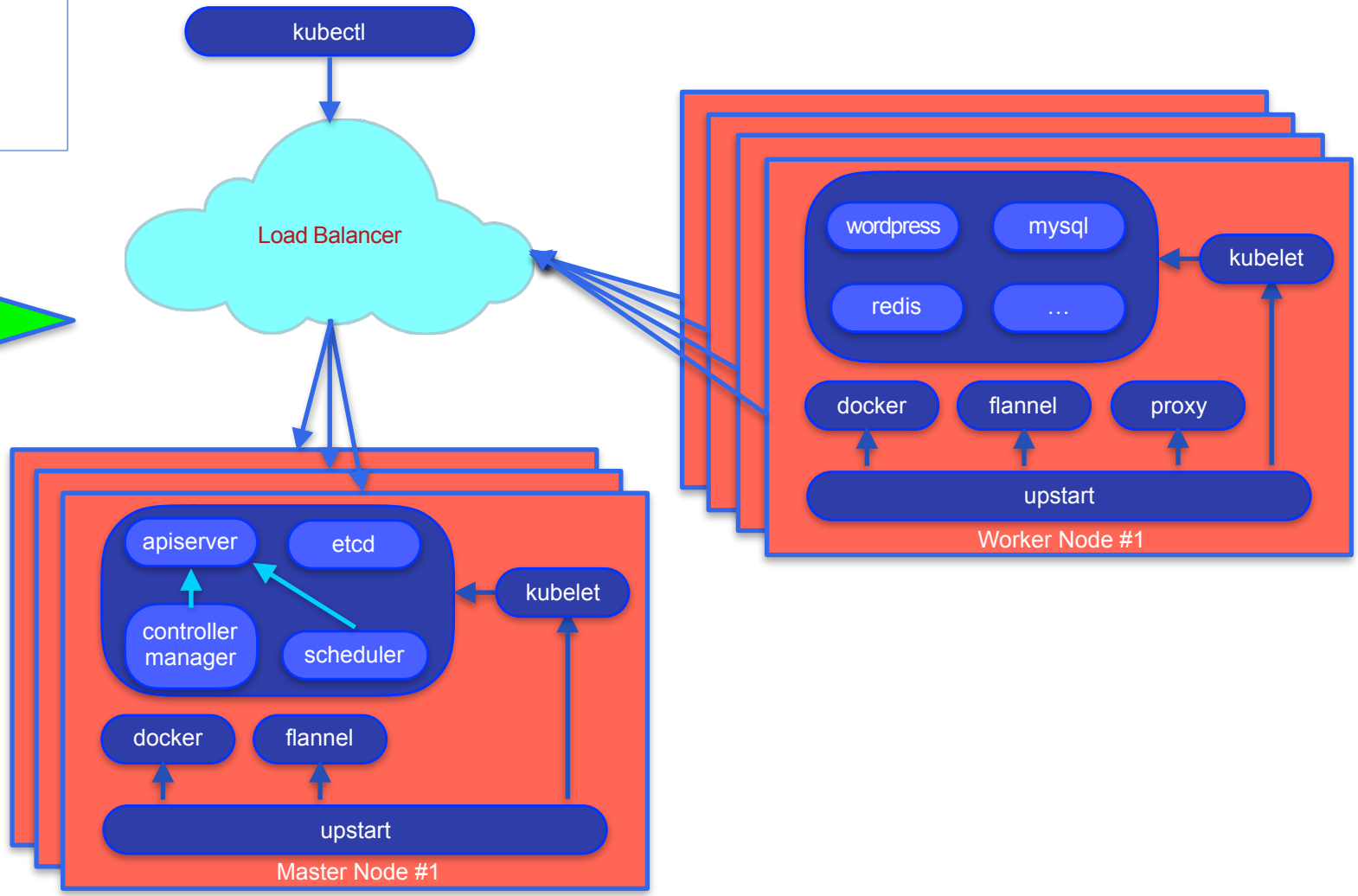
Etcd: 持久化存储

1. Pod不可用?
   RC, Scheduler
2. Node节点不可用?
   多Node节点
3. Kubelet, Proxy, Flannel, Docker不可用?
   进程监控程序
   容器化
4. Master不可用?
   多Master节点
   Master组件多副本
   Master组件容器化
   Etcd集群



社区高可用版本演进





优点

1. podmaster作为公共库
2. 减少静态pod和etc访问
3. self-hosted

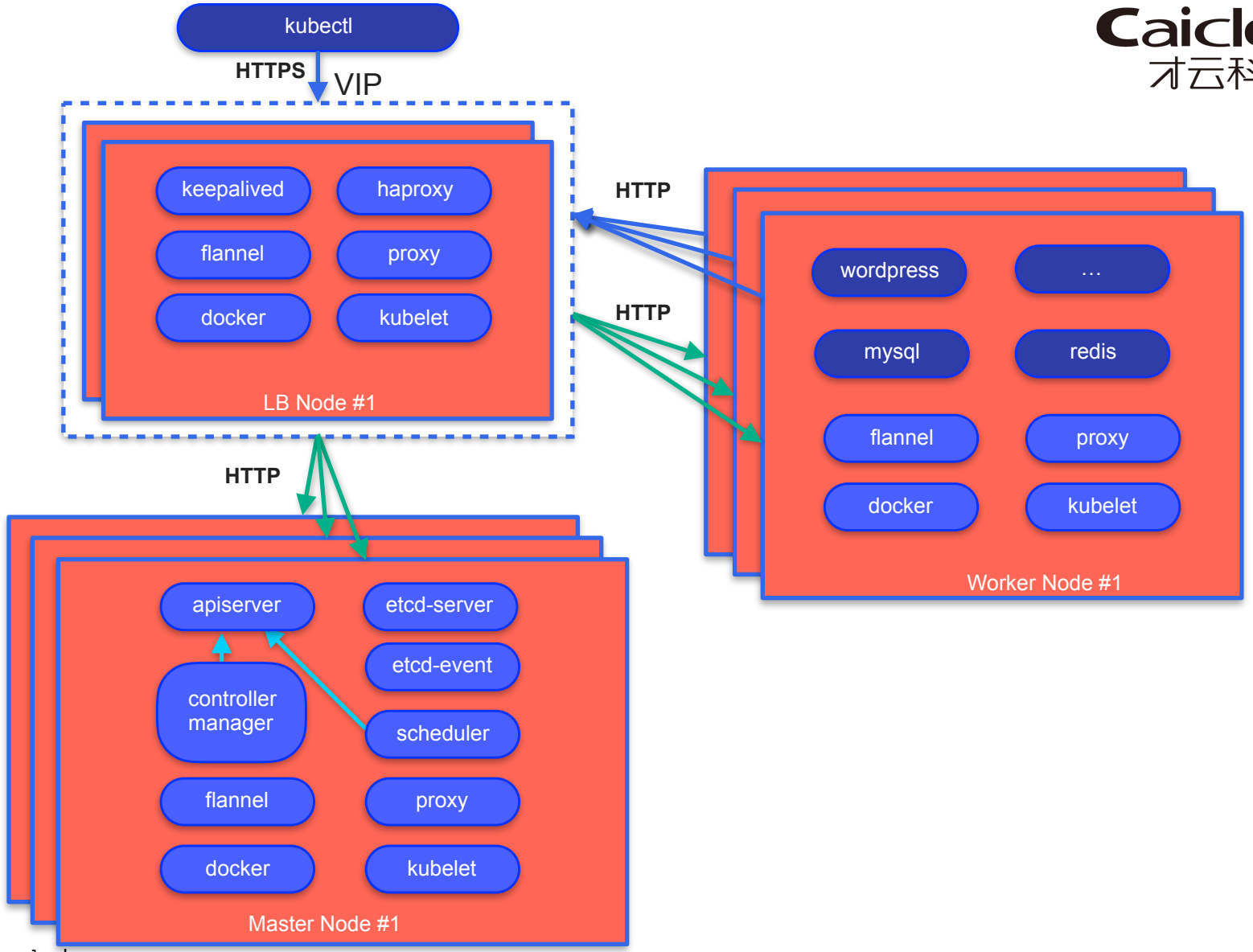
HA Master整体架构（一）

HOST	IP ADDRESS	Components
lb-1	192.168.205.252	kubelet, haproxy, keepalived, flannel
lb-2	192.168.205.253	kubelet, haproxy, keepalived, flannel
master-1	192.168.205.11	kubelet, apiserver, controller manager, scheduler, kubeproxy, etcd, flannel
master-2	192.168.205.12	kubelet, apiserver, controller manager, scheduler, kubeproxy, etcd, flannel
master-3	192.168.205.13	kubelet, apiserver, controller manager, scheduler, kubeproxy, etcd, flannel
node-1	192.168.205.21	kubelet, kubeproxy, flannel
node-2	192.168.205.22	kubelet, kubeproxy, flannel
node-3	192.168.205.23	kubelet, kubeproxy, flannel

VIP：192.168.205.254

HA Master整体架构（二）

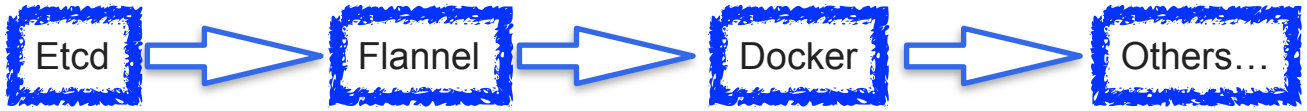
1. HA Master 组件进程形式部署
   - 可靠性保证：monit, upstart, systemd
   - 组件间依赖：调整进程启动顺序
2. 全容器形式部署
   - 可靠性保证：监控程序保证kubelet服务，kubelet保证static pod高可用
   - 组件间依赖：调整pod启动顺序



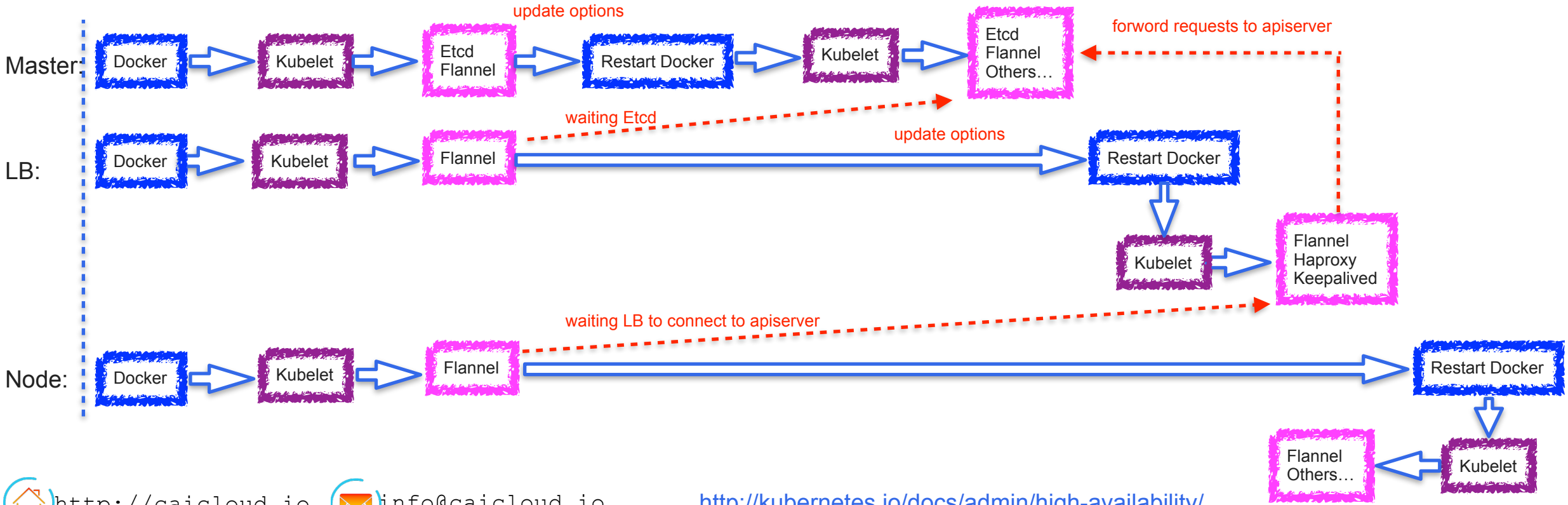
核心技术点和难点 （一：启动顺序）

Caicloud 才云科技

1. 进程形式



1. 全容器形式



http://kubernetes.io/docs/admin/high-availability/

核心技术点和难点（二：运行在特权模式的组件）

1. 开启Kubernetes集群的特权模式权限

```
--allow-privileged=true
```

A. Kubelet

- 允许docker容器向kubelet请求以特权模式运行

B. Apiserver

- 允许docker容器能够访问apiserver

2. 运行在特权模式下的docker容器

```
securityContext:
    privileged: true
```

A. Kubeproxy static pod

- 通过Iptables设置防火墙规则

B. Flannel static pod

- 访问vxlan、openvswitch等路由数据报文

A. Keepalived static pod

- 访问IP_VS内核模块来建立VIP

核心技术点和难点（三：组件pod必须运行在主机网络）

Caicloud
才云科技

- 以static pod形式存在的Kubernetes集群组件必须运行在主机网络下：

  hostNetwork: true

  理由：

  A. 心跳和信息交流通过它们配置文件中的静态IP地址进行

  flannel提供动态网络

  B. kubeproxy、flannel、haproxy需要通过主机网络修改路由规则

  C. haproxy需要将外网请求重定向到内网后端服务器上

核心技术点和难点（四：External Loadbalancer部署）

- 通过haproxy和keepalived pod实现Master的负载均衡，对外提供统一的VIP
- haproxy和keepalived可以放在同一个pod
  - 普通进程形式：killall -0 haproxy
  - 容器模式：通过haproxy的健康检查页面监控状态
- haproxy SSL配置
  - haproxy本身只提供代理：仅支持4层代理
  - haproxy实现SSL Termination proxy

实践中的遇到的那些坑（官网Haproxy镜像的坑）

Caicloud 才云科技

- “haproxy image”的“docker-entrypoint.sh”：

```
#!/bin/sh
set -e

# first arg is `-f` or `--some-option`
if [ "${1#-}" != "$1" ]; then
    set -- haproxy "$@"
fi

if [ "$1" = 'haproxy' ]; then
    # if the user wants "haproxy", let's use "haproxy-systemd-wrapper"
    # instead so we can have proper reloadability implemented by upstream
    shift # "haproxy"
    set -- "$(which haproxy-systemd-wrapper)" -p /run/haproxy.pid "$@"
fi

exec "$@"
```

```
-Ds passe en daemon systemd
 This patch adds a new option "-Ds" which is exactly like "-D", but instead of
 forking n times to get n jobs running and then exiting, prefers to wait for all the
 children it just created. With this done, haproxy becomes more systemd-compliant,
 without changing anything for other systems.
```

```
containers:
  - name: lb-haproxy
    image: index.caicloud.io/caicloud/haproxy:v1.6.5
    command:
    - haproxy                    /usr/local/sbin/haproxy
    - -f
    - /etc/haproxy/haproxy.cfg
    - -p
    - /run/haproxy.pid
  - name: lb-keepalived
    image: index.caicloud.io/caicloud/keepalived:v1.2.19
    command:
    - keepalived
    - --log-console
    - --dont-fork
    - -f
    - /etc/keepalived/keepalived.conf
```

社区关于HA Master的未来发展（一）

Caicloud 才云科技

- —api-servers配置
  - kubelet配置apiserver，通过“—api-servers”指定多个：—api-servers=http://m1b:8080,http://m1c:8080,http://m2a:8080,http://m2b:8080,http://m2c:8080，但是只有第一个起作用
- —master配置
  - controller manager和scheduler：只能通过“—master”配置一个apiserver，无法支持多个apiserver

- 参考链接：

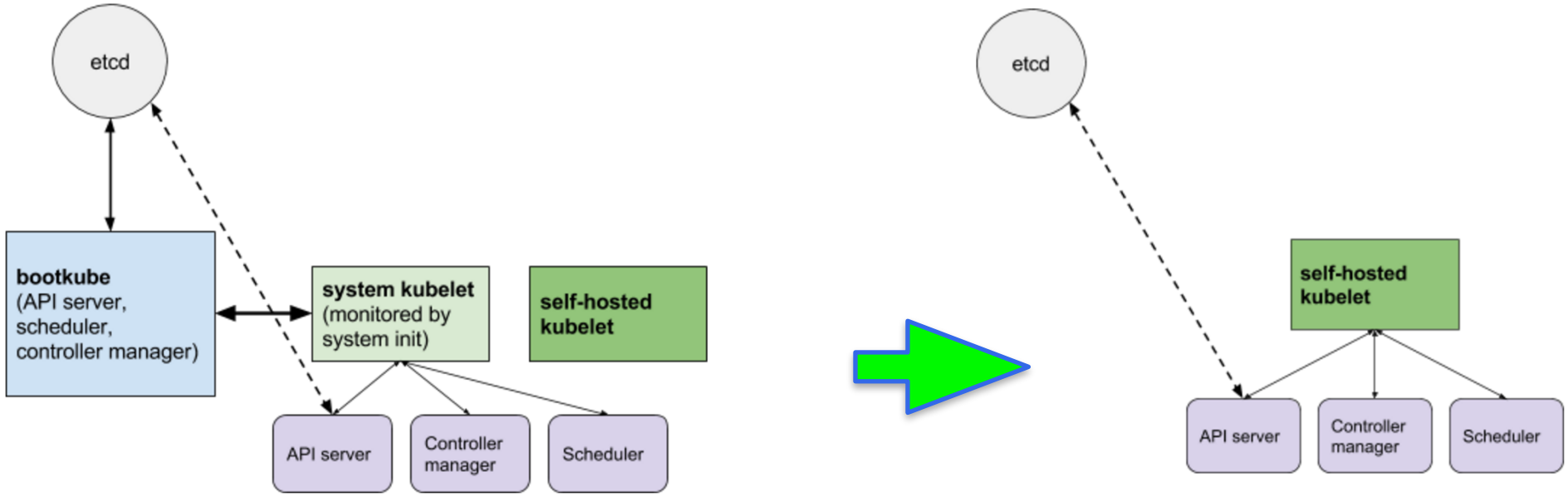
A. https://github.com/kubernetes/kubernetes/issues/26852

B. https://github.com/kubernetes/kubernetes/pull/25428

社区关于HA Master的未来发展（二）

Caicloud 才云科技

- self-hosted install/update design with bootkube
  - self-hosted：runs all required and optional components of a Kubernetes cluster on top of Kubernetes itself.



- 参考链接：

A. https://docs.google.com/document/d/1VNp4CMjPPHevh2_JQGMl-hpz9JSLq3s7HlI87CTjl-8/edit

B. https://groups.google.com/forum/#!topic/kubernetes-sig-cluster-ops/li_brwXYeCI

C. https://github.com/philips/kubernetes/blob/ebcde947994e85488f1511dfcae0295e2a6bd67e/docs/proposals/self-hosted-kubelet.md#proposal

Caicloud
才云科技

DBAplus线上分享实录 | Kubernetes Master High Availability 高级实践

原创 2016-06-29 才云 唐继元 Caicloud

才云科技云开源高级工程师唐继元受邀DBAplus社群，在线分享《Kubernetes Master High Availability 高级实践》，介绍如何构建Kubernetes Master High Availability环境。

http://mp.weixin.qq.com/s?__biz=MzIzMzExNDQ3MA==&mid=2650091772&idx=1&sn=727c986f602e4de6ad6a2cf66a45aa89#rd

Kubernetes高级实践：Master高可用方案设计和踩过的那些坑

唐继元 2016年06月30日

HA Master的整体架构、技术难点和实践遇到的坑，以及社区对HA Master的企望。

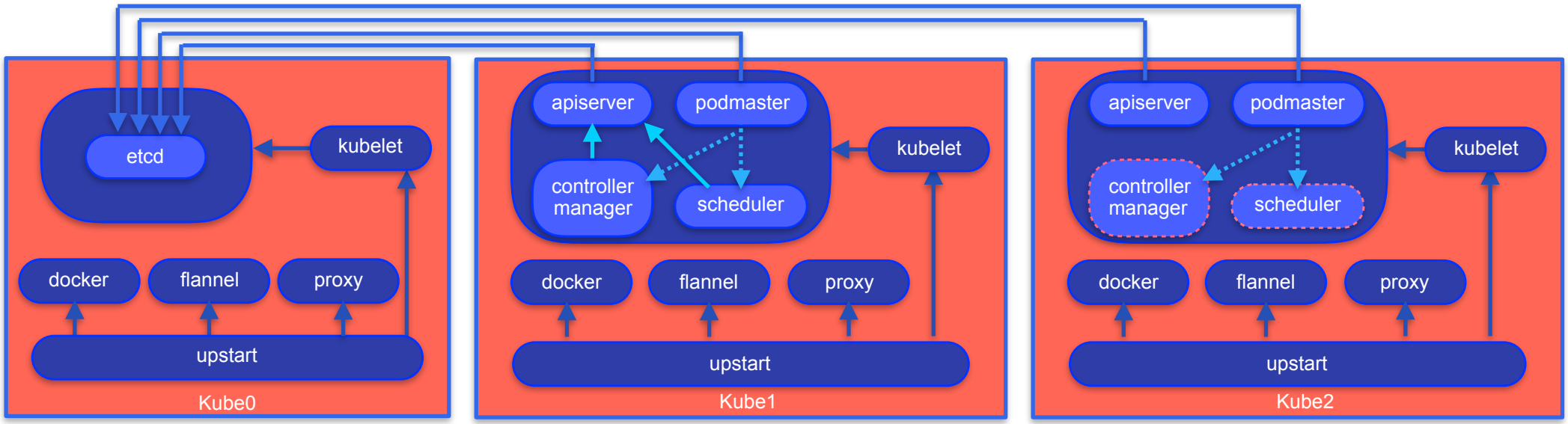
http://dbaplus.cn/news-21-499-1.html

Caicloud
才云科技

Thank you!

http://caicloud.io info@caicloud.io

Kubernetes High Availability V1




```
func (c *config) leaseAndUpdateLoop(etcdClient *etcd.Client) {
  for {
    master, err := c.acquireOrRenewLease(etcdClient)
    c.update(master);
    time.Sleep(c.sleep)
  }
}

func (c *config) update(master bool) error {
  switch {
  case master && !exists:
    return copyFile(c.src, c.dest)
  case master && exists && sum(c.src) != sum(c.dest):
    return copyFile(c.src, c.dest)
  case !master && exists:
    return os.Remove(c.dest)
  }
}

func main() {
  etcdClient, err := etcd.New(cfg)
  leaseAndUpdateLoop(&etcdClient)
}
```

分析

1. apiserver多副本? stateless
2. scheduler多副本? only one is active
   controller manager多副本? only one is active

优点

1. Master三大组件高可用
2. 容器化

缺点

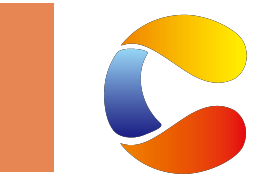
1. Etcd 单点
2. Podcaster 开销

```
"containers":[
    {
    "name": "scheduler-elector",
    "image": "gcr.io/google_containers/podmaster:1.1",
    "command": [
                "/podmaster",
                "--etcd-servers=http://127.0.0.1:4001",
                "--key=scheduler",
                "--source-file=/kubernetes/kube-scheduler.manifest",
                "--dest-file=/manifests/kube-scheduler.manifest"
              ],
    ...
    },
    {
    "name": "controller-manager-elector",
    "image": "gcr.io/google_containers/podmaster:1.1",
    "command": [
                "/podmaster",
                "--etcd-servers=http://127.0.0.1:4001",
                "--key=controller",
                "--source-file=/kubernetes/kube-controller-manager.manifest",
                "--dest-file=/manifests/kube-controller-manager.manifest"
              ],
    ...
    }
],
```

https://github.com/kubernetes/contrib/tree/master/pod-master

https://github.com/kubernetes/kubernetes/tree/release-1.1/examples/high-availability

Kube-controller-managerment self-hosted 源码分析



/* cmd/kube-controller-manager/app/controllermanager.go */

```
// Run runs the CMServer.  This should never exit.
func Run(s *options.CMServer) error {
    ...
    run := func(stop <-chan struct{}) {
        err := StartControllers(s, kubeClient, kubeconfig, stop)
        glog.Fatalf("error running controllers: %v", err)
        panic("unreachable")
    }
    // --leader-elect选项未配置则直接启动controllers
    if !s.LeaderElection.LeaderElect {
        run(nil)
        panic("unreachable")
    }
    // 否则进入选举流程
    // 参与选举
    leaderelection.RunOrDie(leaderelection.LeaderElectionConfig{
        EndpointsMeta: api.ObjectMeta{
            Namespace: "kube-system",
            Name:      "kube-controller-manager",
        },
        Client:        kubeClient,
        Identity:      id,
        EventRecorder: recorder,
        LeaseDuration: s.LeaderElection.LeaseDuration.Duration,
        RenewDeadline: s.LeaderElection.RenewDeadline.Duration,
        RetryPeriod:   s.LeaderElection.RetryPeriod.Duration,
        Callbacks: leaderelection.LeaderCallbacks{
            OnStartedLeading: run,  // 选举成功，启动controllers
            OnStoppedLeading: func() {  // 选举失败
                glog.Fatalf("leaderelection lost")
            },
        },
    })
    panic("unreachable")
}
```

/* pkg/client/leaderelection/leaderelection.go */

```
// RunOrDie starts a client with the provided config
// or panics if the config
// fails to validate.
func RunOrDie(lec LeaderElectionConfig) {
    le, err := NewLeaderElector(lec)
    if err != nil {
        panic(err)
    }
    le.Run() // 选举过程
}

// Run starts the leader election loop
func (le *LeaderElector) Run() {
    defer func() {
        runtime.HandleCrash()
        le.config.Callbacks.OnStoppedLeading()
    }()

    // 周期性 acquire leader lease，直到 get it successfully
    le.acquire()
    // 获取到 a leader lease 后启动 controllers
    stop := make(chan struct{})
    go le.config.Callbacks.OnStartedLeading(stop)
    // 周期性的 renew leader lease
    // 除非 failed to renew leader lease
    le.renew()
    close(stop)
}
```

Caicloud Kubernetes High Availability版本一

Caicloud 才云科技

LB特点

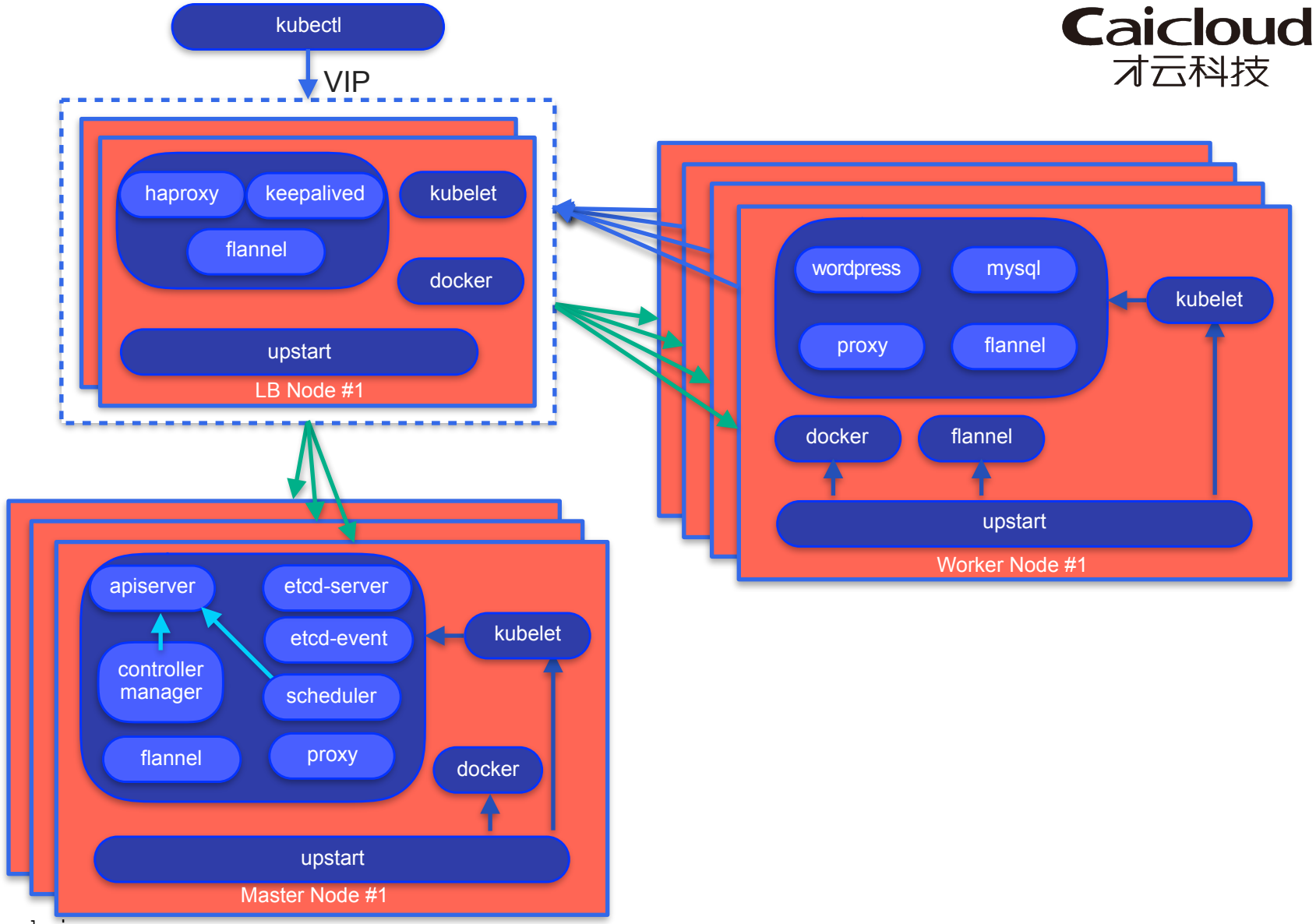
1. 支持HA MASTER
2. K8S集群组件容器化，可移植性高
3. 统一访问入口
4. 支持NodePort的负载均衡
5. 容器化

KeepAlived

1. 保障Haproxy的高可用
2. 提供VIP

Haproxy

1. 提供基于TCP和HTTP应用的代理
2. IP, Session亲和性
3. 基于pod livenessProbe的健康检查



Load Balancing

Caicloud
才云科技

Internal

- Kube-proxy

External

- NodePort
- LoadBalancer
- External IPs
- Ingress