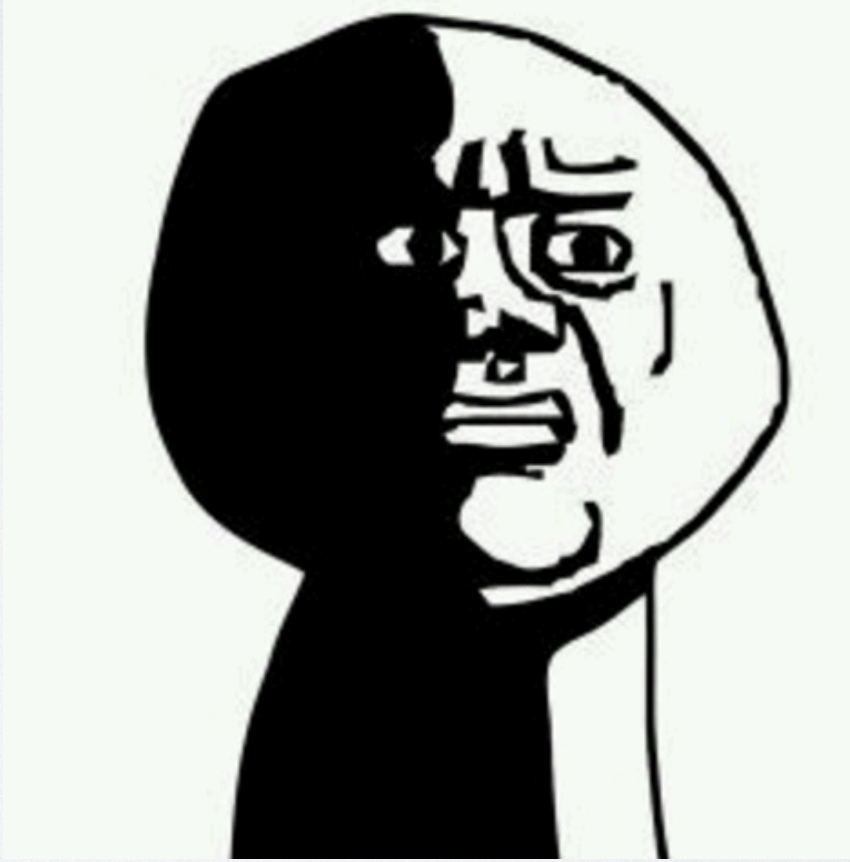
GO在小米商城运维平台的应用与实践

高步双

小米商城运维：忆往昔

- 大大小小的业务项目繁多
- 需求多，运维被动，被业务牵着走
- 业务混部，问题难以排查，容量难以评估
- 运维工具级别：puppet、salt stack



MAE

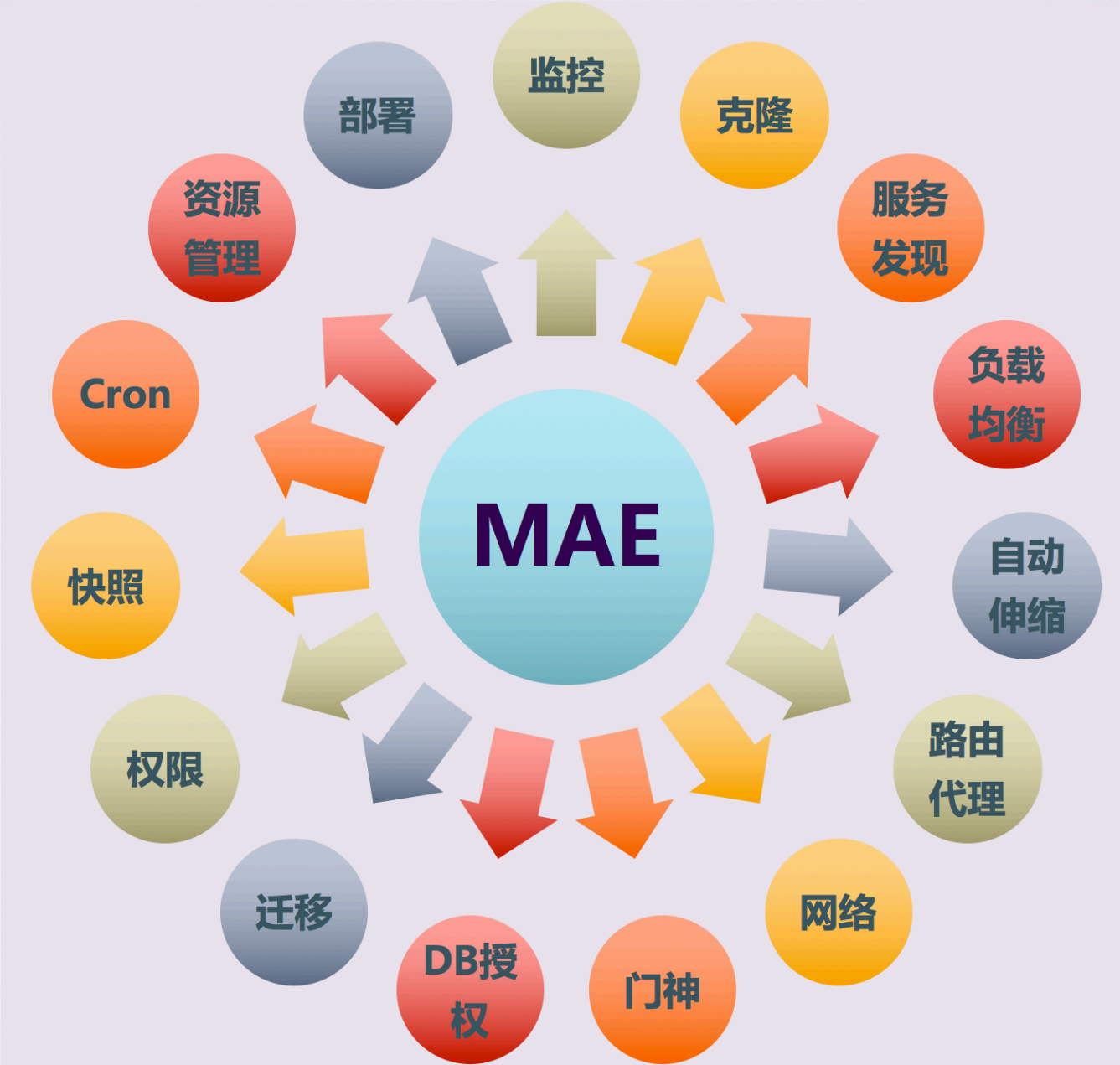
MAE（小米商城应用引擎）

小米商城运维团队基于docker自研的私有云PaaS平台，目的是解决传统混部在资源管理、资源隔离、资源利用率、部署、资源调度、故障容错、监控等方面的问题，解放人力。支持混合云。

MAE – 为何选择GO？

GO足够简单容易上手，具有脚本动态语言的开发效率，静态语言的性能，并发协程，内置GC等等。所有Server端模块、所有Agents、Micron以及Router/Monitor均采用Go语言开发。

MAE – 功能



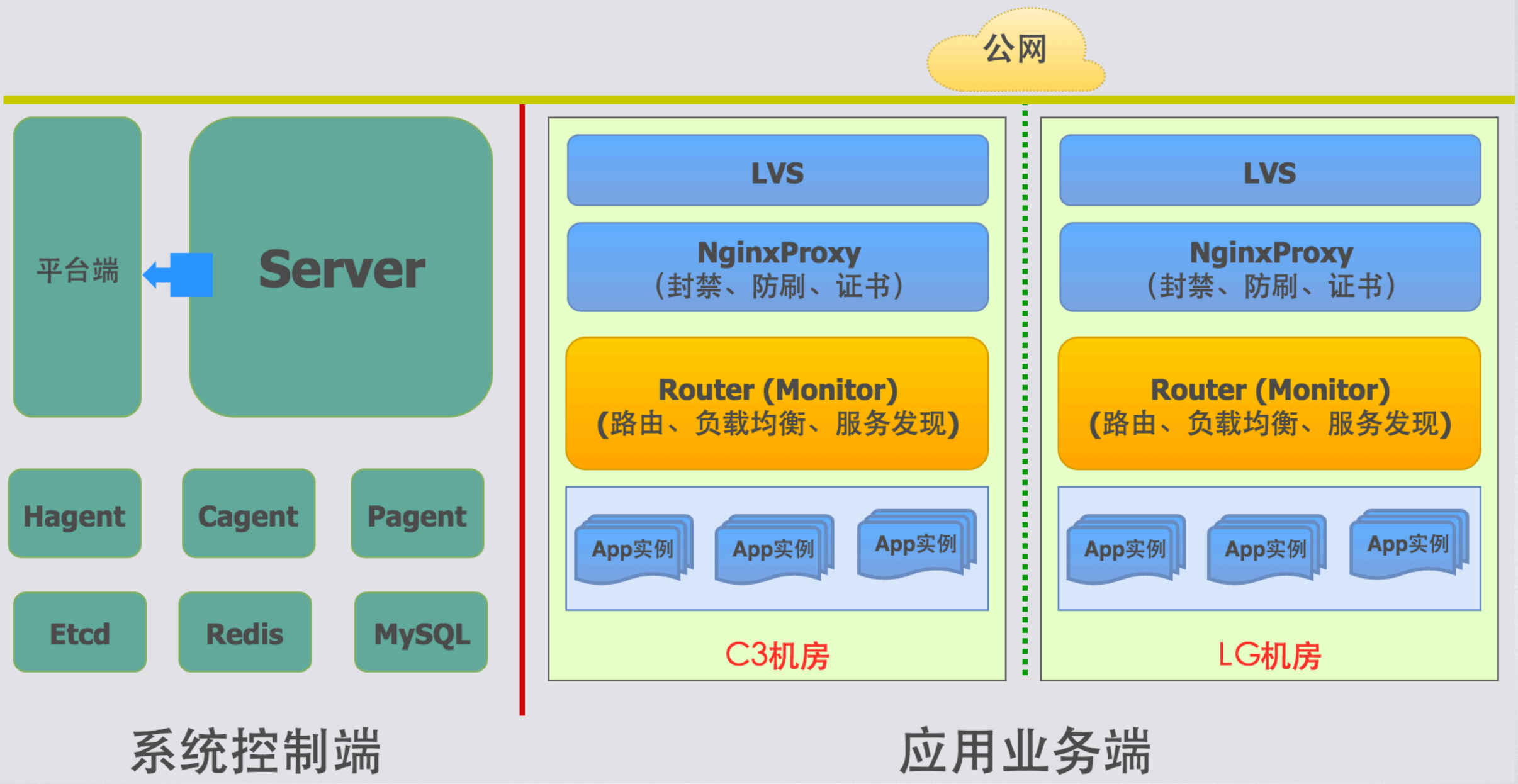
MAE – 概况

实例	≈ 800 个
应用	≈ 150 个
内存	≈ 10 TB
CPU	≈ 2000 核
资源池	≈ 250 台

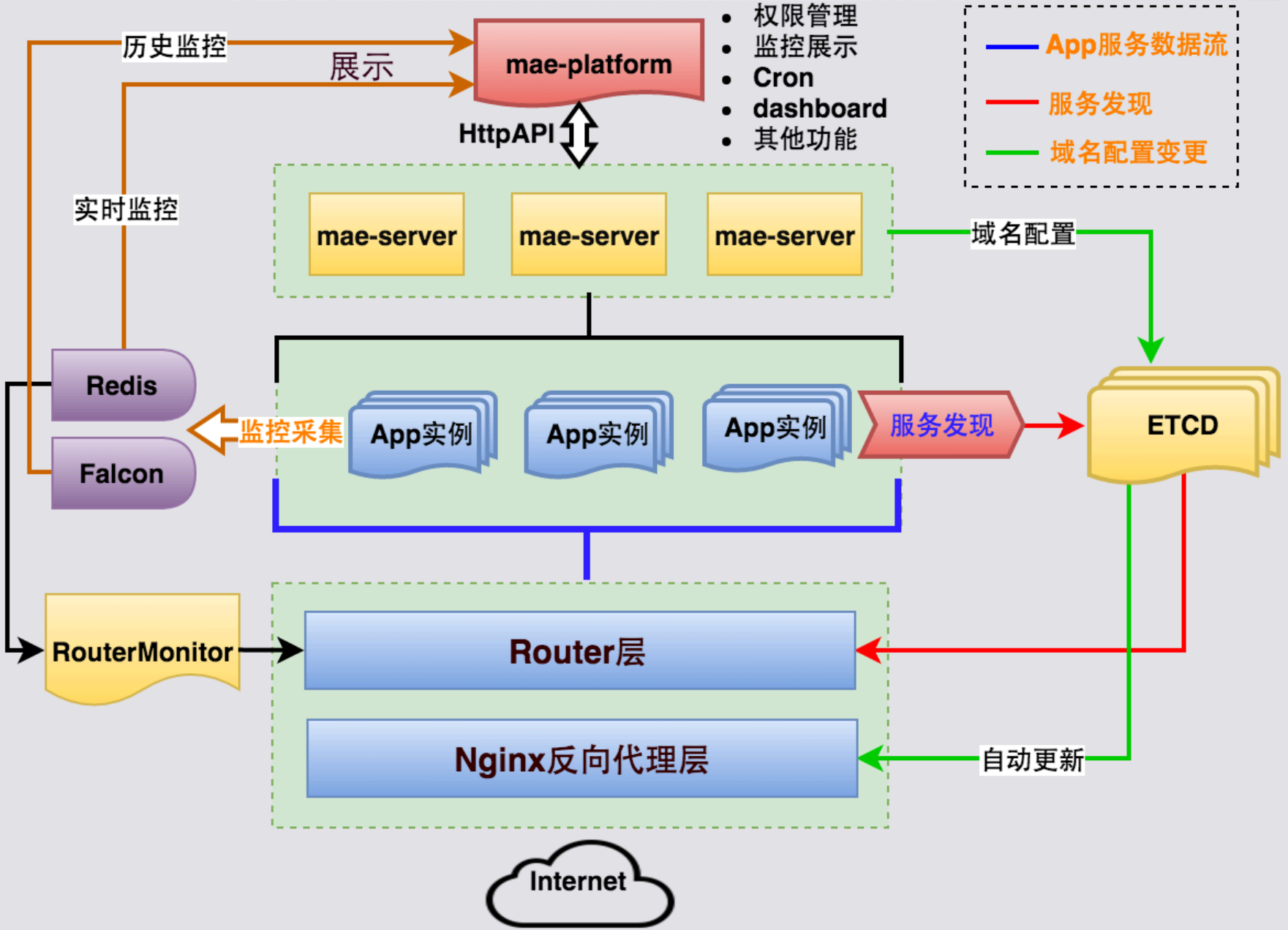
MAE – Docker

Docker版本	docker-1.6.2.2
宿主机OS	CentOS-7.0 + Kernel-4.1 + AUFS
内存	HardLimit + SwapOff
CPU	HardLimit + CpuSet (最小粒度0.5核心)
网卡流量	500Mb
系统资源限制	最大打开文件描述符数量等
网络	none: 自定义网络，LXC桥接，独立IP

MAE – 结构图


公网
平台端
Server
Hagent
Cagent
Pagent
Etcd
Redis
MySQL
LVS
NginxProxy
（封禁、防刷、证书）
Router (Monitor)
(路由、负载均衡、服务发现)
App实例
App实例
App实例
C3机房
LVS
NginxProxy
（封禁、防刷、证书）
Router (Monitor)
(路由、负载均衡、服务发现)
App实例
App实例
App实例
LG机房
系统控制端
应用业务端

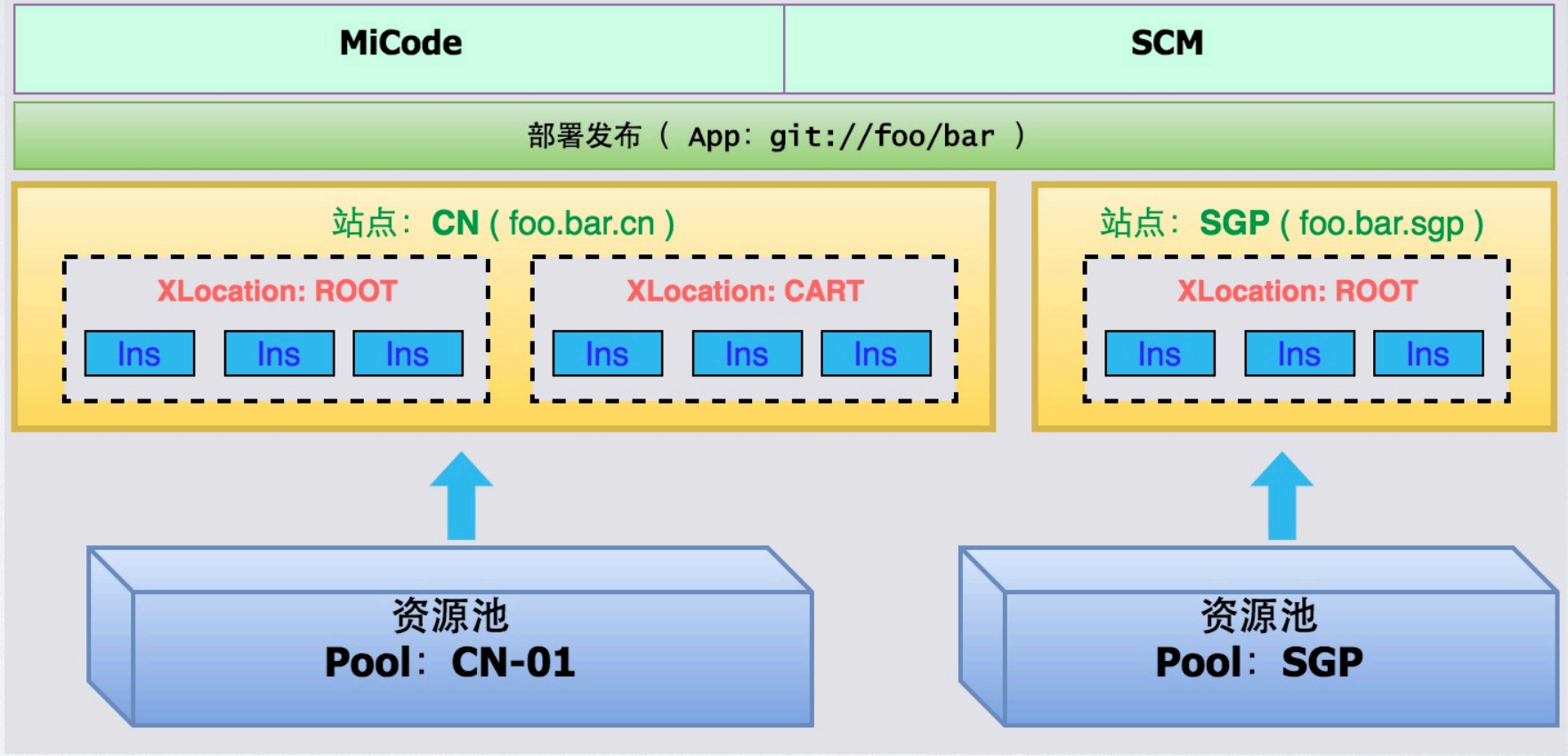
MAE – 拓扑


历史监控
展示
mae-platform
权限管理
监控展示
Cron
dashboard
其他功能
HttpAPI
App服务数据流
服务发现
域名配置变更
实时监控
mae-server
mae-server
mae-server
域名配置
Redis
监控采集
App实例
App实例
App实例
服务发现
ETCD
Falcon
RouterMonitor
Router层
Nginx反向代理层
自动更新
Internet

MAE – 部署

实例创建、应用发布

- 项目App
- 站点Site
- 功能组XLocation
- 实例
- 资源池



MAE – 部署


确定要创建实例?
创建 实例 可以快速的拓展你的业务.
站点信息:
当前站点: cn
机房信息:
当前机房: C3机房
镜像信息:
XLOCATION:
当前LOCATION: ROOT
资源模板
微型
超微型
小型
中型
大型
相同物理机上容器最大数:
2
可创建实例数量 10
0
确定创建实例
是否克隆当前实例?
克隆当前实例需要时间较长,请耐心等待.
选择机房
当前机房:选择部署位置
资源类型
超微型
微型
小型
中型
大型
XLOCATION
当前LOCATION: 无
可克隆数量为 (0)
确认克隆
C3机房
鲁谷机房
C3机房
Ip
是否在线状态
上线
存活
CN
4
迁移
销毁

MAE – 监控

分类和特点：

实时监控

- 秒级，实时性高，
- 最多24小时数据，数据量小、容量小
- 重要程度较高（监控、自动负载调权、Autoscaling）

历史监控

- 长期监控数据存储、数据量大、容量大
- 监控数据归并，实时性差
- 重要程度较低（问题追踪）

MAE – 监控

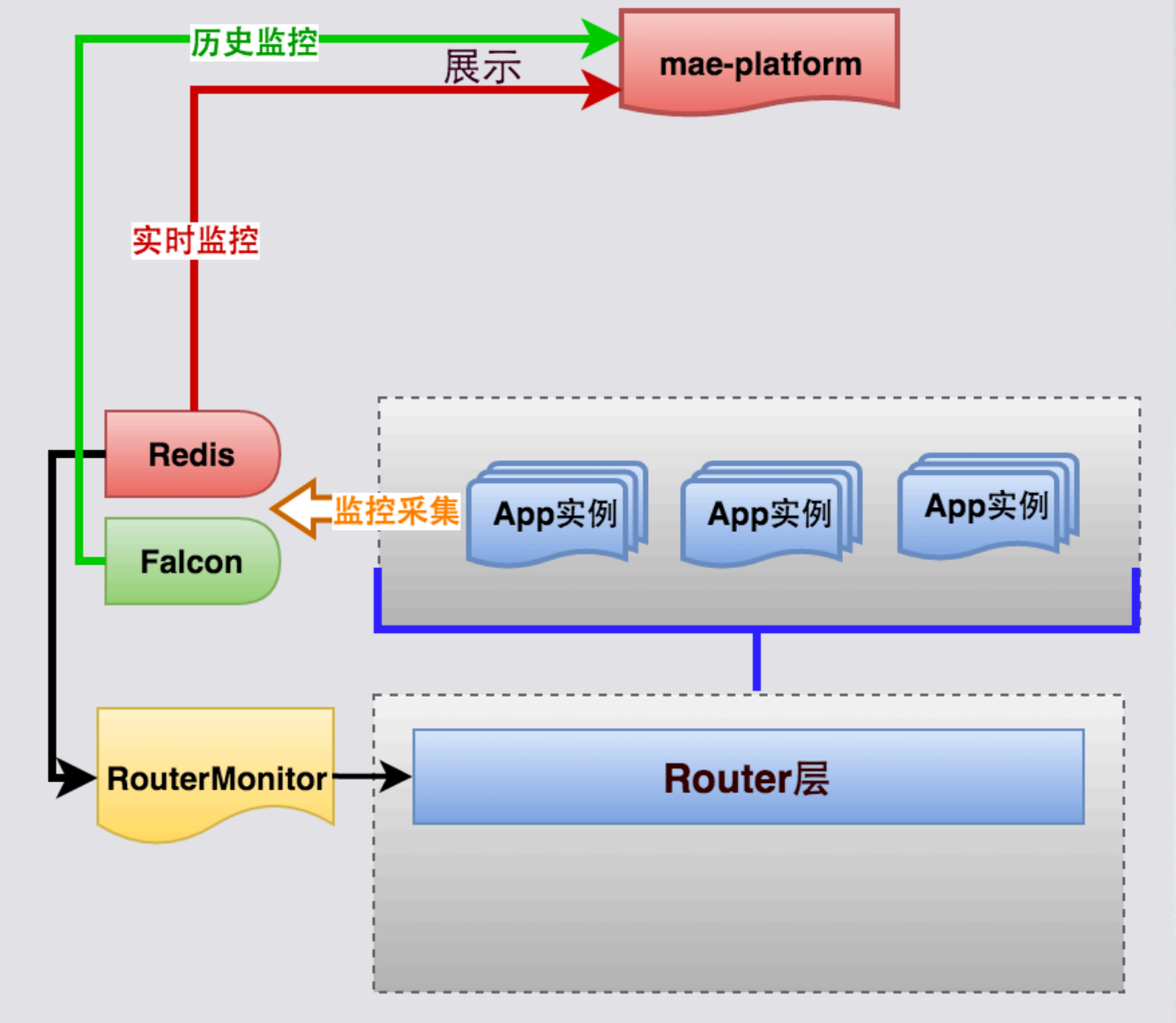
- 监控采集
  - ~~cAdvisor~~ -> metric包
  - 基础监控

    cpu、mem、网卡流量、socket、容器磁盘空间等等
  - 自定义监控

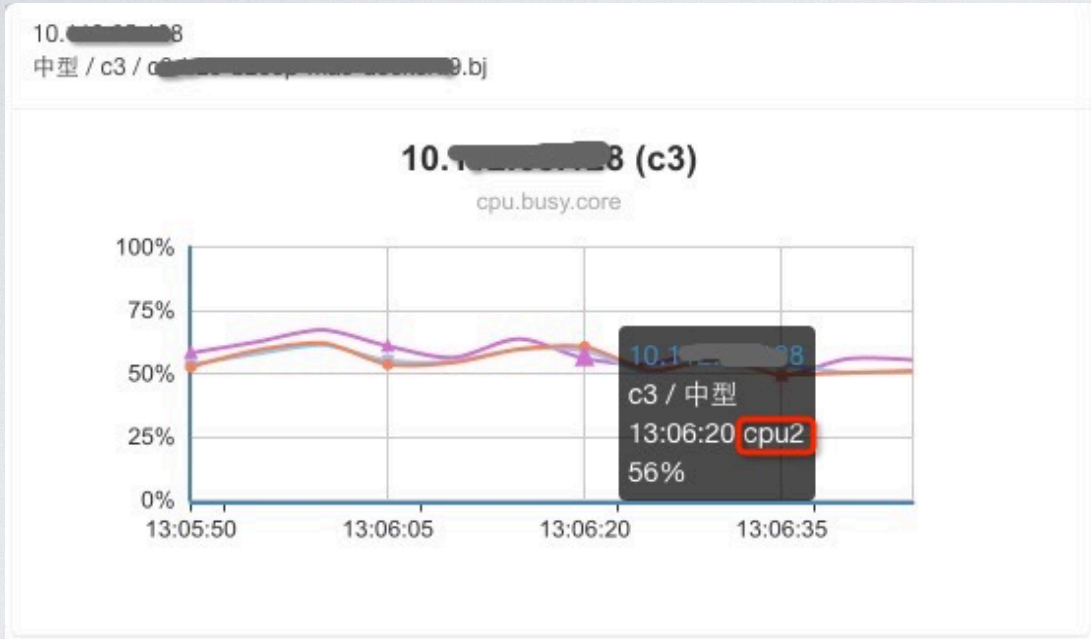
    日志监控、进程、端口、自定义脚本
- 监控展示
  - 实时监控

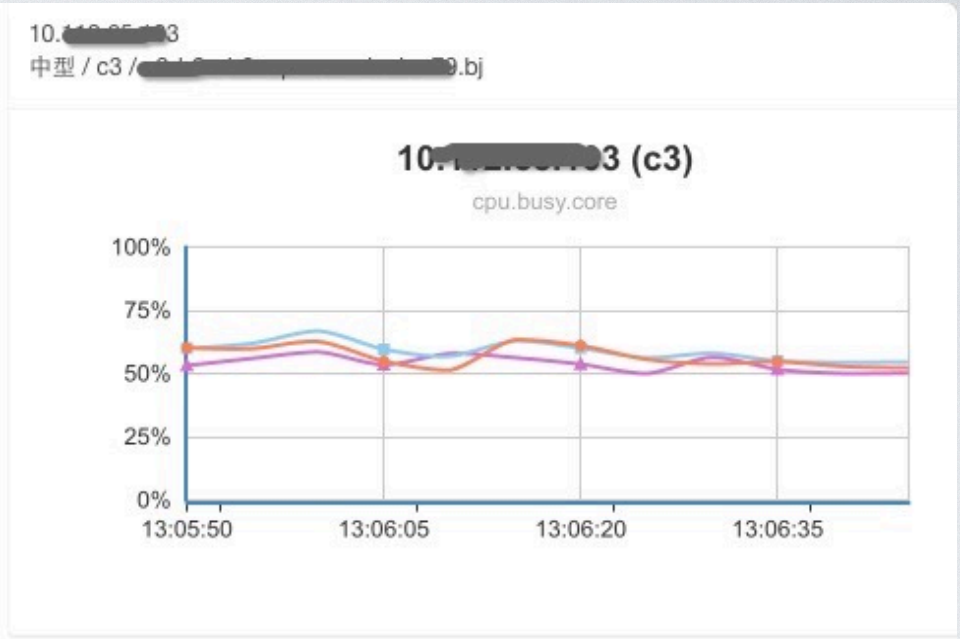
    时间维度、合并/拆分图表、Cpu核心Usage
  - falcon

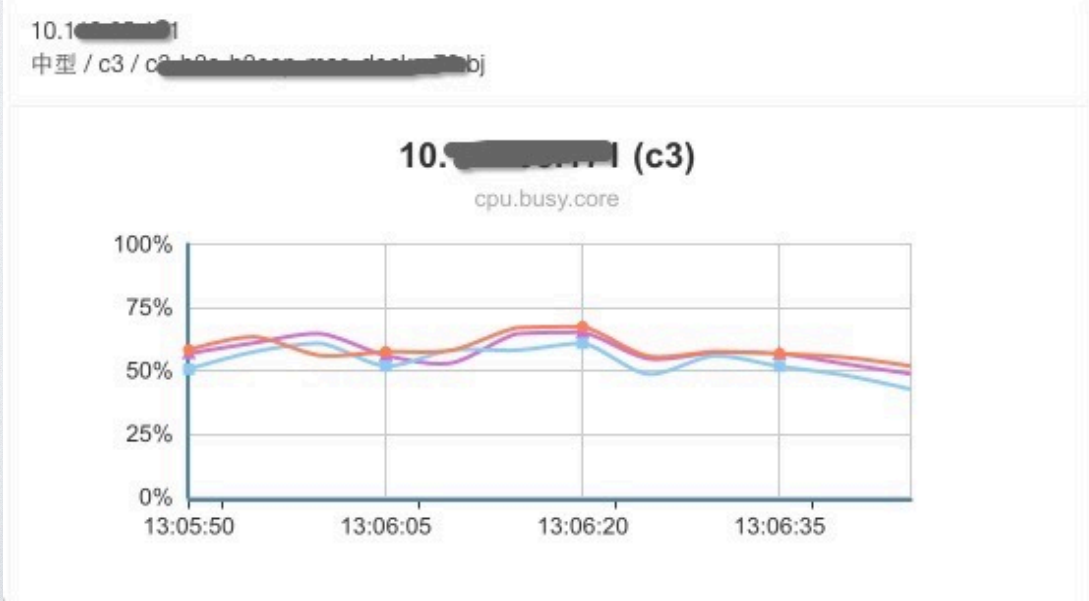
    历史监控、报警

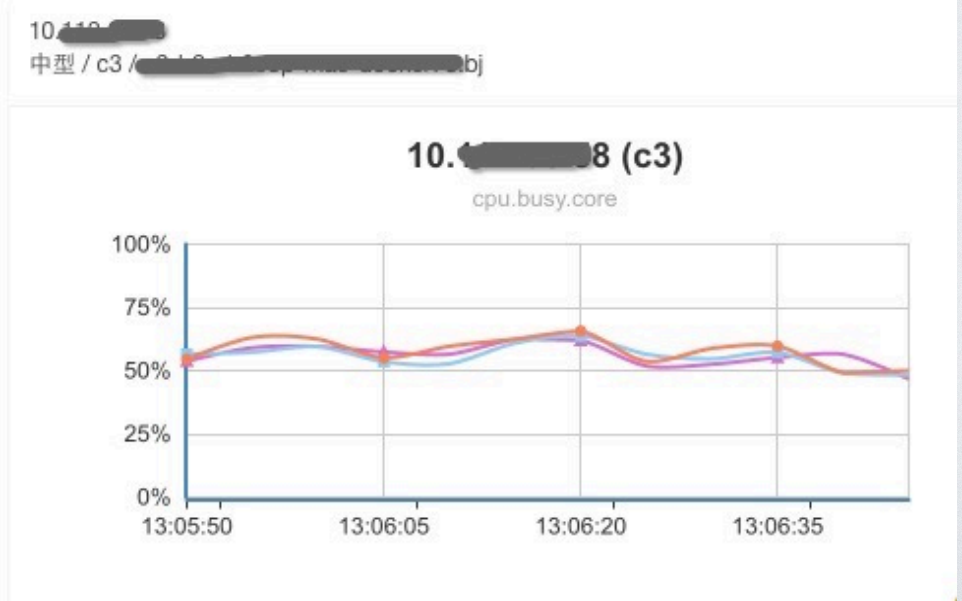


MAE – 监控展示


10.[redacted]8
中型 / c3 / [redacted].bj
10.[redacted]8 (c3)
cpu.busy.core
100%
75%
50%
25%
0%
13:05:50
13:06:05
13:06:20
13:06:35
10.1[redacted]8
c3 / 中型
13:06:20 cpu2
56%


10.[redacted]3
中型 / c3 / [redacted].bj
10.[redacted]3 (c3)
cpu.busy.core
100%
75%
50%
25%
0%
13:05:50
13:06:05
13:06:20
13:06:35


10.1[redacted]1
中型 / c3 / [redacted]bj
10.[redacted]1 (c3)
cpu.busy.core
100%
75%
50%
25%
0%
13:05:50
13:06:05
13:06:20
13:06:35


10.[redacted]
中型 / c3 / [redacted]bj
10.[redacted]8 (c3)
cpu.busy.core
100%
75%
50%
25%
0%
13:05:50
13:06:05
13:06:20
13:06:35

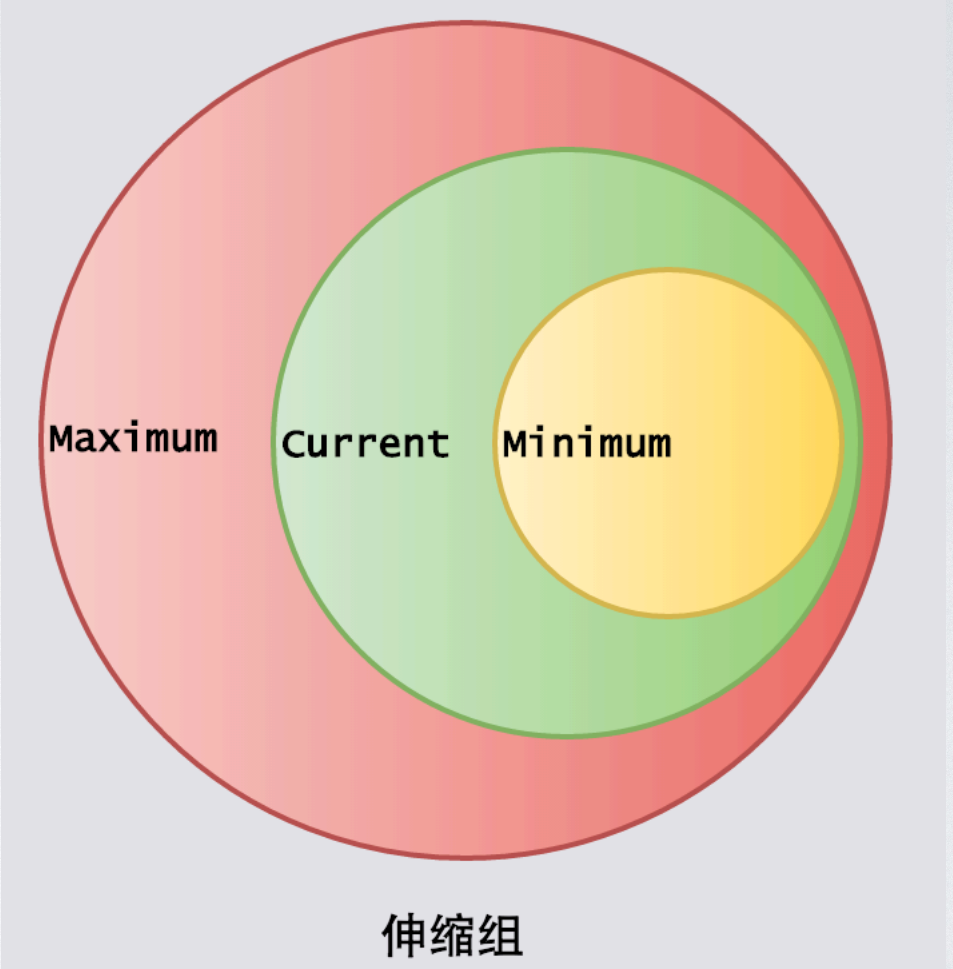
MAE – Autoscaling

- 自动伸缩
- 伸缩组
  - 触发规则

    持续负载（cpu、mem、net）

    负载类型 -> 资源模板
  - 扩容

    1 min



[redacted]

15,540,546

Count

[redacted]

2,514,...

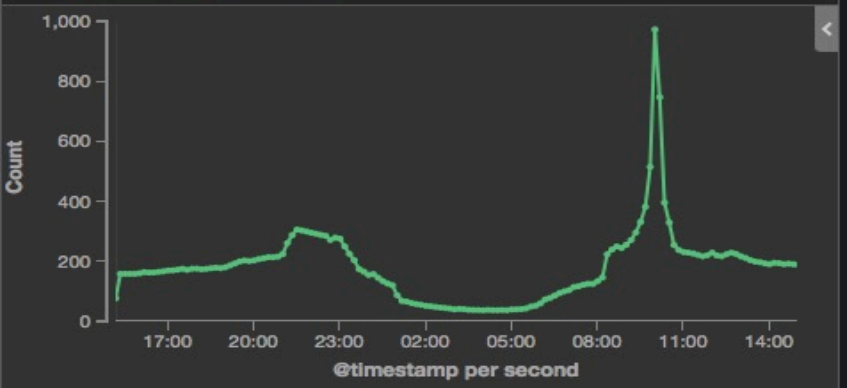
[redacted]

[redacted]

639,746

[redacted]

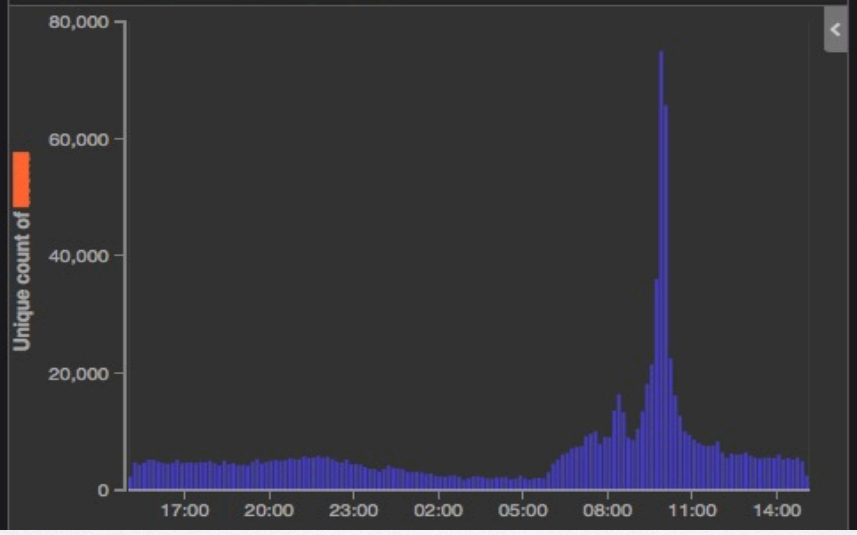
[redacted]



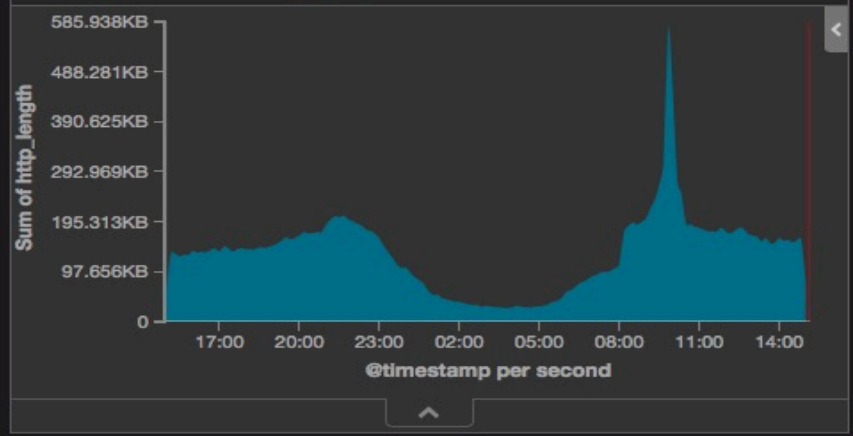
[redacted]

[redacted]分布区域

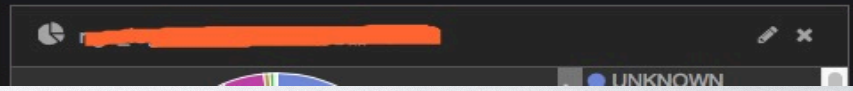
[redacted]



[redacted]



[redacted]



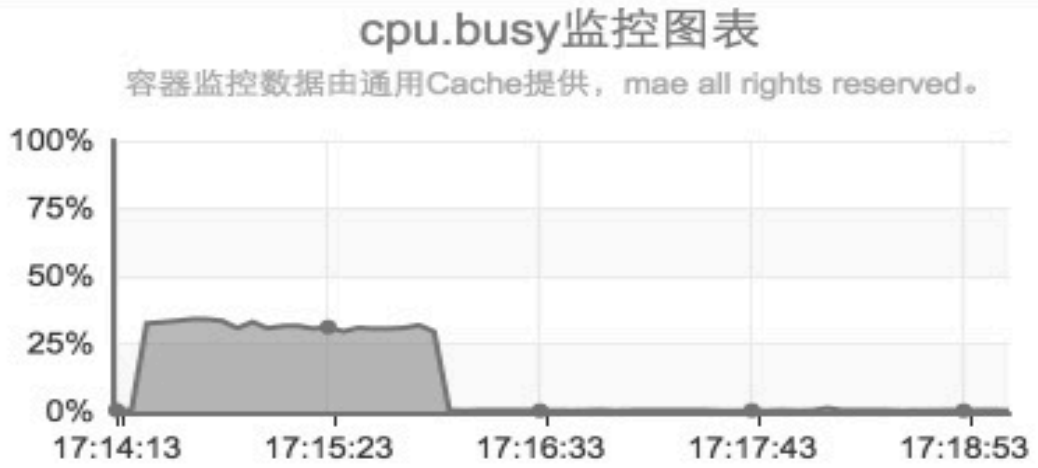
Router

Based on gorouter, but much different:

- **Register Driver: Etcd**
- **Load Balancing: LC/WRR**
- **XLocation**
- **High Proformace (fasthttp with proxy)**

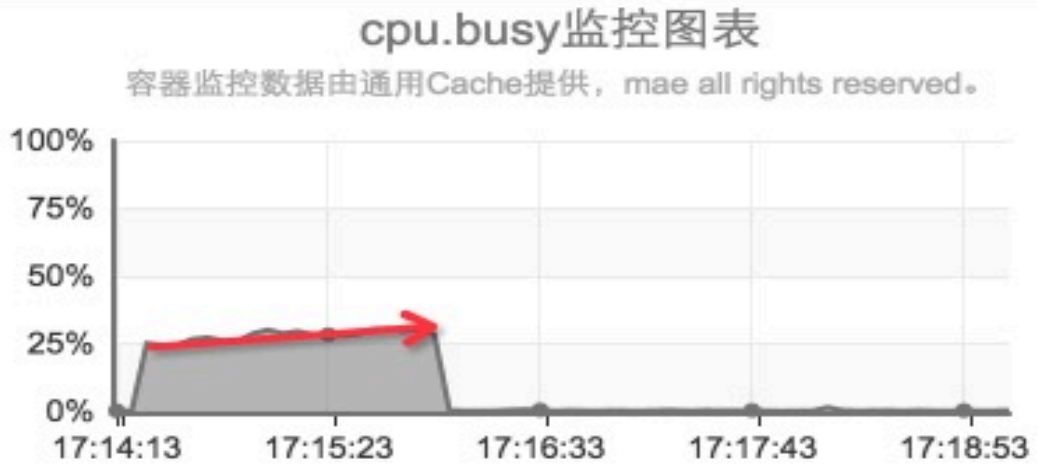
6bbad45f4dc3

小型 / C3机房 / [redacted]



99b3347c15e5

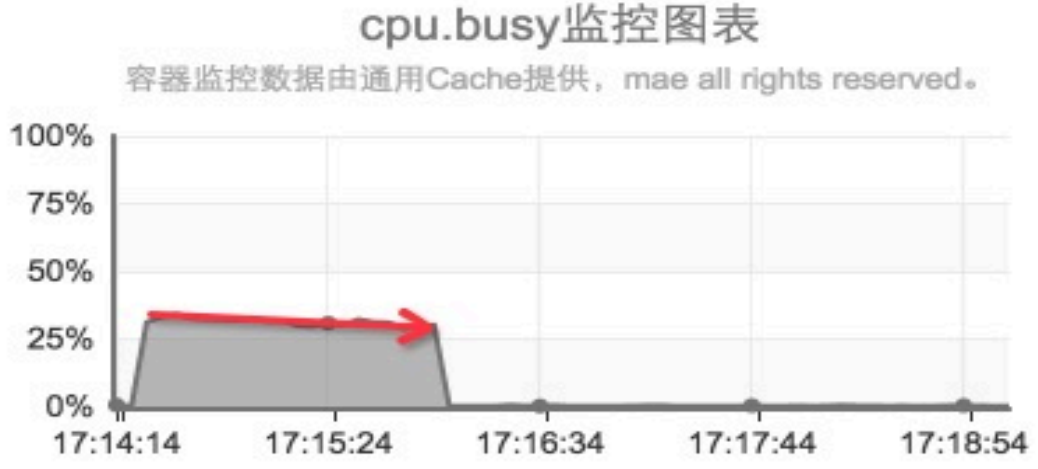
小型 / C3机房 / [redacted]



动态WRR

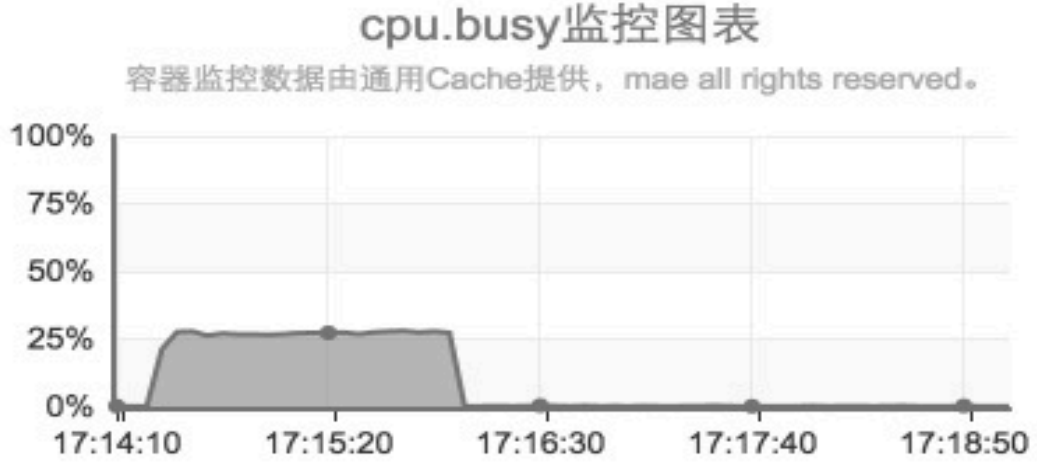
af7b9ba0198c

小型 / C3机房 / [redacted]



cb840f644100

小型 / C3机房 / [redacted]



Router优化

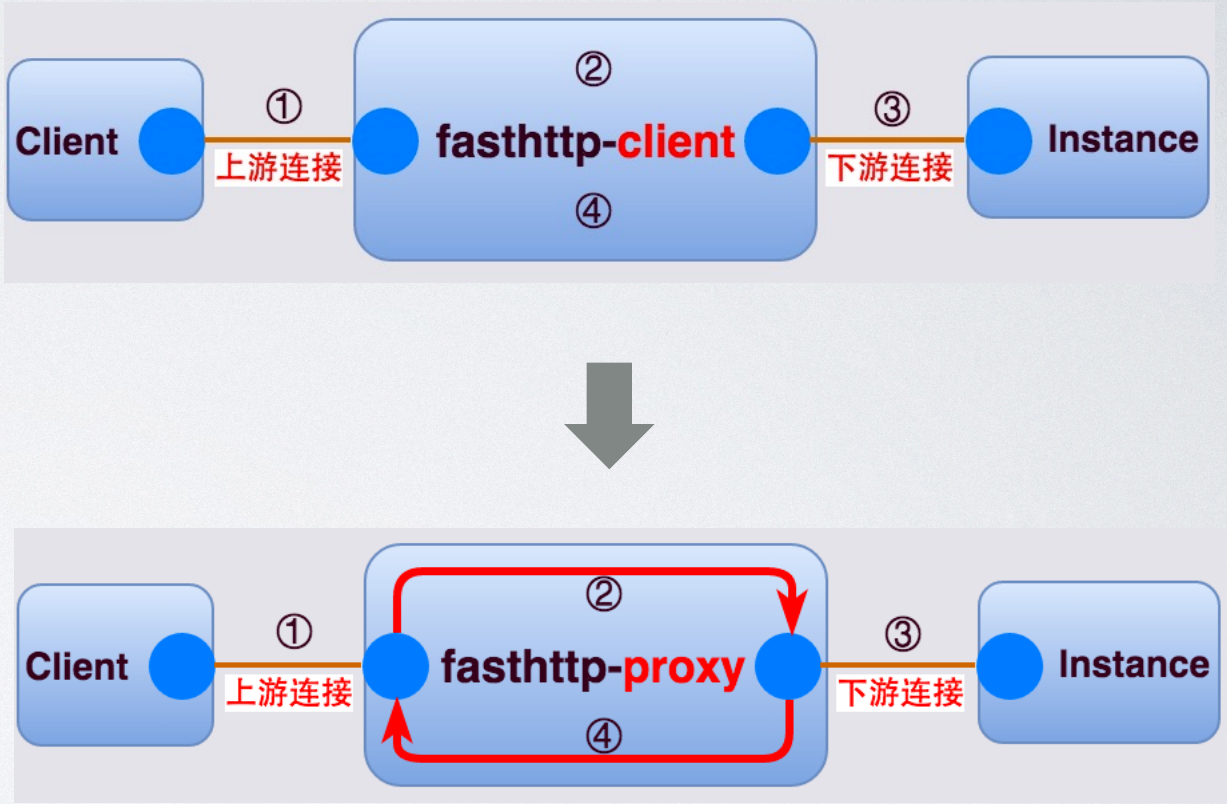
QPS： 6k -> ? -> ?

- 封装严重、sync.Pool(timer...)等

  QPS: 3w

- net/http -> fasthttp
  - Pool : Request/Response/Header
  - Pool : bufio Reader/Writer
  - Pool : conn、buffer
  - header struct
  - string -> []byte
- fasthttp-Proxy
- 日志优化

  缓冲延迟写
- **QPS: 10w**



MAE - Server模块

功能模块：

- ControllerServer

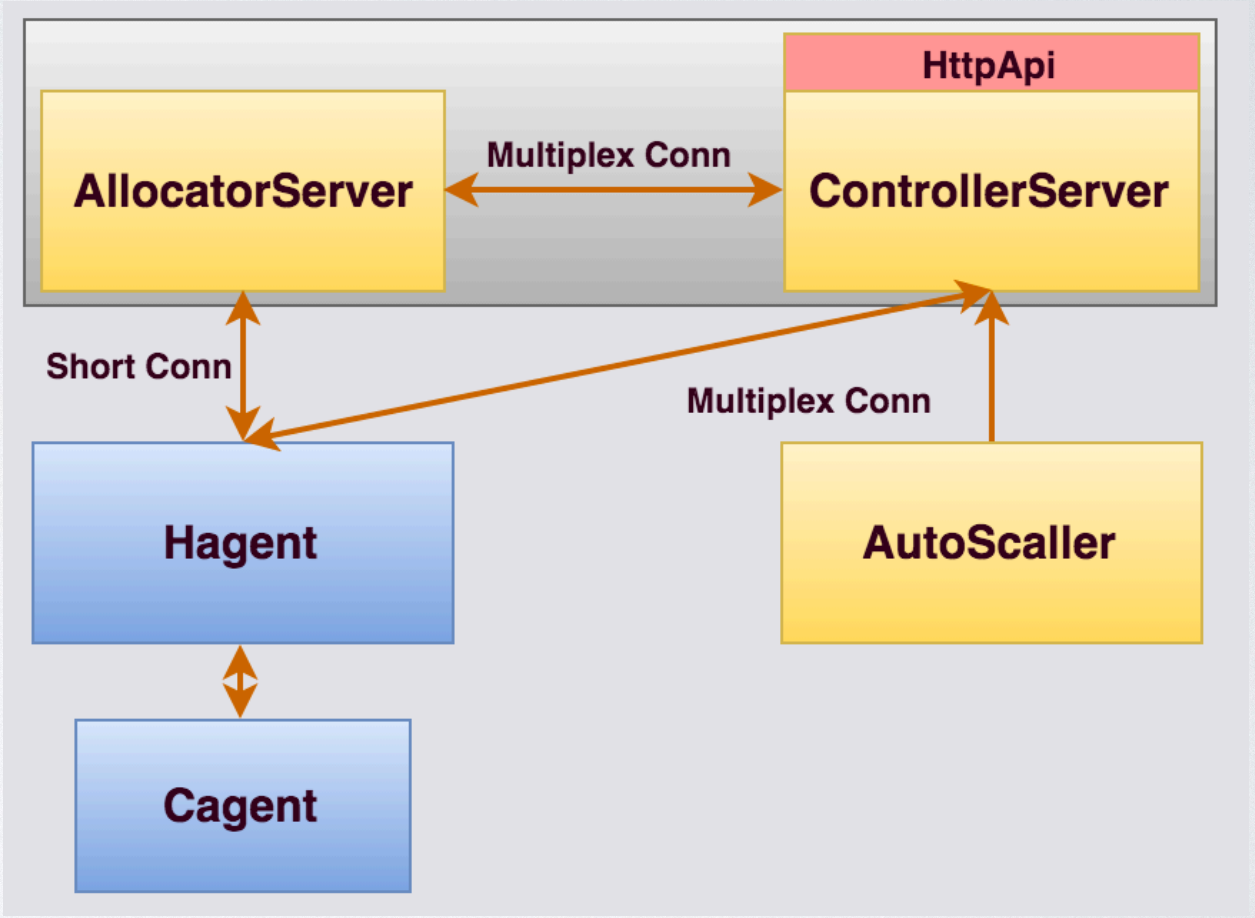
  总控模块，暴露API等。

- AllocatorServer

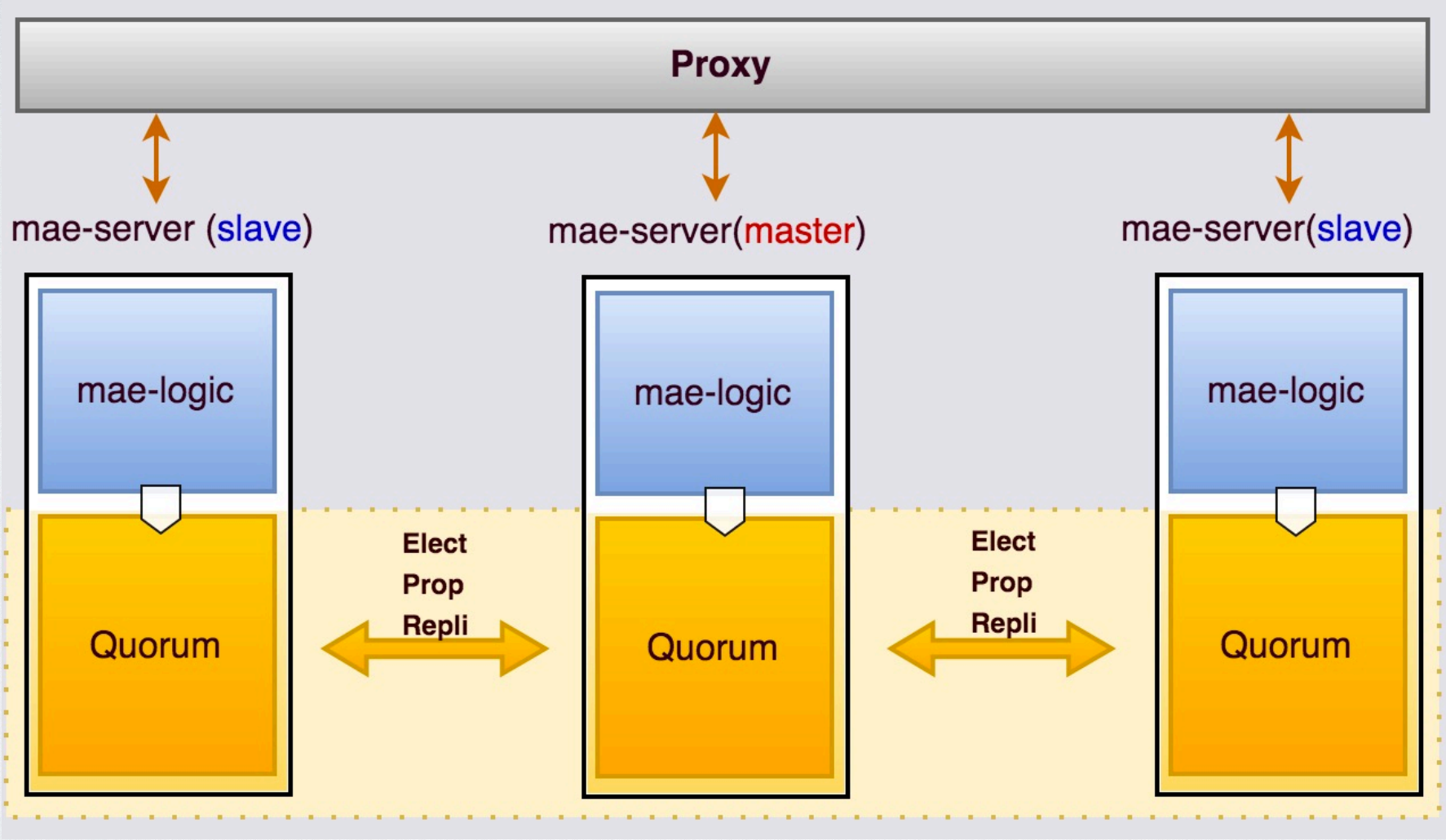
  资源管理/分配模块。

- AutoScaller

  自动伸缩模块。



MAE – Server高可用



Quorum

基于Raft协议的高可用框架。

Features:

- ✓ One-master & multi-slaves.
- ✓ Strong consistency： W + R > N (W == R == N/2 +1)
- ✓ Fault-tolerant.
  - Master election on faulture automatically .
  - Tolerate (N-1)/2 nodes to be crashed .
- ✓ R/W is transparent to the user.
- ✓ Key-value storage (memory only).
- ✓ Entire Log-Replication on node-startup.
- ✓ Read-Repair (repair old data on reading).

MAE – Quorum原理图

Layers：

- 应用逻辑层
- 接口层
- Quorum逻辑层

  election、hearbeat、proposal、rw、sync等
- 协议层

  msgpack、lz4
- Multiplex Connection
- KV存储层

```
type Quorumer interface {
    Start()
    Stop()

    IsLeader() bool
    HasLeader() bool
    Leader() string

    Keys(match string) []string
    DBSize() int

    Get(key string) (value string, err error)
    Set(key string, value string) error
    Delete(k string) error

    LocalGet(key string) (value string, err error)

    CompareAndSet(ekv storage.EKV) error
    CompareAndDelete(ekv storage.EKV) error

    MultiGet(keys []string) (kvs map[string]storage.GKV, err error)

    // Transactional sets, it'll rollback all if anyone fails.
    MultiSet(kvs []storage.KV) error
    // Transactional deletes, it'll rollback all if anyone fails.
    MultiDelete(kvs []storage.KV) error

    MultiCompareAndSet(ekvs []storage.EKV) error
    MultiCompareAndDelete(ekvs []storage.EKV) error
}
```

Multiplex Connection

Multiplex-Connection，是一种在单一TCP连接上实现并发请求的一种技术手段（框架）。 类似spdy功能。

特点：

- ✓ 单一长连接。
- ✓ 同时支持同步查询和异步查询。
- ✓ 同步查询请求超时设置。
- ✓ 连接双方，对等关系，均可以发送、接收、处理请求。
- ✓ 高性能

Multiplex Connection – 协议Packet

格式

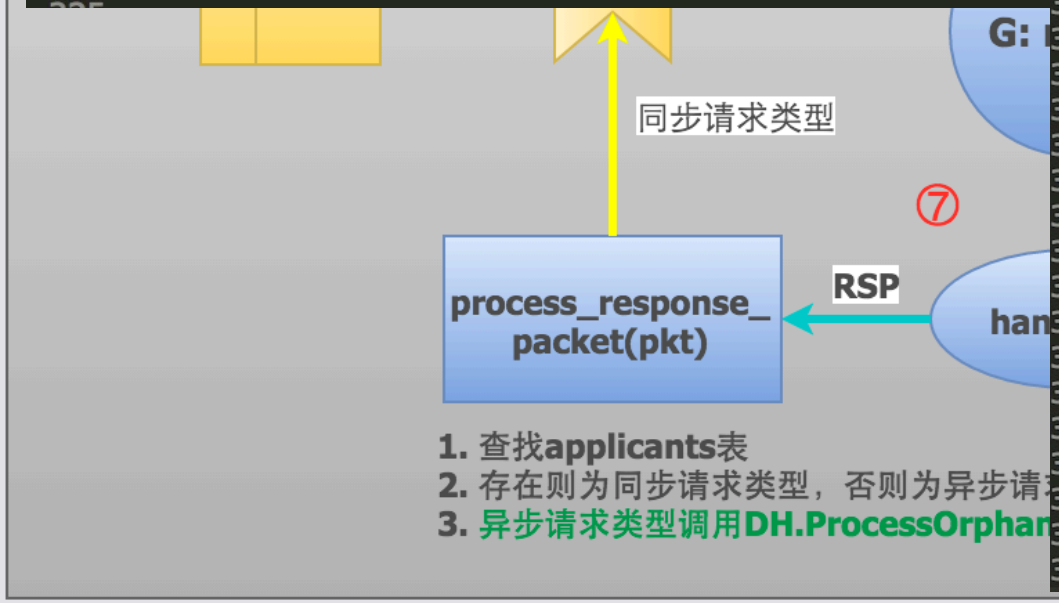
[协议头] [数据] \r\r\n

[3-bytes type] [4-bytes identity] [4-bytes body-size] [Y-bytes body] \r\r\n

协议头

- type: REQ或RSP。
- identity: 协议包序列号。
- body-size: 用户数据长度。

```go
func (c *connection) handle(data []byte) (err error) {
    var pkt *Packet
    pkt, err = decode_packet(data)
    if err != nil {
        log.Println("decode_packet error:", err)
        return
    }

    switch pkt.Type {
    case "REQ":
        return c.process_request_packet(pkt)
    case "RSP":
        return c.process_response_packet(pkt)
    default:
        err = ErrProtoUnknownType
    }

    return
}
```




```go
// 处理 对方的响应
// 查找recv_chan
func (c *connection) process_response_packet(p *Packet) (err error) {
    if p == nil {
        return errors.New("empty packet")
    }

    recv, ok := c.popApplicant(p.Identity)
    if !ok {
        return c.dh.ProcessOrphanResponse(p.Body)  //处理：异步查询类型
    } else if recv == nil {
        return ErrAppNotFound
    }

    select {
    case <-c.chexit:
        return ErrExited
    case <-recv.timer.C:
        return ErrTimeout
    case recv.ch <- p.Body:
        break
    }

    return nil
}
```

查找applicants表

处理：异步查询请求的响应

处理：同步查询请求的响应

踩过的坑

docker daemon进程内存泄露（pkg/ioutils/readers.go）：

```
    // The buffered data is moved to a fresh buffer,
    // swap the old buffer with the new one and
    // reset all counters.
    if reset {
        newbuf := &bytes.Buffer{}
        newbuf.ReadFrom(r.buf)
        r.buf = newbuf
        lastReset = now
        reset = false
        dataSinceReset = 0
        maxBufLen = 0
        reuseCount = 0
    }
}
if err != nil {
    r.err = err
} else {
    r.buf.Write(r.drainBuf[0:n])
}
reuseCount++
r.wait.Signal()
```

```
169             // The buffered data is moved to a fresh
170             // swap the old buffer with the new one
171             // reset all counters.
172             if reset {
173                 oldbuf := r.buf
174                 newbuf := &bytes.Buffer{}
175                 //newbuf.ReadFrom(r.buf)
176                 r.buf = newbuf
177                 if oldbuf != nil {
178                     oldbuf.Reset()
179                     oldbuf = nil
180                 }
181                 r.buf.Write([]byte("\n[Docker] Warni
    \n"))
182                 lastReset = now
183                 reset = false
184                 dataSinceReset = 0
185                 maxBufLen = 0
186                 reuseCount = 0
187             }
188         }
```

运维标准：应用程序日志

踩过的坑

docker stop 命令夯死问题（daemon/container.go ）：

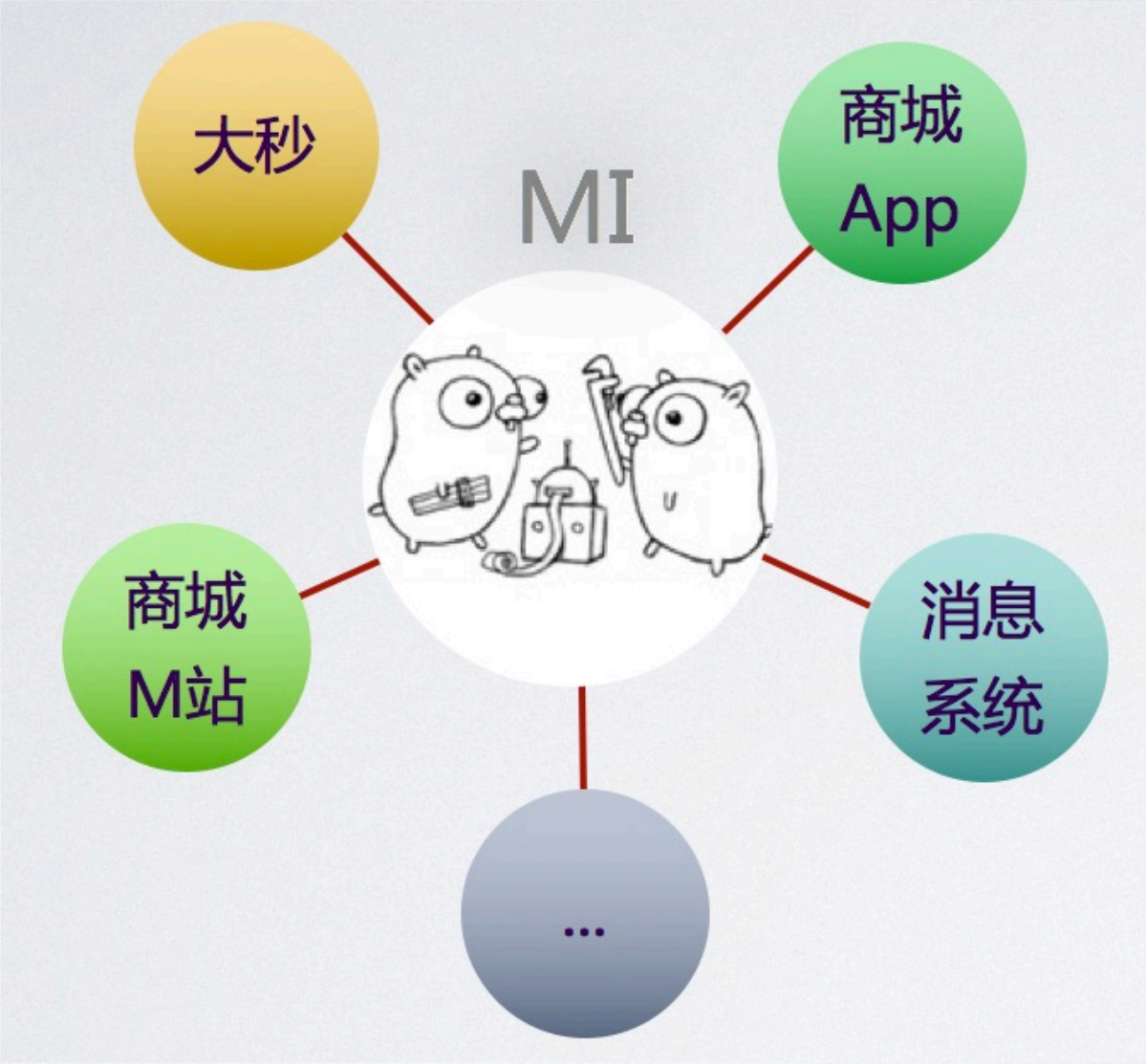
```
    // 1. Send a SIGTERM
    if err := container.killPossiblyDeadProcess(15); err != nil {
        log.Infof("Failed to send SIGTERM to the process, force killing")
        if err := container.killPossiblyDeadProcess(9); err != nil {
            return err
        }
    }

    // 2. Wait for the process to exit on its own
    if _, err := container.WaitStop(time.Duration(seconds) * time.Second); err != nil {
        log.Infof("Container %v failed to exit within %d seconds of SIGTERM - using the force", container.ID, seconds)
        // 3. If it doesn't, then send SIGKILL
        if err := container.Kill(); err != nil {
            if err == execdriver.ErrNotRunning {
                container.Reset()
                return nil
            }
            log.Infof("Container %v Kill error: %v", container.ID, err)
            container.WaitStop(-1 * time.Second)
            return err
        }
    }
    return nil
}
```

超时

新增：重置容器状态，强制退出。

消息chan未正常close，无限等待。

GO在小米商城研发中的应用



MAE – 2016规划

- 混合云
- 资源精细化运维

stormgbs

Thanks !