Open Talk

OpenResty x Open Talk
WUHAN 2019

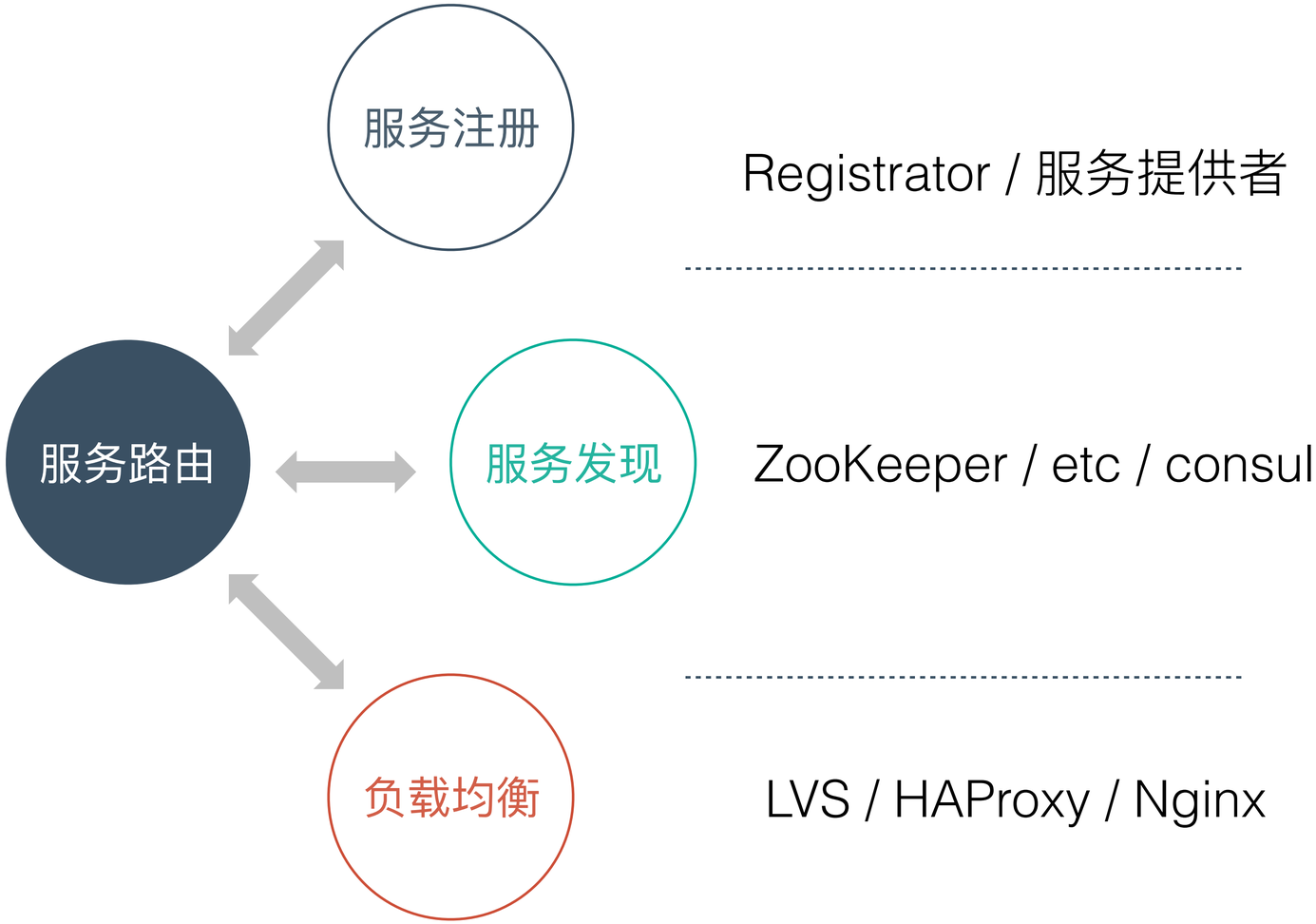
基于 OpenResty 的动态服务路由方案

邵海杨 @ 又拍云

主办方：OpenResty 又拍云

如何做到服务的 zero down-time 更新?

服务路由


服务注册
服务路由
服务发现
负载均衡
Registrator / 服务提供者
ZooKeeper / etc / consul
LVS / HAProxy / Nginx

服务发现

服务发现、健康检查、
DNS、监控、多数据中心
等完整的解决方案

轻量、需要第三方工具配合

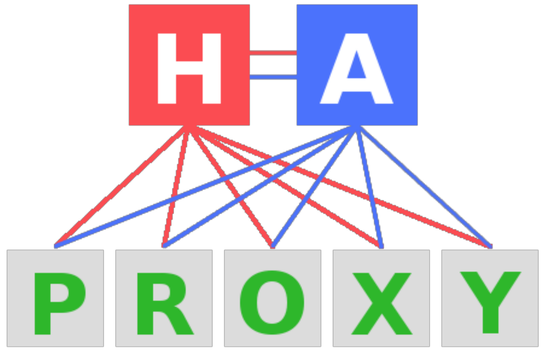
成熟、强大、复杂、资源要求高

Apache ZooKeeper™

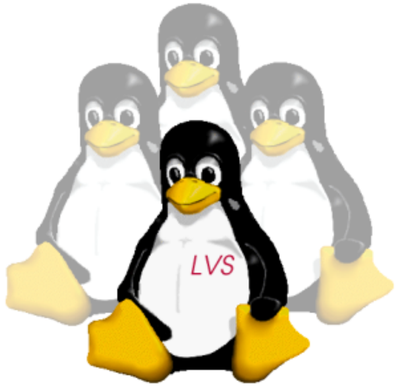
负载均衡

专注于 HTTP，功能强大

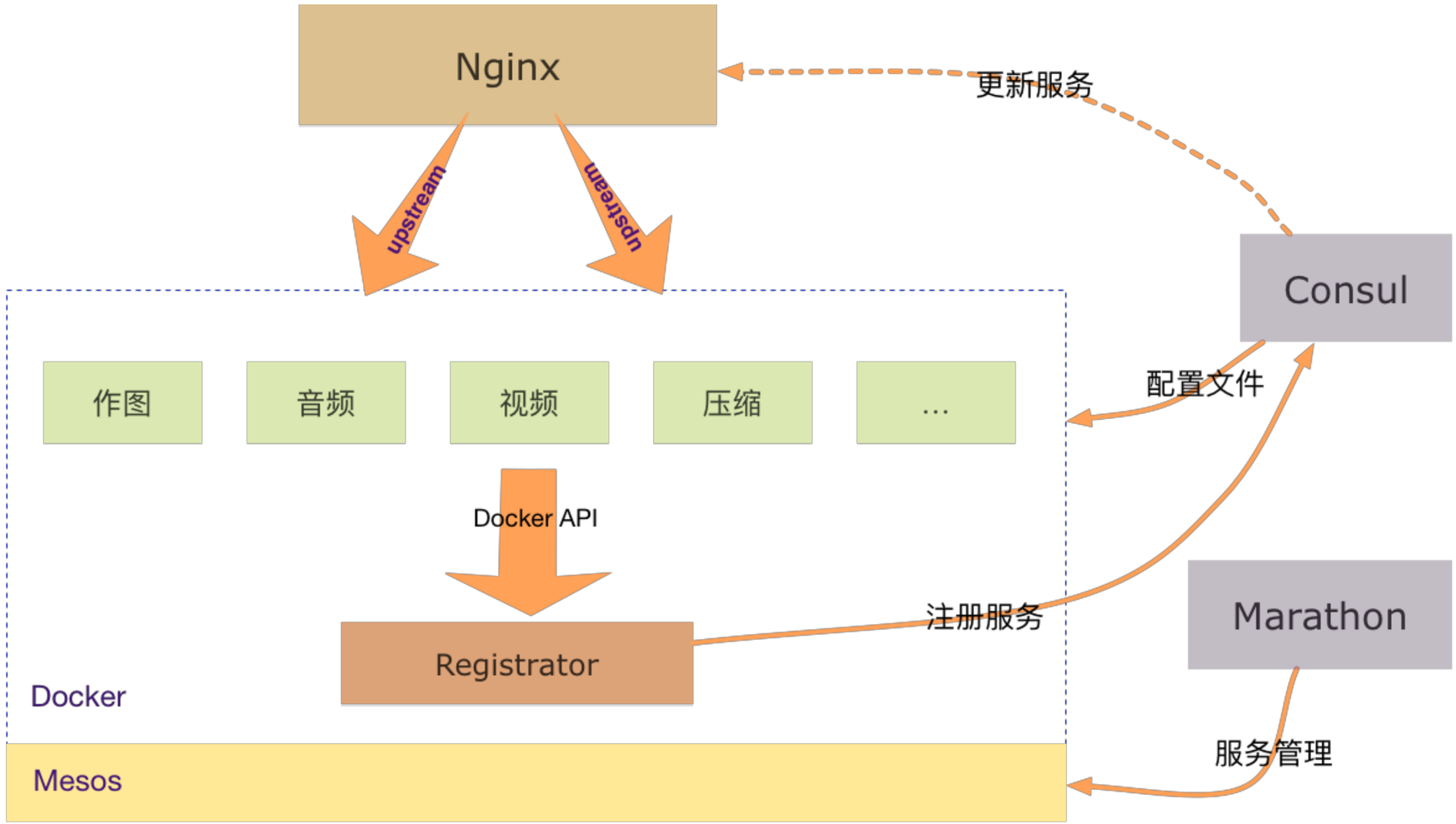
TCP、HTTP，使用方便



网络层高可用、功能少



Nginx + Consul + Registrator



面临的问题

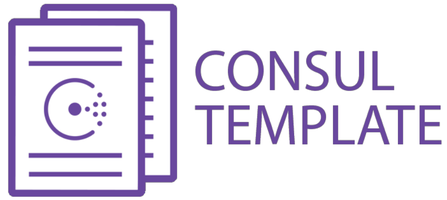
Consul 里的服务如何更新到 Nginx?

方案一 - consul-template

监听consul 中服务变动

由模板重新生成配置

执行 nginx reload 操作



方案一 - Nginx reload 过程

1. 由模板生成 upstream.conf

```
upstream imgprocess {
{{range service "app.imgprocess"}}
    server {{.address}} max_fails={{.max_fails}} fail_timeout={{.fail_timeout}};{{end}}
}
```

```
upstream imgprocess {
    server 192.168.1.1:8080 max_fails=3 fail_timeout=30s;
    server 192.168.1.2:8080 max_fails=3 fail_timeout=30s;
    server 192.168.1.3:8080 max_fails=3 fail_timeout=30s;
}
```

2. kill -s HUP nginx.pid 或者 nginx -s reload

方案一 - Reload 的缺点

频繁操作会有性能损耗

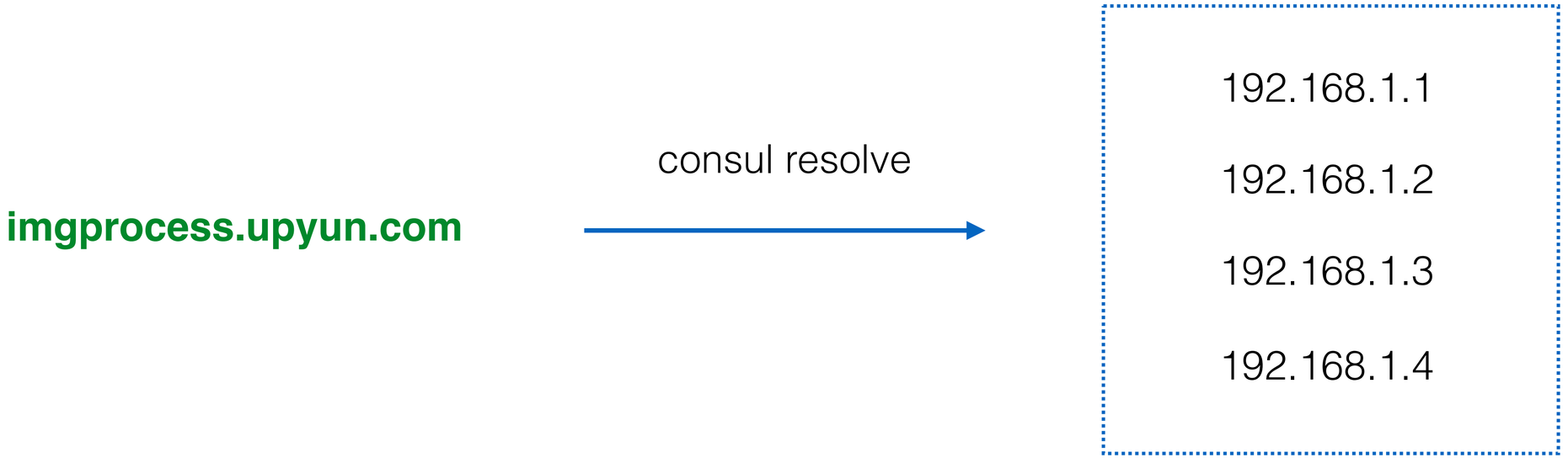
旧进程长时间处于 shutting down 状态

进程内缓存失效

与设计初衷不符！

方案二 - 内部 DNS 方案

```
upstream imgprocess {
    server imgprocess.upyun.com:8080 max_fails=3 fail_timeout=10s;
}
```



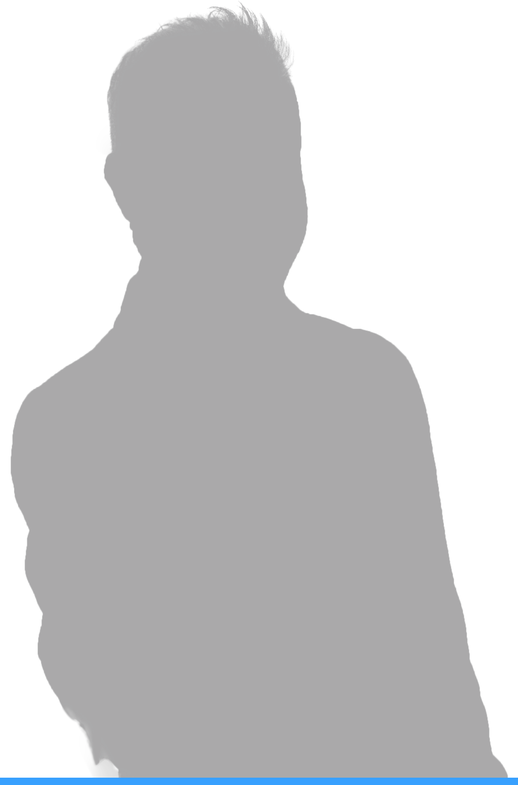
方案二 - DNS 的缺点

多一层 DNS 解析时间 ..... 增加了额外的处理时间

DNS 缓存 ..... 即使异常服务被摘掉，
仍会持续出错直到缓存失效

端口号不能改变 ..... Docker 容器在更新和迁移过程中
IP 和端口号都有可能改变

我们想要的是



通过**HTTP接口**动态修改Nginx上游服务列表

方案三 - ngx_http_dyups_module

GET /detail 获取所有upstream列表和servers
GET /list 获取upstream列表
GET /upstream/name 根据name获取upstream列表

POST /upstream/name 更新upstream
body里是新的servers列表

DELETE /upstream/name 删除upstream

方案三 - ngx_http_dyups_module

```
$ curl -H "host: dyhost" 127.0.0.1:8080
<html>
<head><title>502 Bad Gateway</title></head>
<body bgcolor="white">
<center><h1>502 Bad Gateway</h1></center>
<hr><center>nginx/1.3.13</center>
</body>
</html>

$ curl -d "server 127.0.0.1:8089;server 127.0.0.1:8088;" 127.0.0.1:8081/upstream/dyhost
success

$ curl -H "host: dyhost" 127.0.0.1:8080
8089
```

方案三 - 为什么不用 C 方案

依赖Nginx本身的负载均衡算法

只能用nginx自身的负载均衡算法
加新的算法也需要C实现

二次开发效率低

C的开发效率远不及lua

纯lua的方案无法使用

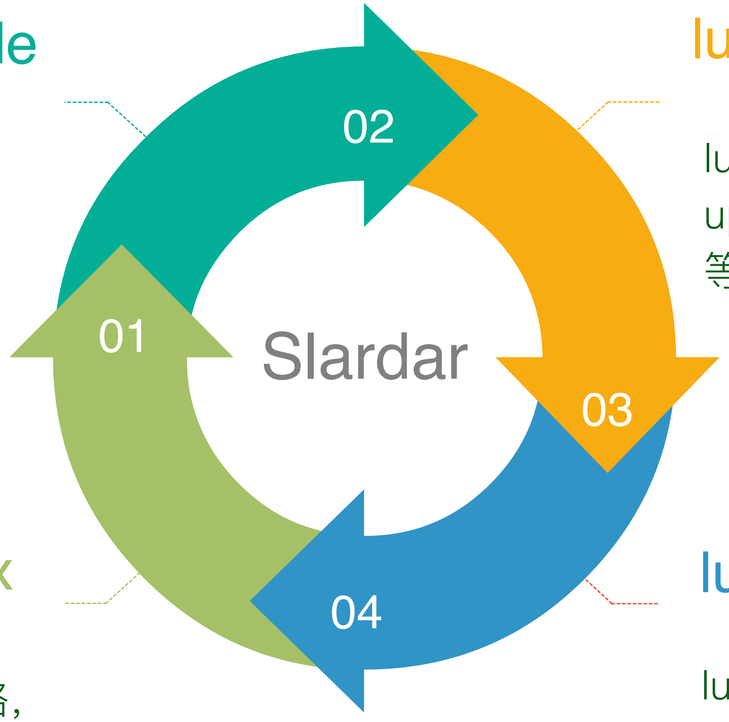
只影响C的proxy模块，不能在
lua代码里拿到peer

开始造我们的轮子

动态负载均衡 - Slardar

lua-nginx-module

在Nginx 里用lua写逻辑，
balance_by_lua_*对
upstream选择进行hook



lua-resty-checkups

lua版的upstream，实现了动态
upstream管理、主被动健康检查
等功能

Nginx

proxy_* 指令和upstream重试策略，
如proxy_next_upstream_retry
和proxy_next_upstream_timeout

luasocket

lua的阻塞网络库
用于在启动时从consul
拉upstream列表

动态负载均衡 - lua-resty-checkups

动态upstream管理	update_upstream / delete_upstream 基于共享内存实现worker间同步
被动健康检查	max_fails / fail_timeout fail_timeout秒内失败max_fails 次则把该上游标记为fail_timeout 秒内不可用
主动健康检查	heartbeat 定时给上游发送心跳包检测服务是否存活
负载均衡算法	wrr / 一致性哈希 / 速度优先 / 本地优先 本地优先可节约内网流量

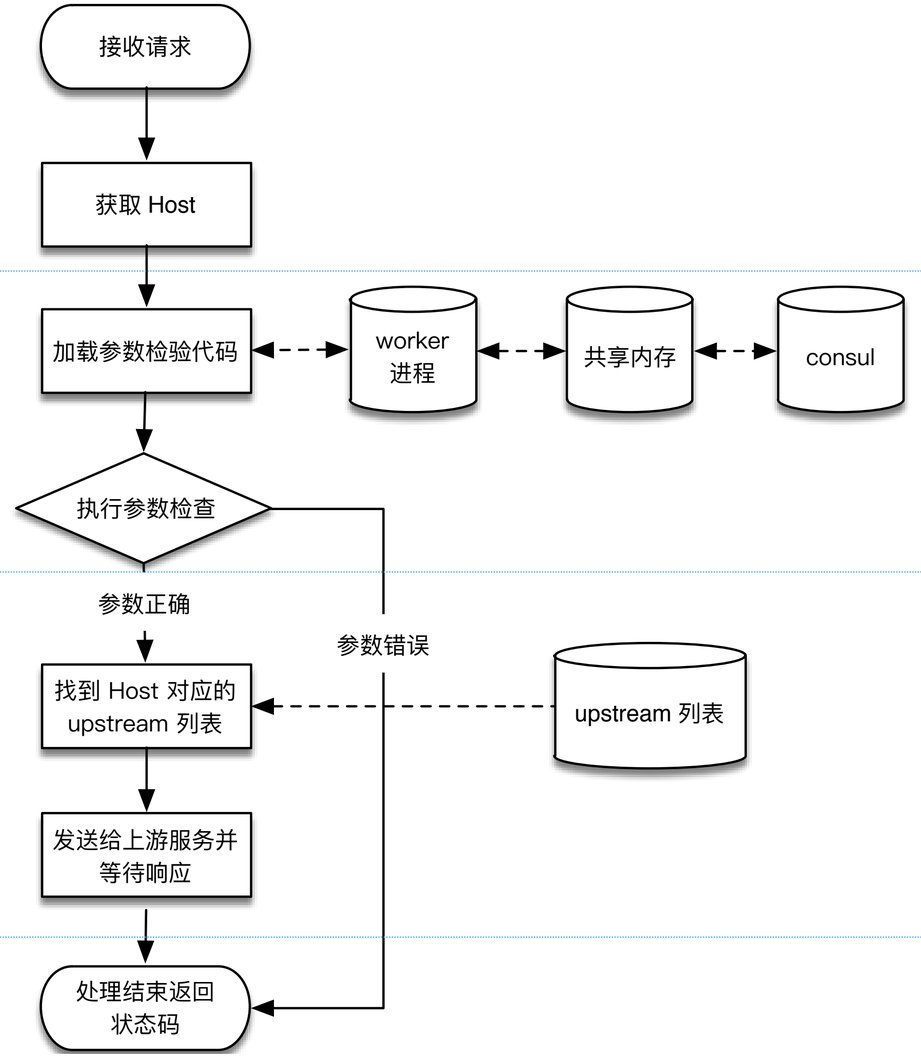
动态负载均衡 - 服务区分

以 Host 区分服务:

```
$ curl -T cat.jpg 192.168.1.155:3130 -H "Host: imageinfo”
$ curl -T cat.jpg 192.168.1.155:3130 -H "Host: imgprocess”
```

动态负载均衡 - 请求流程

请求处理流程图



动态lua代码加载

动态负载均衡

动态负载均衡 - 动态 upstream 更新

通过 HTTP 接口动态更新 upstream 列表:

```
curl -d \
  '{"servers":[ \
   {"host":"10.0.5.108", "port": 4001}, \       --> node docker 1
   {"host":"10.0.5.109", "port": 4001}], \      --> node docker 2
   "keepalive": 20}'\
  127.0.0.1:1995/upstream/node-dev.upyun.com
```

动态负载均衡 - 动态 upstream 更新

http://127.0.0.1:1995/status


```
"cls:node-dev.upyun.com": [
    [
        {
            "server": "node-dev.upyun.com:10.0.5.108:4001",
            "msg": null,
            "status": "ok",
            "lastmodified": "2016-07-05 16:23:48",
            "fail_num": 0
        },
        {
            "server": "node-dev.upyun.com:10.0.5.109:4001",
            "msg": "connection refused",
            "status": "err",
            "lastmodified": "2016-07-06 14:50:22",
            "fail_num": 1
        }
    ]
],
```


主动健康检查

动态负载均衡 - 动态 lua 加载

对请求做改写

执行简单的参数检查，节省带宽

动态负载均衡 - 动态 lua 加载

E.X. 禁止删除操作

```
if ngx.get_method() == “DELETE” and ngx.var.host == “admin.upyun.com” then
    return ngx.exit(403)
end
```

动态负载均衡 - 动态 lua 加载

http://127.0.0.1:1995/lua


```
{
    "version": 0,
    "modules": [
        {
            "time": "2016-08-04 13:51:56",
            "version": "a5b5c2c708781801917c0197369ed129",
            "name": "script.brook"
        },
        {
            "time": "2016-08-04 13:51:56",
            "version": "861da3db0627c4cc00a5d6637b175540",
            "name": "script.image"
        },
        {
            "time": "2016-08-04 13:51:56",
            "version": "5ad13455d5b449489c8073ac693f0c77",
            "name": "script.imageinfo"
        },
        {
            "time": "2016-08-04 13:51:56",
            "version": "8fb64fe3debfb525242b1f19e523100e",
            "name": "script.imgprocess"
        }
    ]
}
```

动态负载均衡 - 实现

1. 动态 upstream 管理
2. 负载均衡
3. 动态 lua 代码加载

动态负载均衡 - 动态 upstream 管理

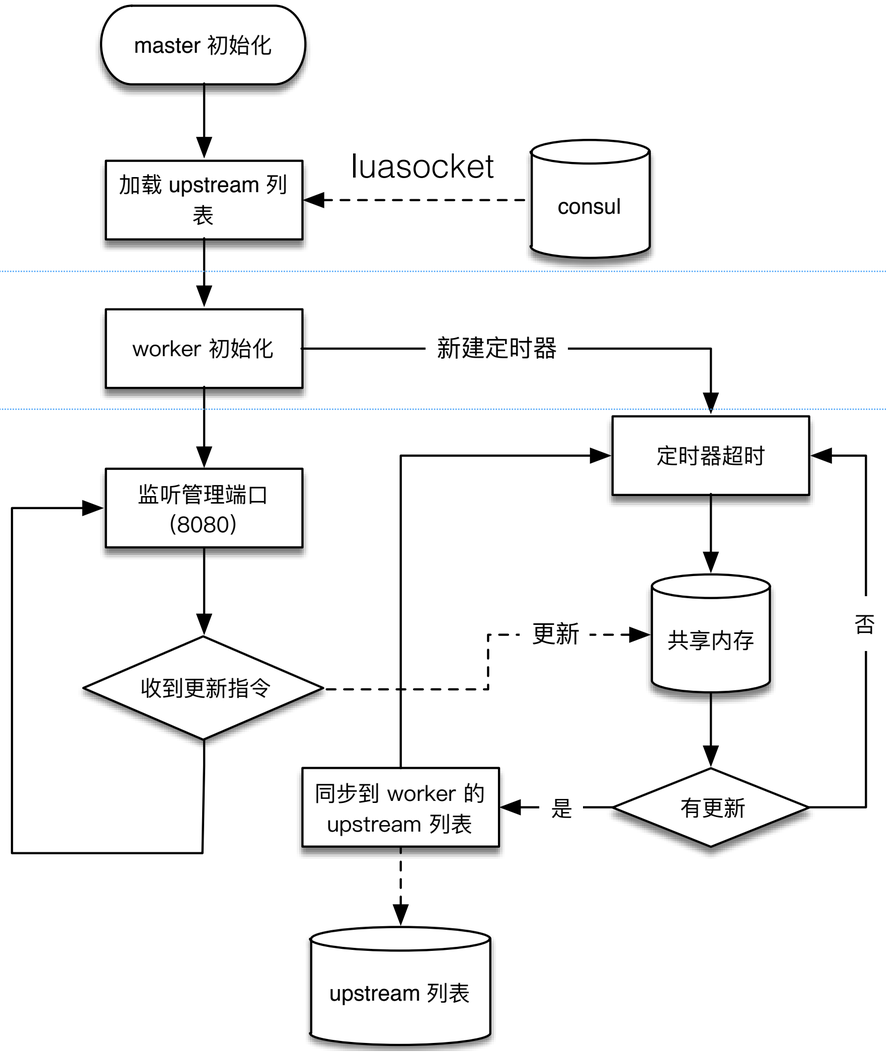
启动时通过 luasocket 从 consul 加载配置文件

监听管理端口，接收 upstream 更新指令

利用共享内存和定时器进行 worker 间同步

动态负载均衡 - 动态 upstream 管理

动态upstream流程图



动态负载均衡 - 负载均衡

balance_by_lua_*

upstream.conf:

```
upstream common {
    server 0.0.0.1;
    balancer_by_lua_file app/src/slardar_balance.lua;
}
```

动态负载均衡 - 负载均衡

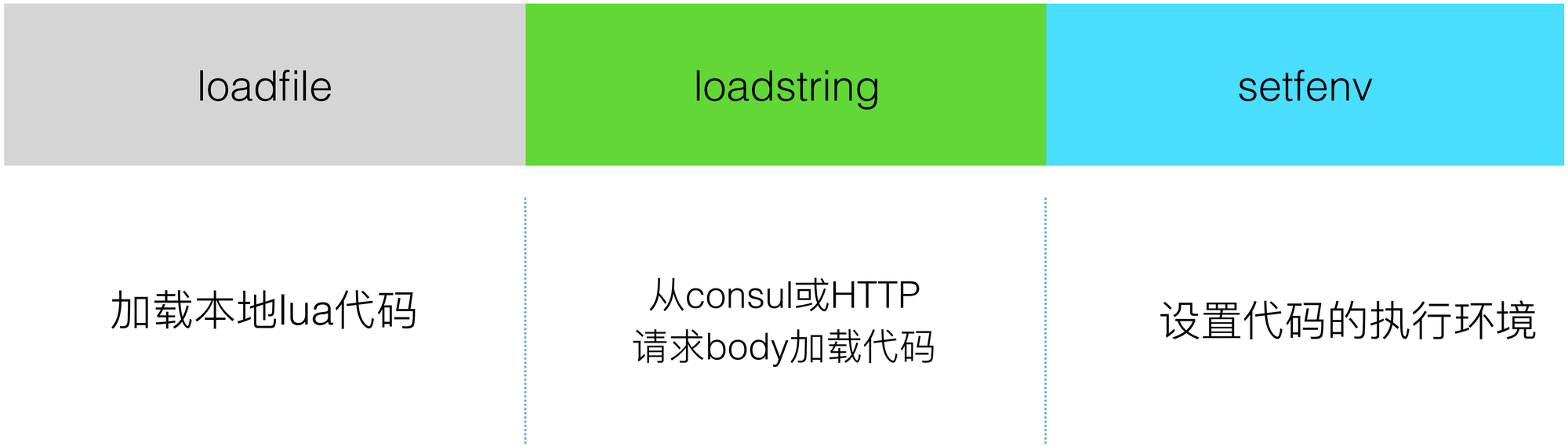
app/src/slardar_balance.lua:

```
local status, code = balancer.get_last_failure()
if status == "failed" then
    local last_peer = ngx.ctx.last_peer
    -- mark last_peer failed
    checkups.feedback_status(skey, last_peer.host, last_peer.port, true)
end

local peer = checkups.select_peer(ngx.var.host)
ngx.ctx.last_peer = peer

balancer.set_current_peer(peer.host, peer.port)
balancer.set_more_tries(1)
```

动态负载均衡 - 动态 lua 加载



动态负载均衡 - 优势

lua-resty-checkups ＋ balance_by_lua_*

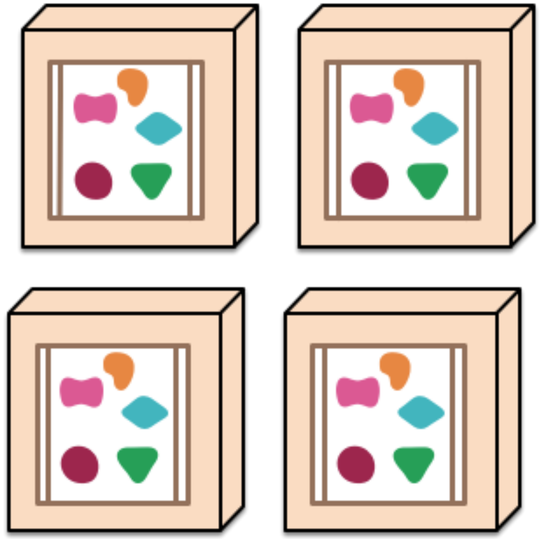
- **纯 lua 实现，不依赖第三方 C 模块**
  - 二次开发非常高效，减少维护负担
- **可以用 Nginx 原生的 proxy_***
  - proxy_next_upstream_tries / proxy_next_upstream_timeout
  - proxy_xxx
- **适用于几乎任何 ngx_lua 项目**
  - 可同时满足纯lua方案与c方案

在微服务架构里，Slardar 能做什么?

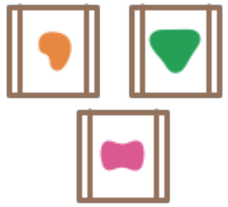
微服务

A monolithic application puts all its functionality into a single process...

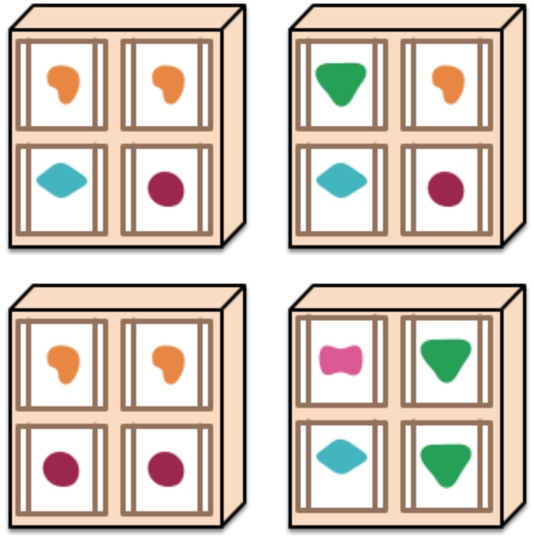
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...

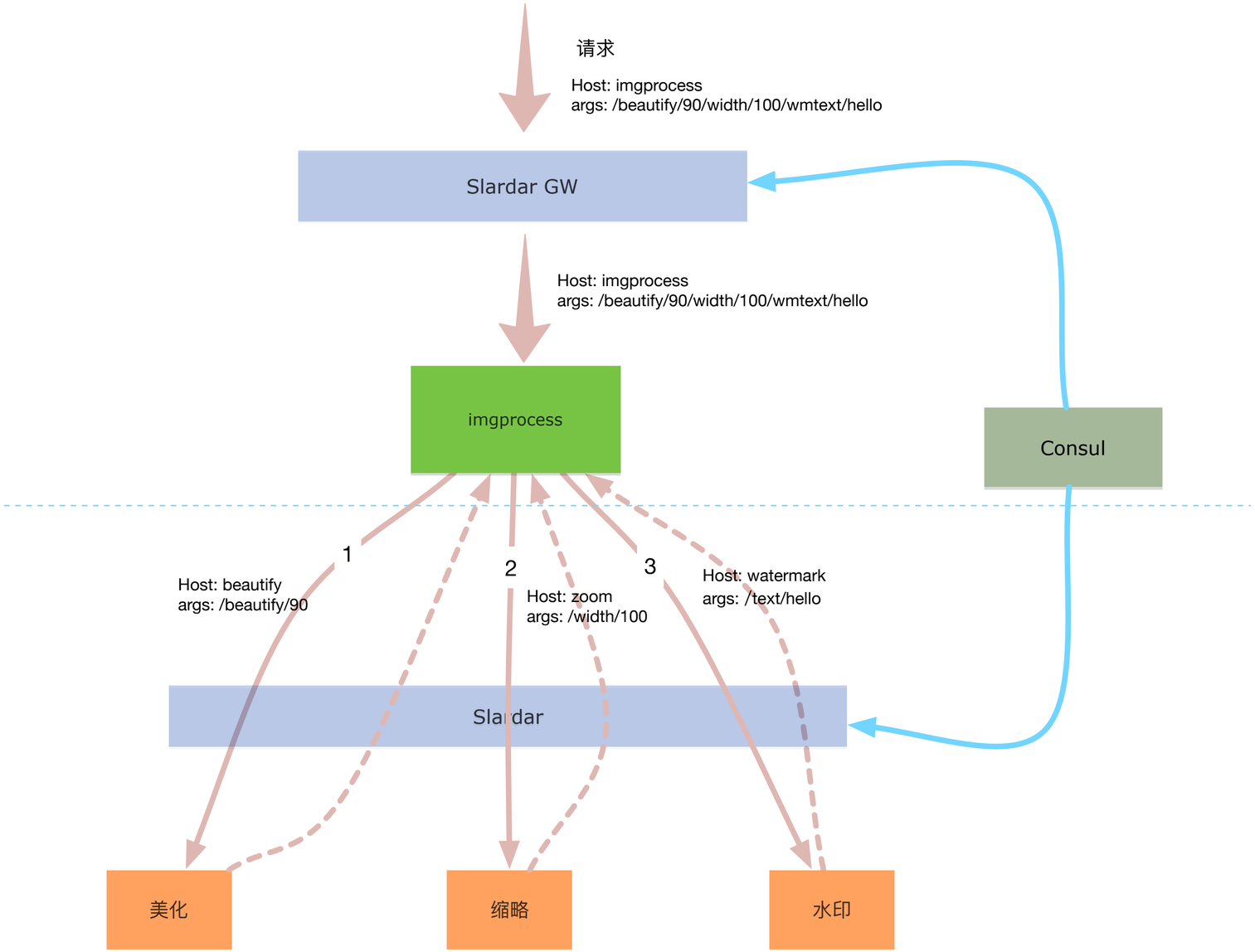


... and scales by distributing these services across servers, replicating as needed.



图片来源：http://martinfowler.com/articles/microservices.html

服务拆分


请求
Host: imgprocess
args: /beautify/90/width/100/wmtext/hello
Slardar GW
Host: imgprocess
args: /beautify/90/width/100/wmtext/hello
imgprocess
Consul
1
Host: beautify
args: /beautify/90
2
Host: zoom
args: /width/100
3
Host: watermark
args: /text/hello
Slardar
美化
缩略
水印

Talk is cheap. Show me the code.

开源!

Q & A

关注又拍云微信公众号，
获取更多干货！

TKANKS!

主办方：OpenResty 又拍云