
Architect
SACC
2022 中国系统架构师大会
SYSTEM ARCHITECT CONFERENCE CHINA 2022
激发架构性能 点亮业务活力
云上会议 网络直播 | 2022年10月27-29日
ITPUB

迈向多云时代

趣丸多云架构演进

趣丸科技/架构师/黄金

关于我

黄金 趣丸科技 资深架构师

- 广州趣丸网络科技有限公司
- 负责趣丸基础架构组
- 五年以上基础架构与容器相关平台经验

目录

1. 趣丸云端架构的发展历程
2. 多云的优势与挑战
3. 趣丸多云解决方案
4. 未来展望

趣丸云端架构的发展历程

单云时代 → 多云1.0 → 多云2.0

单云时代

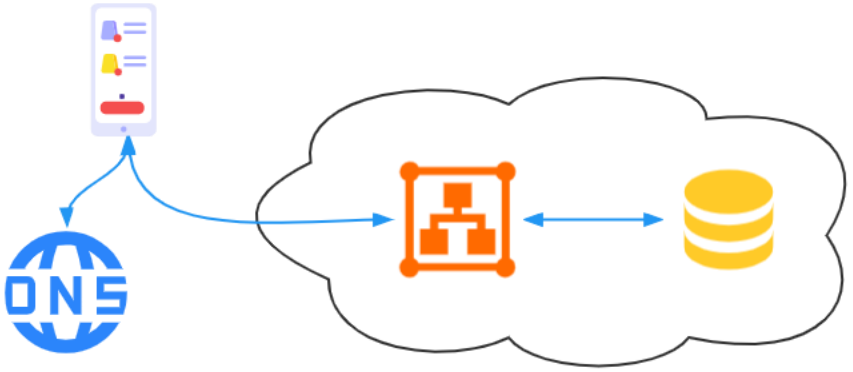
基础设施都在一个云

优点：

简单，快速实现，充分复用云服务商能力

缺点：

可靠性差，完全依赖于云服务商的可靠性保证。



多云1.0

业务层实现多云部署，状态层依然是单云，入口流量按比例调度。

优点：

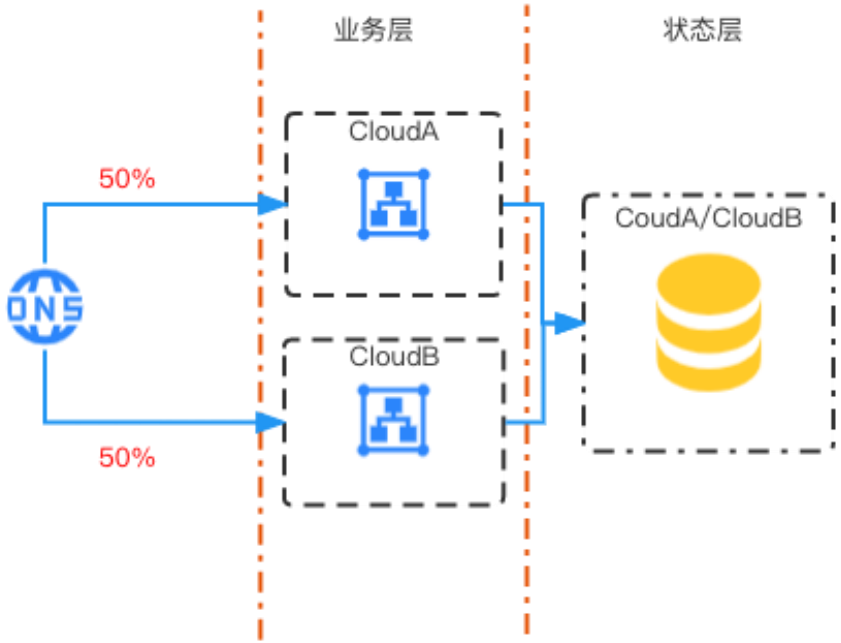
很好的避免业务层的故障，一定程度上避免云服务商基础设施故障。

对业务要求小，基本不用改造

缺点：

满足了业务层的容灾要求。

状态层故障，业务依然全部受损。



多云2. 0

多入口智能调度，状态层分离

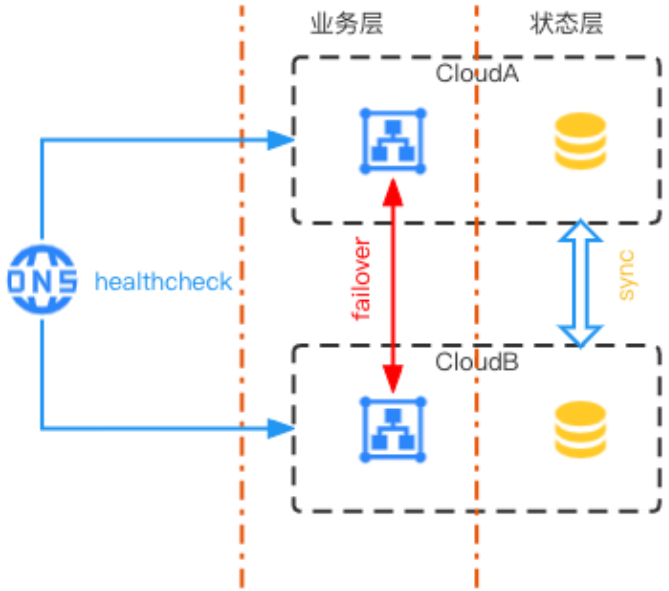
特点：

多云多活，可靠性最高

任何一边故障完全不影响业务

缺点：

复杂度最高，业务需要改造

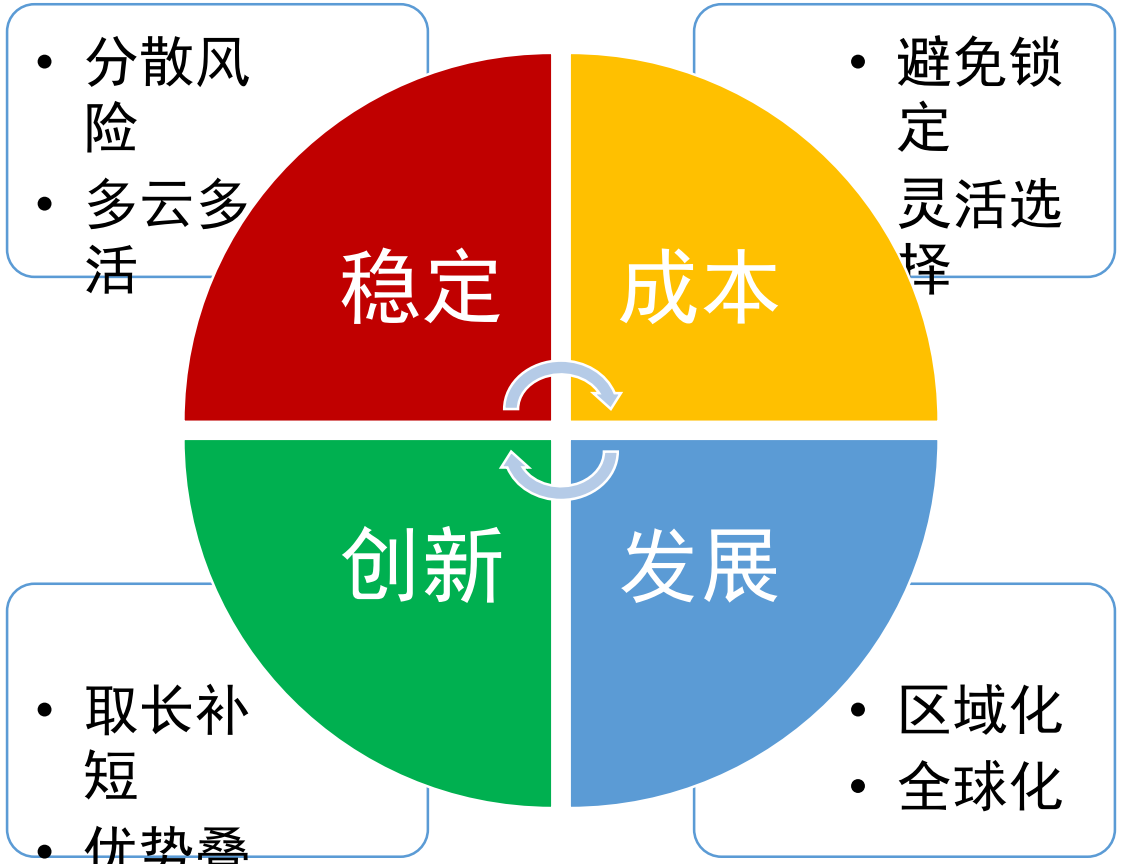


没有银弹

极端的可用性一般都会带来极端的复杂度

目录

多云的优势



多云带来的挑战

跨云互联互通

跨云应用管理问题与流量控制

异构基础设施带来的管理复杂性

多身份识别问题

.....

目录

1. 趣丸云端架构的发展历程
2. 多云的优势与挑战
3. 趣丸多云解决方案
   - 多云互联互通
   - 应用管理与流量控制
4. 未来展望

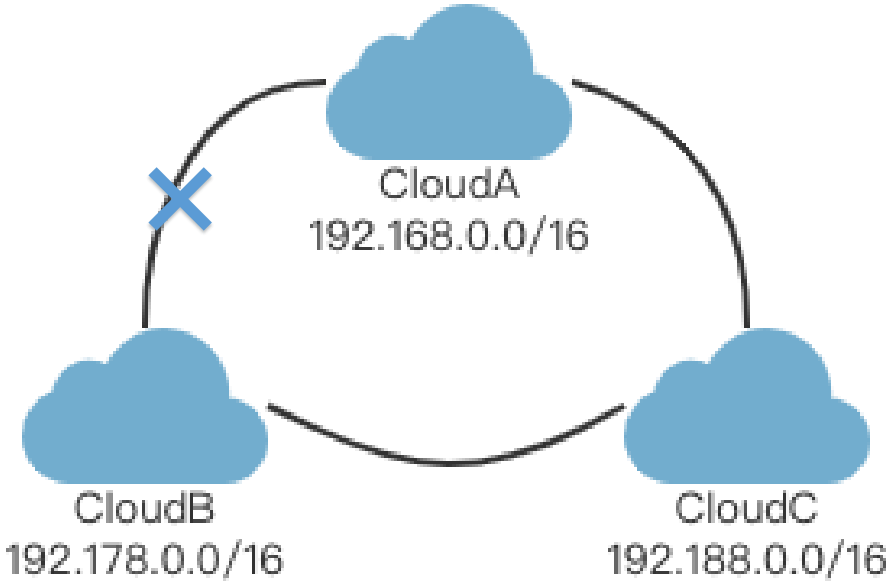
多云互联互通

- 基于多云的骨干网络建设
- 南北向流量高可用方案

基于多云的骨干网络建设

传统静态路由表方式

路由表	目的网段	下一跳
Cloud	192.178.0.0/16	CloudC
	192.188.0.0/16	CloudC
CloudA	192.168.0.0/16	CloudC
	192.188.0.0/16	CloudC
CloudB	192.168.0.0/16	CloudA
	192.178.0.0/16	CloudB

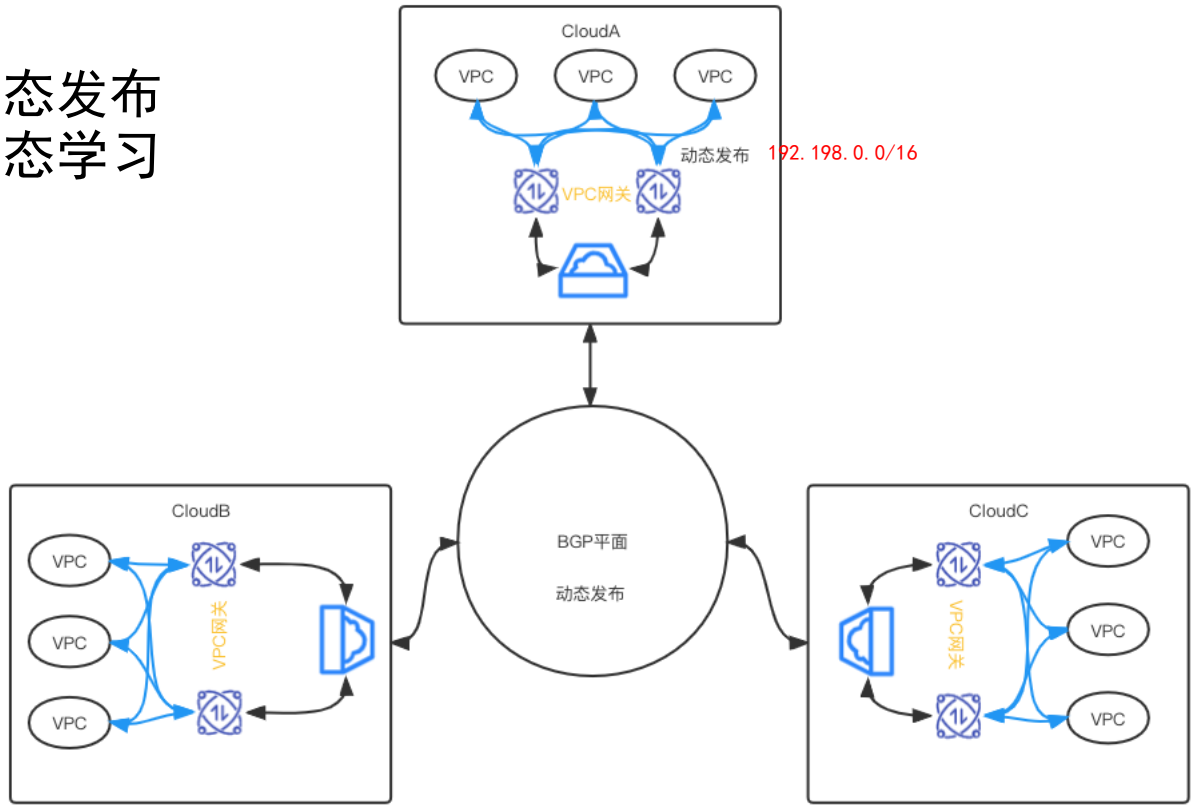


基于多云的骨干网络建设

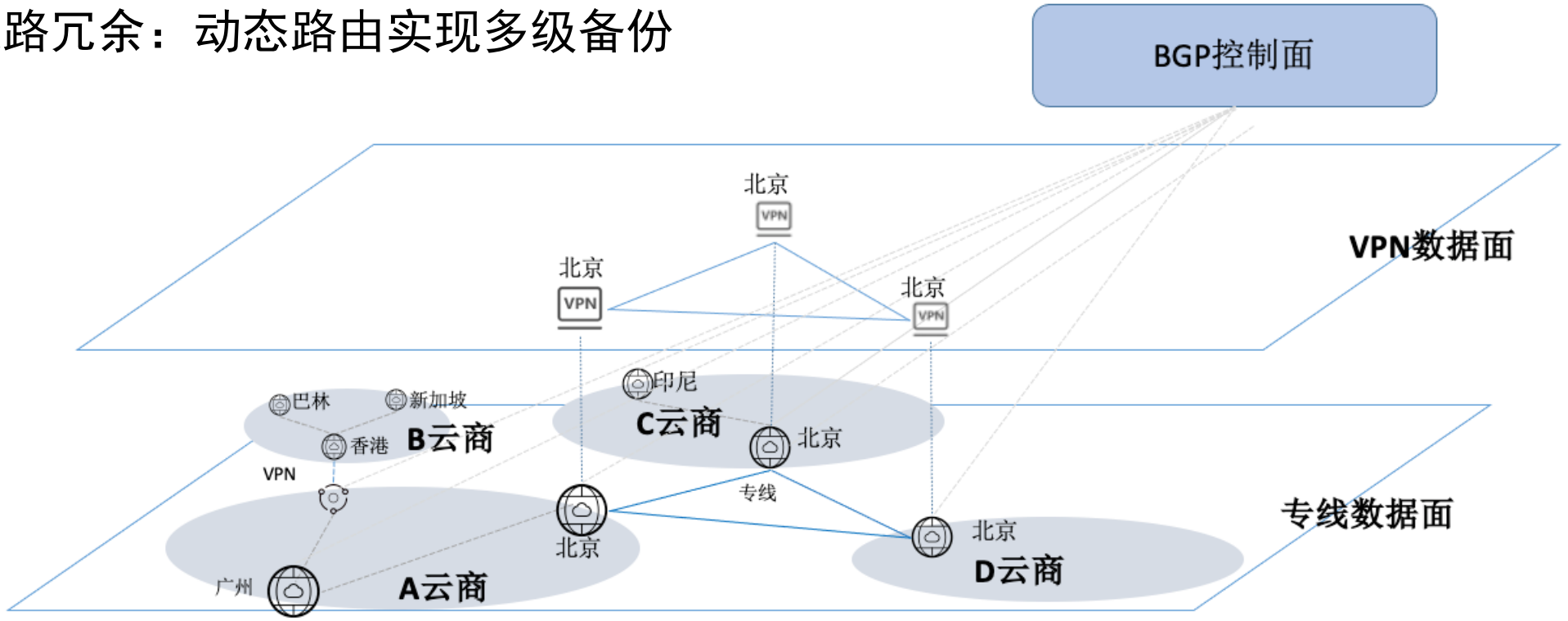
动态路由表方式

路由表	目的网段	下一跳
CloudA	192.178.0.0/16	CloudB
	192.188.0.0/16	CloudC
CloudB	192.168.0.0/16	CloudA
	192.198.0.0/16	CloudA
	192.188.0.0/16	CloudC
CloudC	192.168.0.0/16	CloudA
	192.198.0.0/16	CloudA
	192.178.0.0/16	CloudB

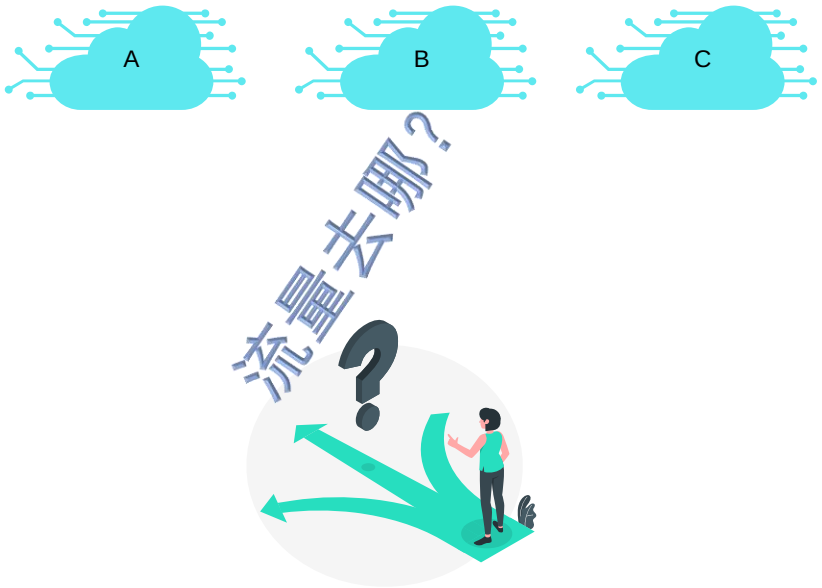
动态发布
动态学习



全球一张网：多云能力公用，区域灵活选择
多级链路冗余：动态路由实现多级备份



南北向流量高可用



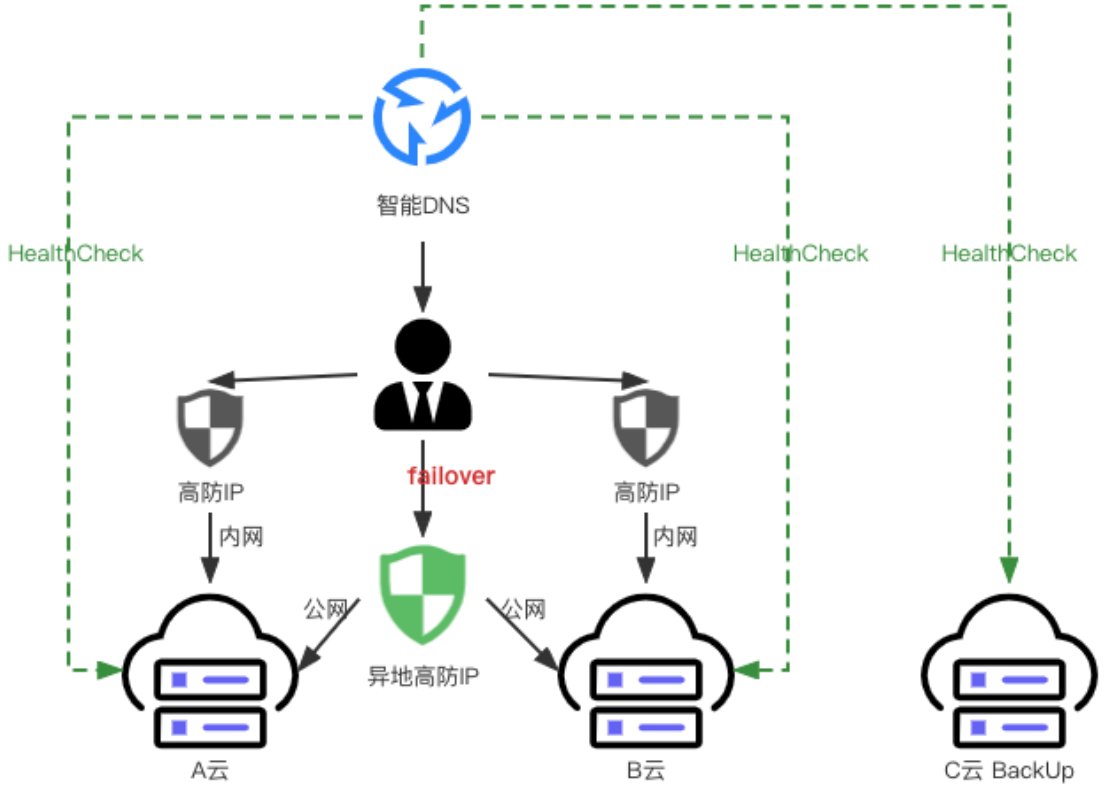
主备域名

HTTP DNS ✓

智能 DNS ✓

灵活度高
业务适配容易

南北向流量高可用



DNS

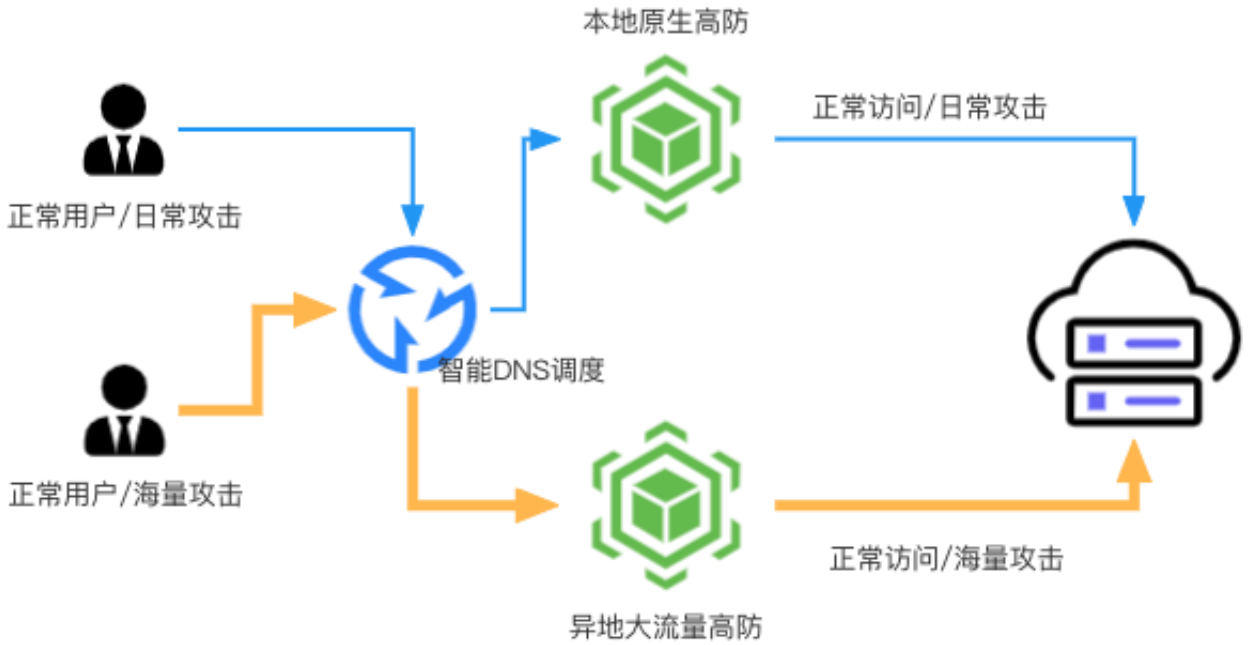
- 智能 DNS（智能全局流量管理）解决互联网接入层多活、主备接入的切换，通过健康检测，实现故障节点自动剔除
- HTTP DNS 绕开运营商 Local DNS，防止域名劫持和封堵风险

高防

- 优先本地高防：低延时，低成本
- 攻击自动切换异地高防：高可靠

南北向流量高可用

- 默认原生高防提供防护能力
- 当遭受海量流量攻击，调度流量到大容量异地高防进行清洗



应用管理与流量控制

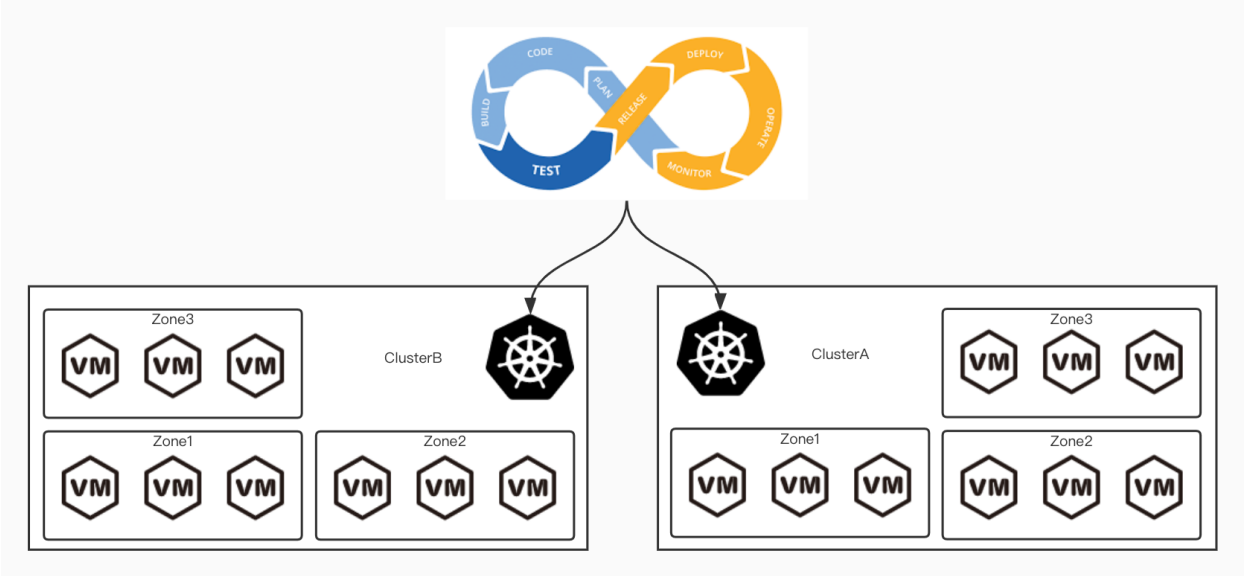
基于Kubernetes和Istio的多云流量调度与容灾方案

1. Kubernetes多集群管理方案

2. 基于Istio的跨Region/Zone的东西向流量调度

Kubernetes多集群管理

- Kubernetes单云跨Zone部署
- 应用通过CI/CD部署多个集群
- 每个集群相互独立



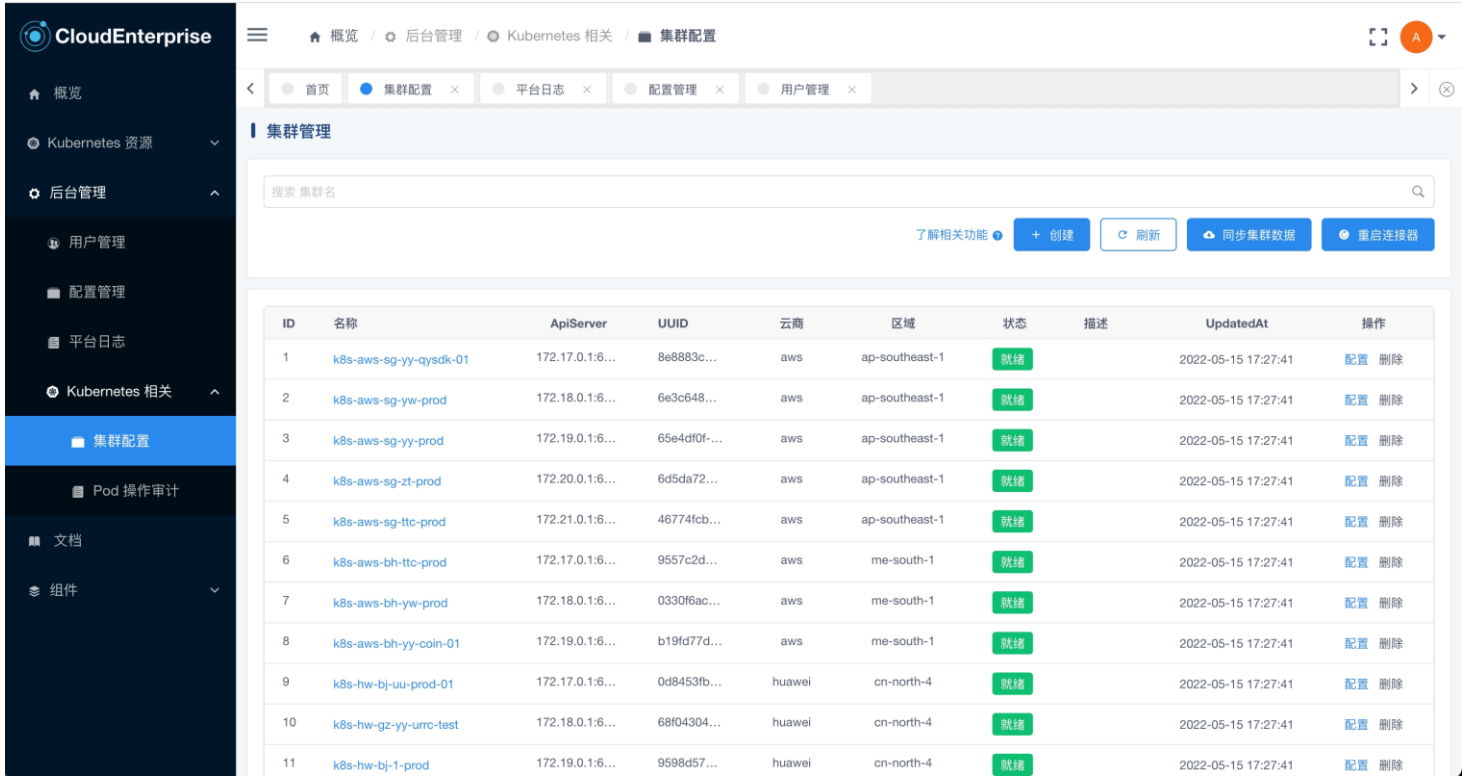
Kubernetes多集群管理

为什么没用跨集群管理工具？

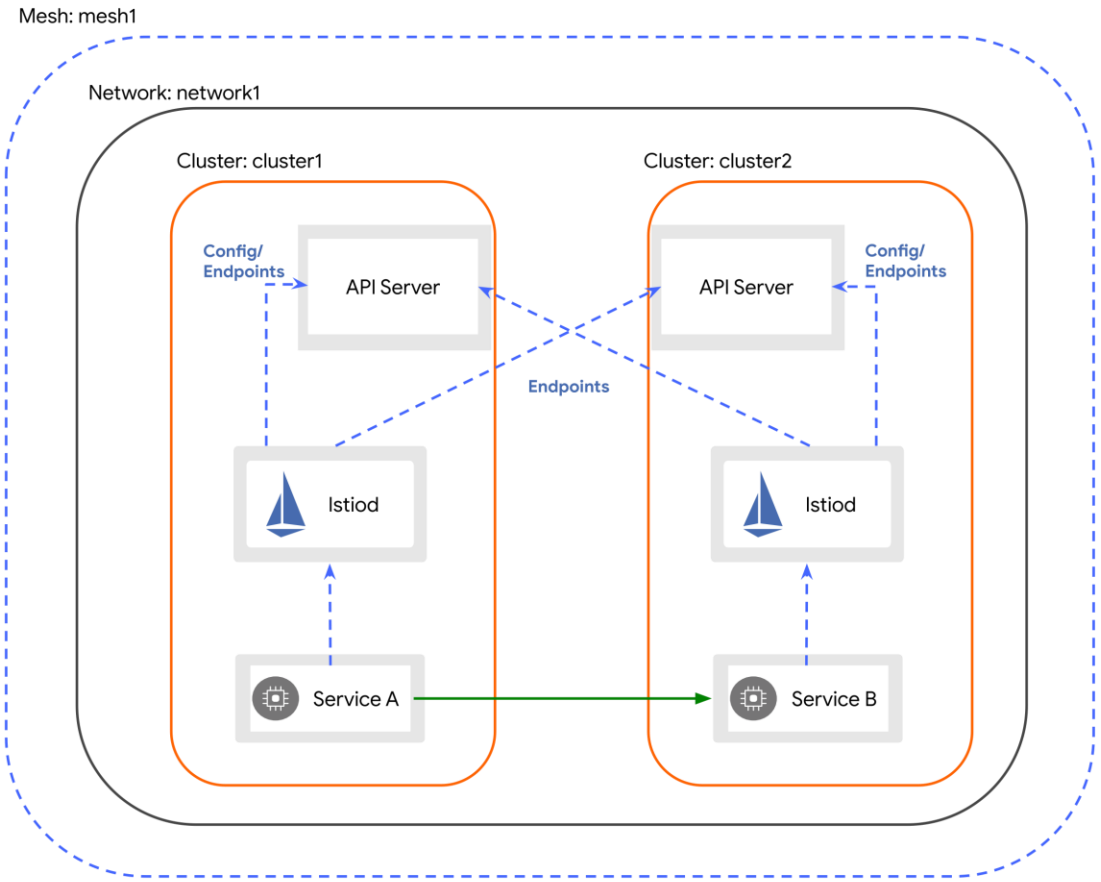
A. 没有跨集群调度的强需求

B. 满足一定隔离性要求

C. 多集群管理没有实际标准



基于Istio的东西向流量管理



Istio 多主架构

每一个 Istiod 都能发现其他所有集群的 endpoint。
应用可以跨集群访问其他集群的服务。

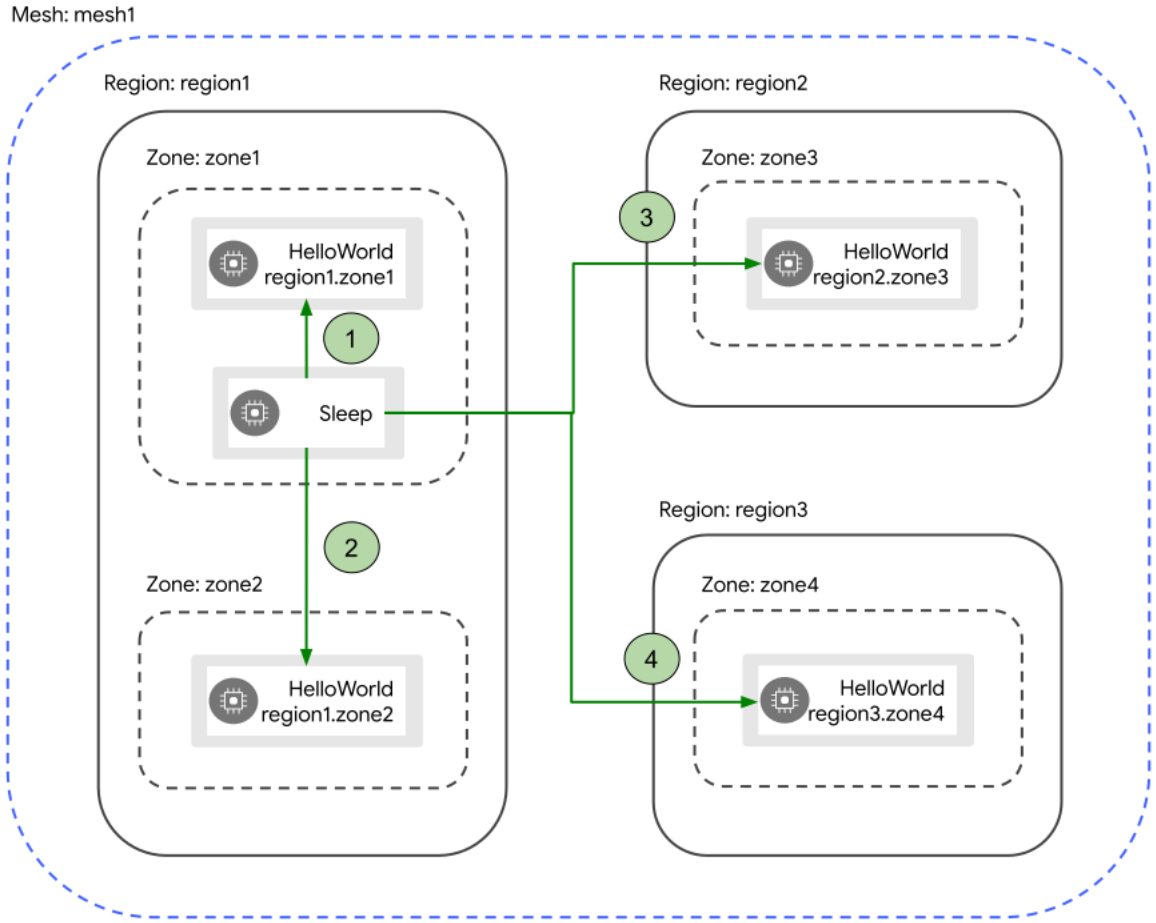
基于Istio的东西向流量管理

Istio流量管理策略：本地优先

优先本region本zone

本zone失效，优先本region的其他zone

本region失效，访问其他region的



基于Isito多主集群的实践

1. 在大规模集群中控制面性能瓶颈问题
2. 数据面带来的性能消耗问题

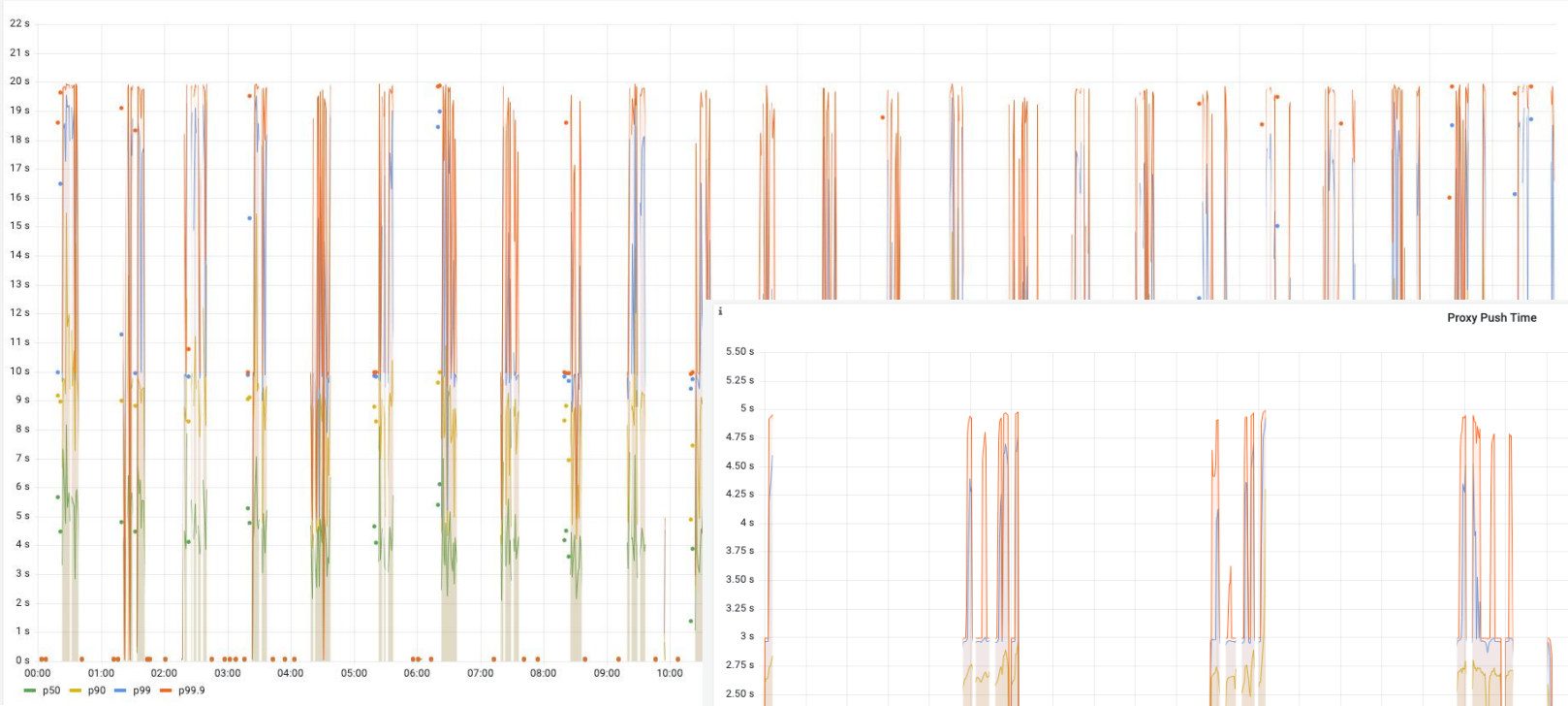
基于Isito多主集群的实践

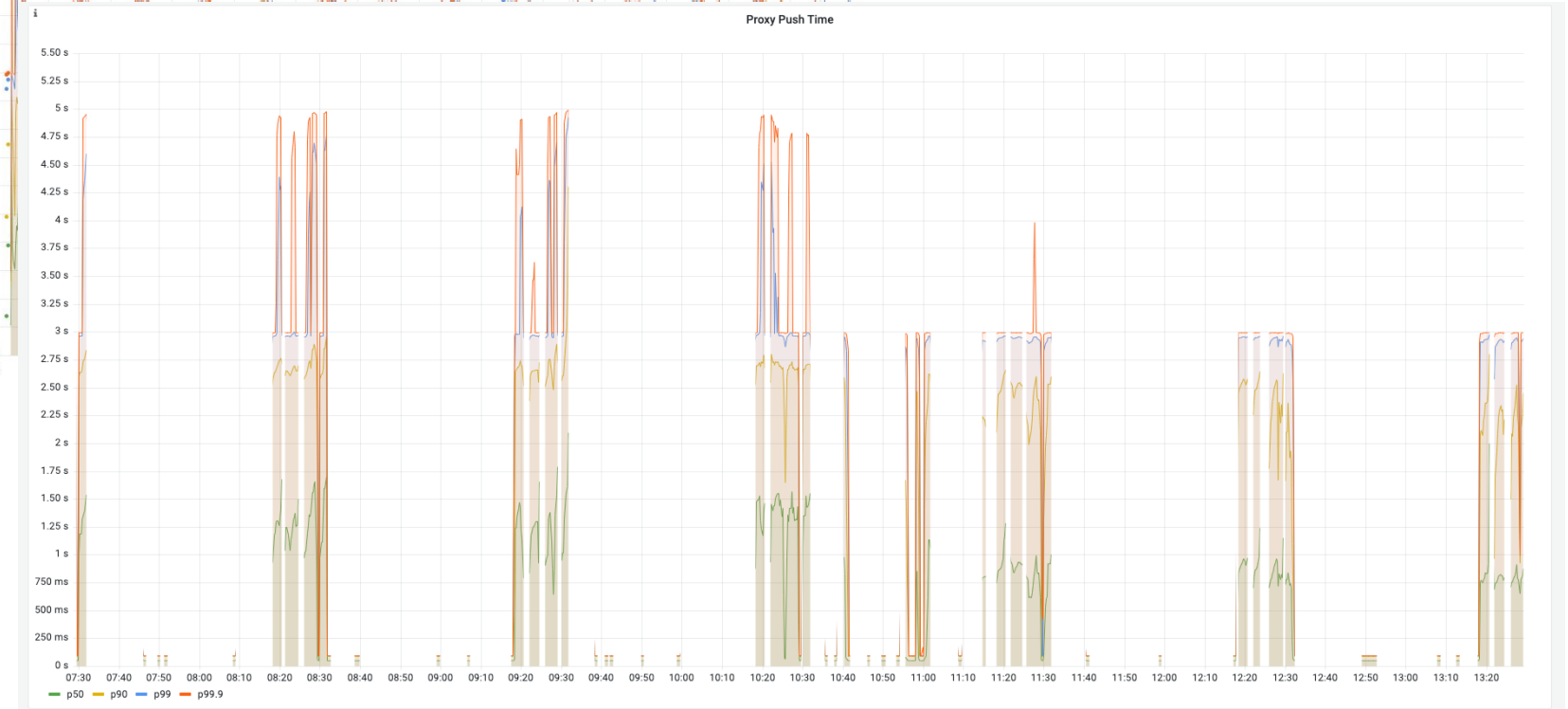
Istiod在大规模集群中的问题：

1. 消耗大量CPU
2. xds下发延时
3. 服务发现缺少

1. 扩容istiod，固定副本数
2. 减少下发xds与发现endpoints
   - DiscoverySelector
   - Sidecar
3. 优化xds参数
   - PILOT_DEBOUNCE_AFTER
   - PILOT_PUSH_THROTTLE
   - PILOT_DEBOUNCE_MAX
4. Istiod的一个BUG
   - PILOT_REMOTE_CLUSTER_TIMEOUT
   - PILOT_ENABLE_CROSS_CLUSTER_WORKLOAD_ENTRY

优化效果


22 s
21 s
20 s
19 s
18 s
17 s
16 s
15 s
14 s
13 s
12 s
11 s
10 s
9 s
8 s
7 s
6 s
5 s
4 s
3 s
2 s
1 s
0 s
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
p50
p90
p99
p99.9


Proxy Push Time
5.50 s
5.25 s
5 s
4.75 s
4.50 s
4.25 s
4 s
3.75 s
3.50 s
3.25 s
3 s
2.75 s
2.50 s
2.25 s
2 s
1.75 s
1.50 s
1.25 s
1 s
750 ms
500 ms
250 ms
0 s
07:30
07:40
07:50
08:00
08:10
08:20
08:30
08:40
08:50
09:00
09:10
09:20
09:30
09:40
09:50
10:00
10:10
10:20
10:30
10:40
10:50
11:00
11:10
11:20
11:30
11:40
11:50
12:00
12:10
12:20
12:30
12:40
12:50
13:00
13:10
13:20
p50
p90
p99
p99.9

基于Isito多主集群的实践

Sidecar问题

1. 时延增高
2. CPU资源消耗过大

```go
// Uses the default concurrency 2, unless either overridden by proxy config to a positive number
// or special value 0, in which case the value is computed from CPU limits/requests.
func estimateConcurrency(cfg *meshconfig.ProxyConfig, annotations map[string]string, valuesStruct *opconfig.Values) int {
    if cfg != nil && cfg.Concurrency != nil {
        concurrency := int(cfg.Concurrency.Value)
        if concurrency > 0 {
            return concurrency
        }
        if limit, ok := annotations[annotation.SidecarProxyCPULimit.Name]; ok {
            out, err := quantityToConcurrency(limit)
            if err == nil {
                return out
            }
        } else if request, ok := annotations[annotation.SidecarProxyCPU.Name]; ok {
            out, err := quantityToConcurrency(request)
            if err == nil {
                return out
            }
        } else if resources := valuesStruct.GetGlobal().GetProxy().GetResources(); resources != nil { // nolint: staticcheck
            if resources.Limits != nil {
                if limit, ok := resources.Limits["cpu"]; ok {
                    out, err := quantityToConcurrency(limit)
                    if err == nil {
                        return out
                    }
                }
            }
            if resources.Requests != nil {
                if request, ok := resources.Requests["cpu"]; ok {
                    out, err := quantityToConcurrency(request)
                    if err == nil {
                        return out
                    }
                }
            }
        }
    }
    return 2
}
```

concurrency

eads to run. If unset,
determined based on
t to 0, all cores on the
ault is 2 worker

traffic.sidecar.istio.io/excludeOutboundIPRanges
traffic.sidecar.istio.io/excludeOutboundPorts

accessLog
tracing.sampling
...

目录

我们的思考

基础设施即代码

零信任网络

基础设施即代码

差异化的云服务产品对基础设施管理带来了较大冲击。

管理流程不同

云服务商 API 不同

功能特性不同

基础设施即代码

能帮我们解决什么问题？

1. 云无关
2. 统一的资源生命周期管理
3. 快捷的环境复制与迁移
4. 标准化、自动化、可视化

零信任网络

传统安全边界的问题：

- 边界的假设是不合理的
- 只在边界做认证
- 基于IP/MAC认证

多云情况下，问题更加突出：

- 云厂商安全能力不尽相同
- 云厂商也不可信任
- 核心数据都在云端
- 全球化，边界分布在世界范围内
- 在云环境下，业务更容易获得公网 IP

核心思路

微边界/永远验证

1. Network 分段：入口/出口云微边界和更深的微分段
2. 流量加密：所有流量 TLS 加密
3. 威胁防护：基于多种策略云主机与业务防护

THANKS