模块13 Kubernetes集群联邦和Istio多集群管理

多集群治理的驱动力

- 分布式云的好处
  - 成本优化
  - 更好的弹性以及灵活性
  - 避免厂商锁定
  - 容灾
  - 数据保护以及风险管理
  - 提升响应速度（应该就是架构实战营里的分区架构就近访问带来的）
- 分布式云的挑战
  - Kubernetes单集群承载能力有限
  - 异构基础设施
  - 配置变更以及下发
  - 存量资源接入（比如淘汰资源再利用）
  - 跨地域、机房应用的部署和管理
  - 异地容灾、多活
  - 弹性调度以及自动伸缩
  - 监控告警

集群联邦

必要性

- etcd存储的限制
- API server缓存占用内存的限制
- kubernetes控制器缓存占用内存的限制
- 控制器的处理耗时随着对象的增多而不断增长
- 单个计算节点资源上限
  - CPU、内存等可量化资源
  - 端口、进程数等不可量化资源
- 故障域控制
- 高可用部署——异地容灾和多活

职责

- 跨集群同步资源——比如可以将一个Deployment部署到多个集群中
- 跨集群服务发现——汇总各个集群的服务和ingress，暴露到全局DNS服务器中
- 高可用——动态调整每个集群的应用实例，隐藏具体的集群信息
- 避免厂商锁定

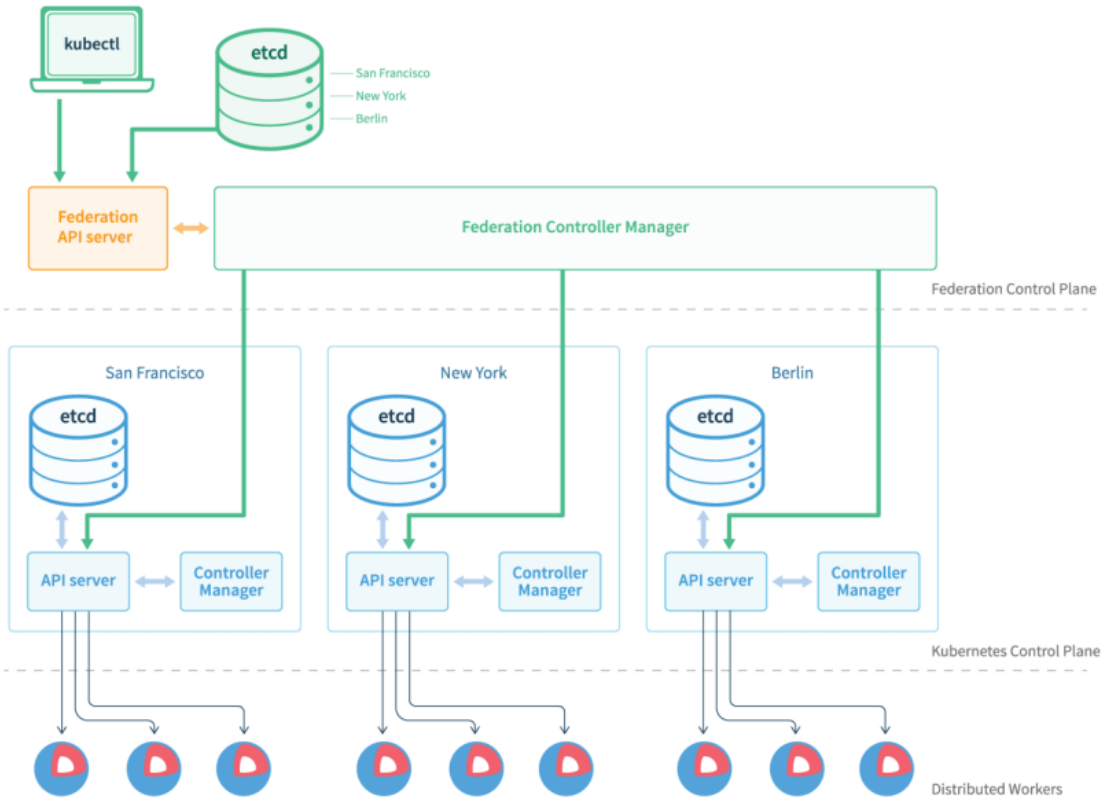
混合云

混合云特指将公有云和私有云整合在一起的统一云平台。可支持复杂业务场景

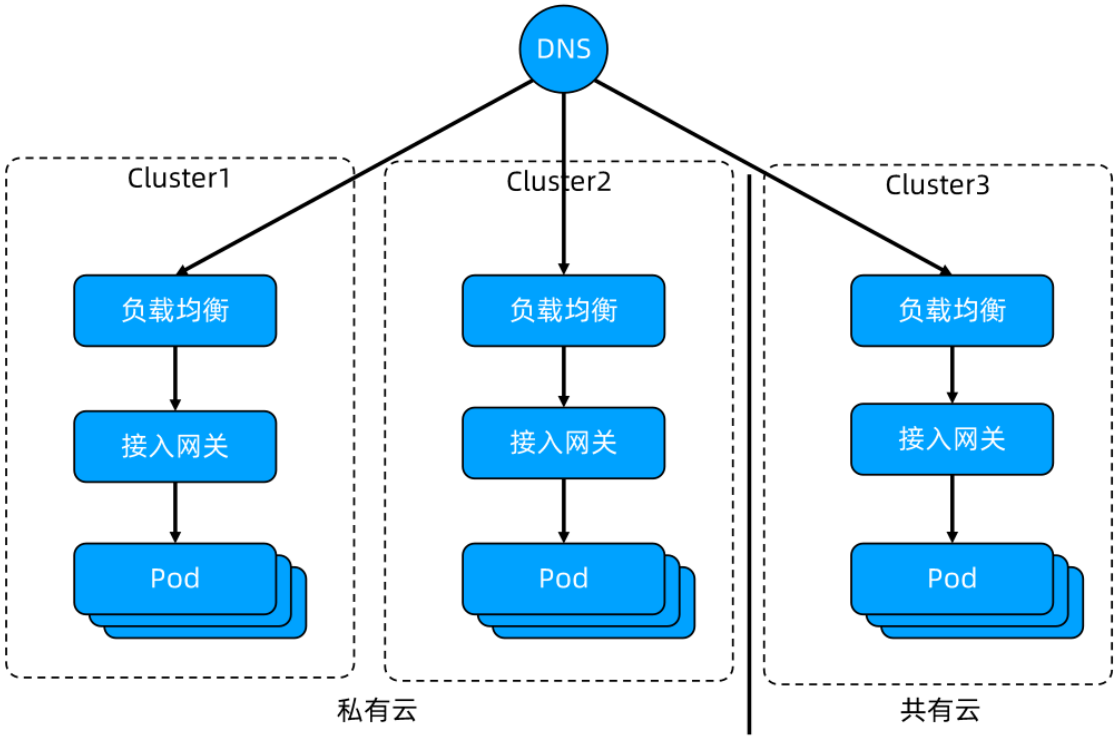
- Cloud Burst。
- 如 12306，低流量时运行在本地数据中心，当请求量突然爆发的时候，可以直接在公有云扩容，节省数据中心成本。
- 可将业务分为包含敏感数据和不包含敏感数据的业务，将敏感数据业务运行在私有云，将非敏感数据业务运行在公有云。
- 可重复利用公有云提供商数据中心靠近边缘的能力。

集群联邦的定义

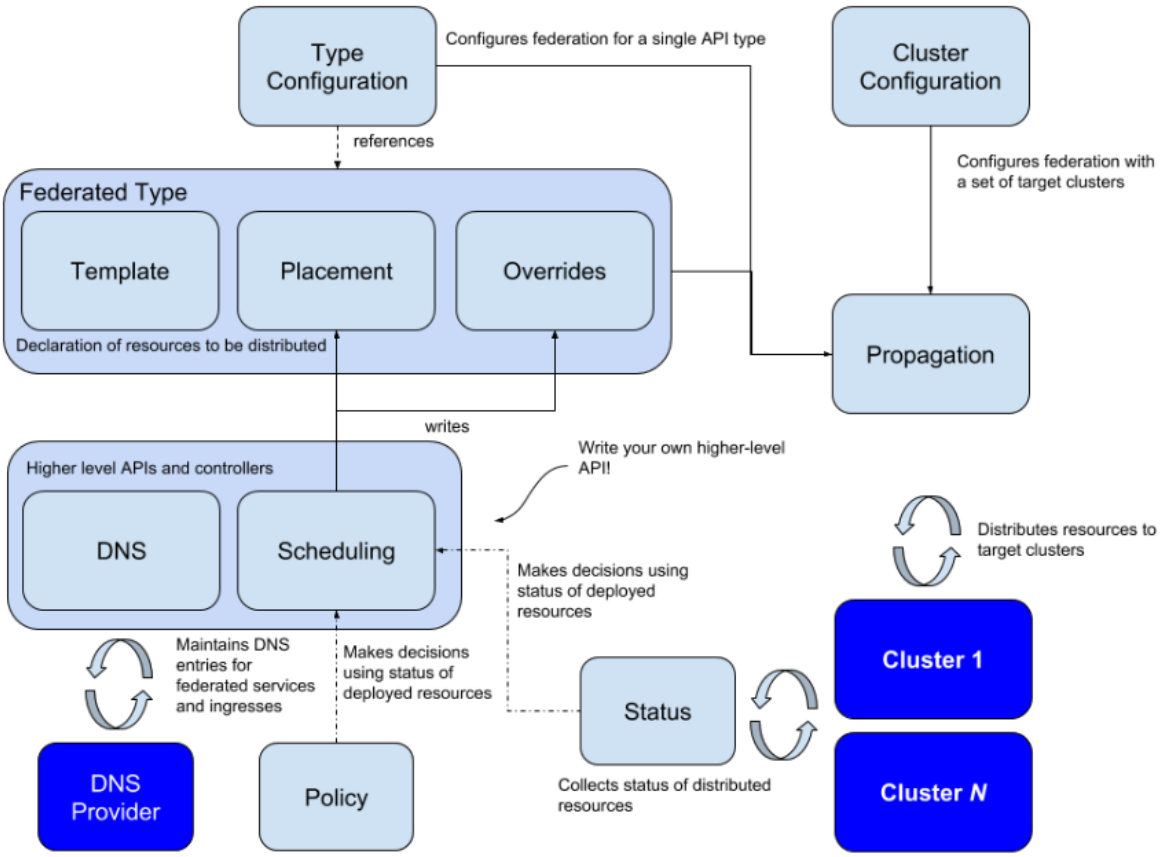
将多个K8S集群注册到统一控制平面，并为用户提供统一API入口的多集群解决方案。提供在全局层面对应用的描述能力，并将**联邦对象**实例化为kubernetes对象，分发到联邦下辖的各个成员集群中。



基于集群联邦的高可用应用部署



核心架构



- Federated Type，联邦类型对象
  - Template，定义集群对象的模板

```
apiVersion: types.federation.k8s.io/v1alpha1
kind: FederatedDeployment
metadata:
 name: sample-federated-deployment
spec:
 template:
  metadata:
 spec:
    replicas: 2
    template:
     spec:
       containers:
       - image: nginx
         name: nginx
 placement:
 # ....
 overrides:
 # ....
```

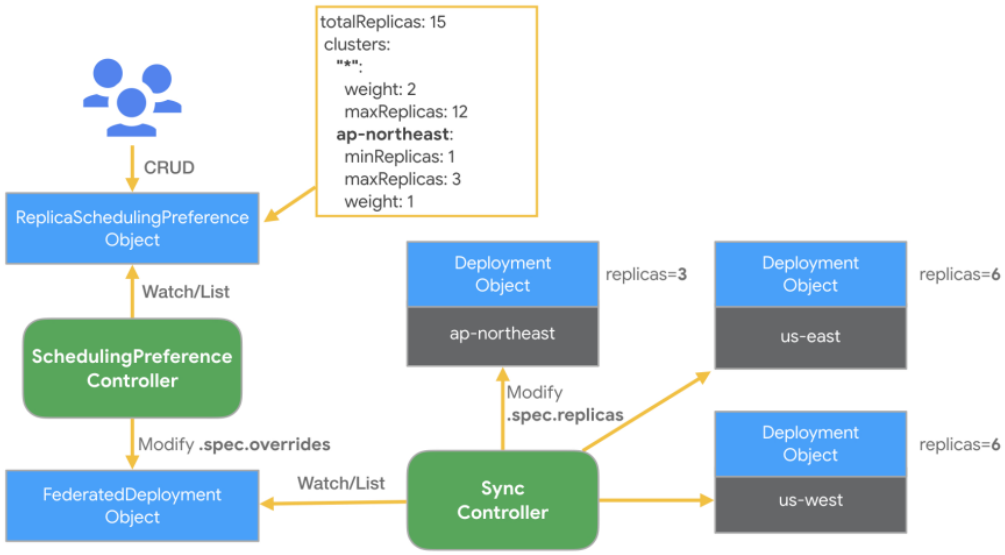
- Placement，指定联邦对象的目标集群，即部署哪个集群。取值可以是具体的集群名单（优先级高），也可以是clusterSelector选择的对应集群（优先级低）；因为有前者的优先级高，所以即使提供了一个空的名单，clusterSelector也会被忽略

```
apiVersion: types.federation.k8s.io/v1alpha1
kind: FederatedDeployment
metadata:
 name: sample
 namespace: sample
spec:
 template:
  # .....
 placement:
  clusters:
  - name: cluster1
  - name: cluster2
  clusterSelector:
   matchLabels:
    region: china
    zone: shanghai
 overrides:
  #....
```

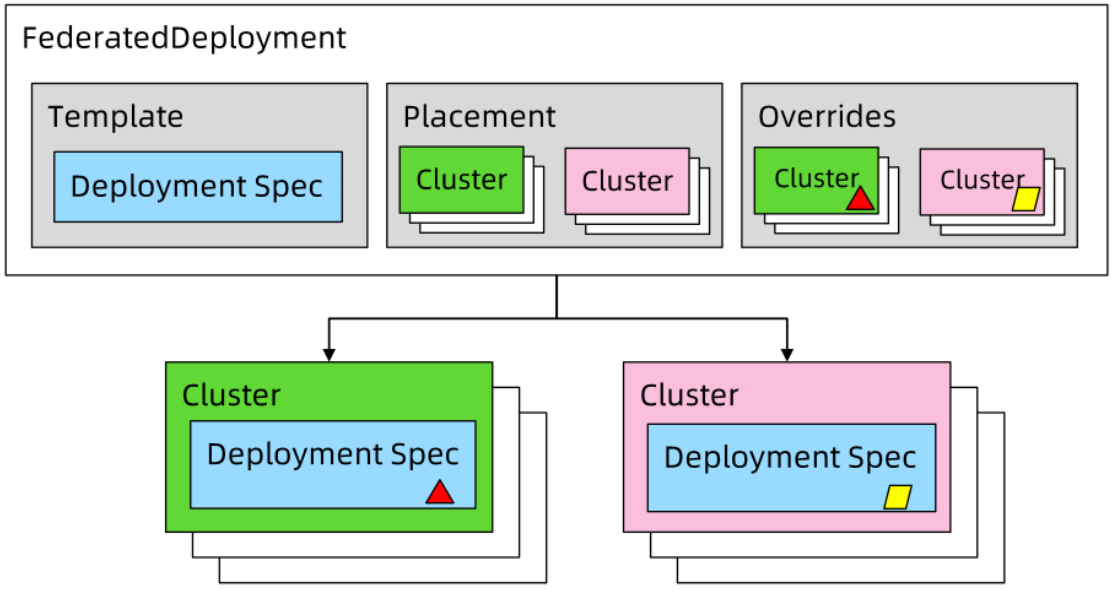
- Overrides，将联邦对象分发到下属集群时，根据集群的分布情况，重写某些属性，针对不同集群调整模板，以更好地发挥网络、计算资源、存储等的优势。不支持对列表的重写。

```
apiVersion: types.federation.k8s.io/v1alpha1
kind: FederatedDeployment
metadata:
spec:
 template:
  spec:
   replicas: 2
   template:
    # ....
 placement:
  clusters:
  - name: cluster1
  - name: cluster2
 overrides:
 - clusterName: cluster2
  clusterOverrides:
  - path: /spec/replicas
   value: 3
```

能够基于副本总数于集群的策略定义对Deployment进行编排，比如6个副本的Deployment，指定集群A有3个副本，集群B有2个副本，集群C有1个副本。编排策略是通过建立ReplicaSchedulingPreference(RSP)进行的，如下图：



联邦类型对象整体情况如下



- Propagation，将Overrided过的对象发布到下属集群中
- Status，每个集群都有自己的状态
- Scheduling根据Policy，基于Status进行调度
- 服务部署以后，根据服务的部署情况，DNS Provider配置DNS。**多集群DNS在2020年已经被移出。**

集群联邦管理的对象

> 现在主流是集群联邦V2。

工具集Kubefedctl允许用户对单个对象动态创建联邦对象，动态对象的生成基于CRD

	Kubernetes 集群	Kubernetes 集群联邦
计算资源	Node	Cluster
部署	Deployment	FederatedDeployment
配置	Configmap	FederatedConfigmap
密钥	Secret	FederatedSecret
任何其他 Kubernetes 对象	Foo	FederatedFoo

支持的核心对象

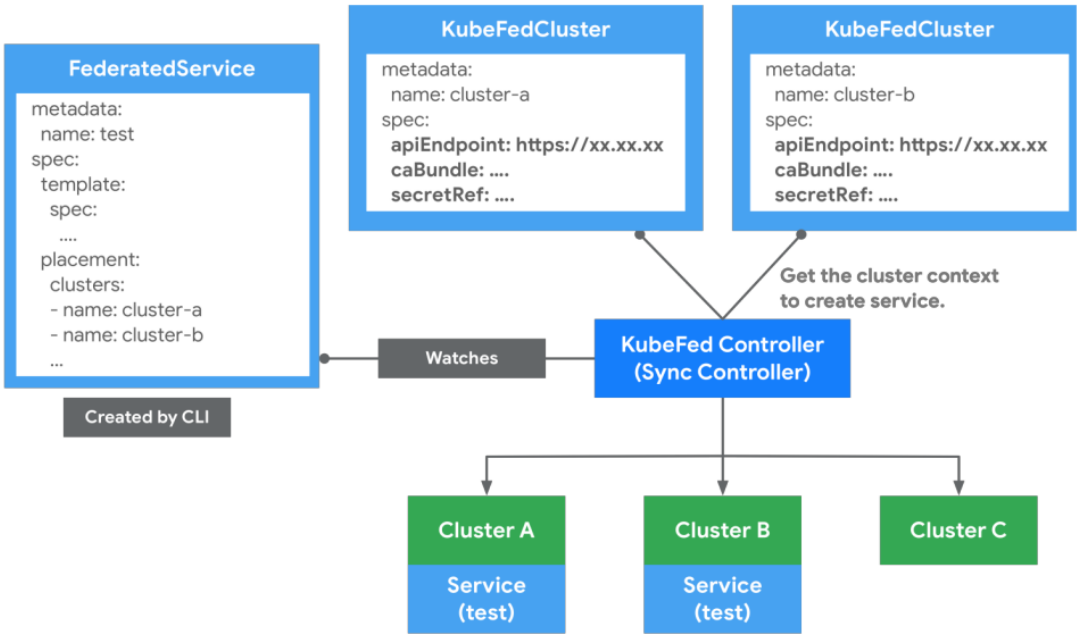
API Group	用途
core.kubefed.k8s.io	集群组态、联邦资源组态、KubeFed Controller 设定档等。
types.kubefed.k8s.io	被联邦的 Kubernetes API 资源。
scheduling.kubefed.k8s.io	副本编排策略。
multiclusterdns.kubefed.k8s.io	跨集群服务发现设定。

集群注册中心

- 联邦下辖集群清单、认证信息、状态信息保存在集群注册中心中。
  - 注册信息使用单独的KubeFedCluster对象进行，除了注册信息，还包括集群的健康状态、域名等。
  - 因为集中进行了管理，所以当集群出现问题时，可以直接通过集群状态信息获知控制器异常的原因
- 集群联邦本身不提供算力，只承担集群的协调工作

集群类型

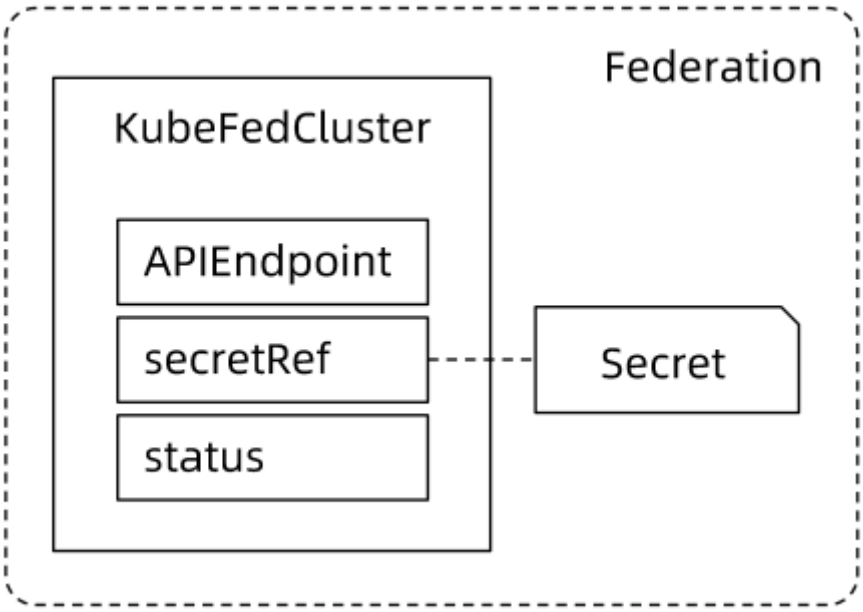
- Host，用于提供KubeFed API与控制平面的集群，本质上就是联邦的控制面
- Member，通过KubeFed API注册的集群，提供身份凭证让KubeFed控制器存取它。
- 集群注册成功后，在联邦层面会生成一个KubeFedCluster对象来存储相关信息，这些信息被KubeFed控制器使用



定义示例如下：

```
apiVersion: core.kubefed.io/v1beta1
kind: KubeFedCluster
metadata:
 name: "cluster1"
spec:
 apiEndpoint: https://API Server.cluster1.example.com
 caBundle: LS0tLS****LS0K
 disabledTLSValidations:
 - '*'
 secretRef:
  name: cluster1-vhvfw
status:
 conditions:
 - reason: ClusterReady
  type: Ready
 region: China
 zones:
 - Shanghai
```

对象的结构示意如下：



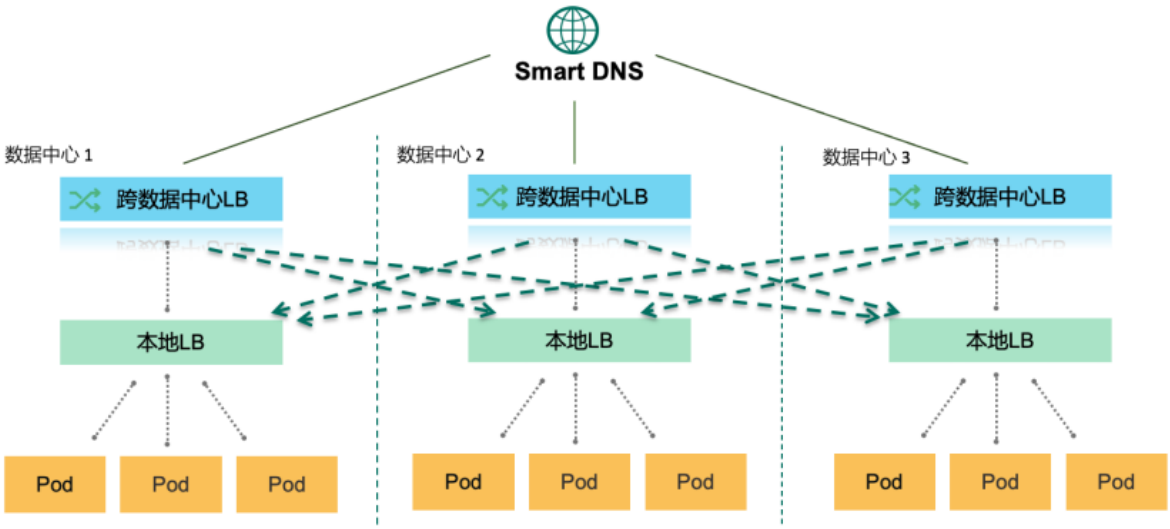
Clusternet

腾讯开源的类似集群联邦的解决方案。略。

Istio多集群

跨地域流量管理的挑战

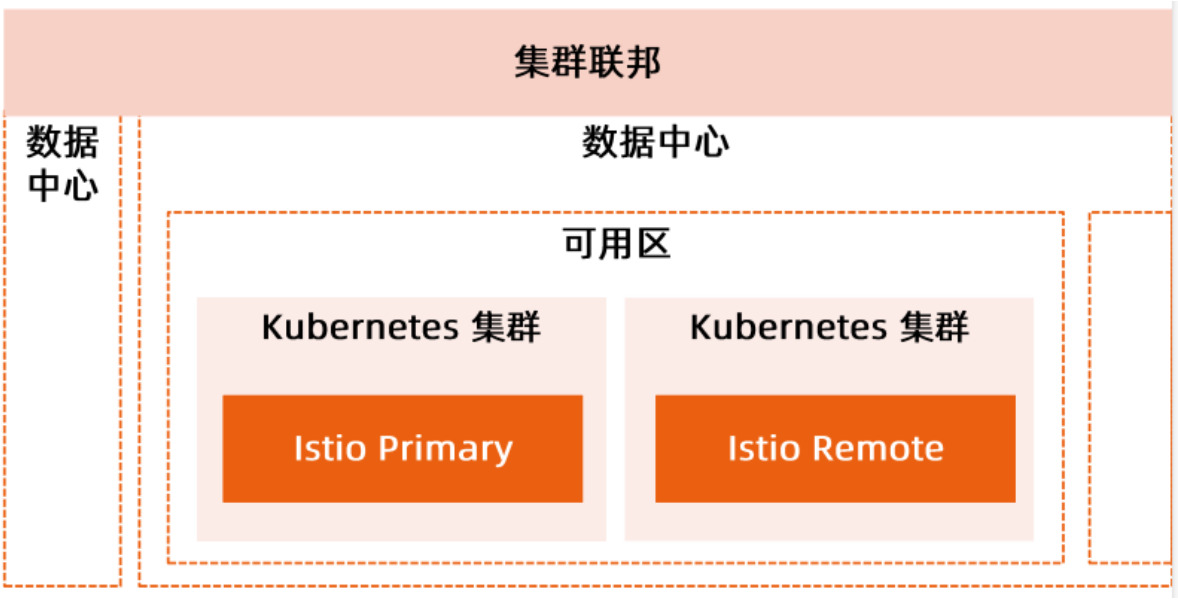
主要是解决跨地域数据多数据中心多活架构，同时实现分区架构下，优先访问本地域数据中心的服务以提高效率



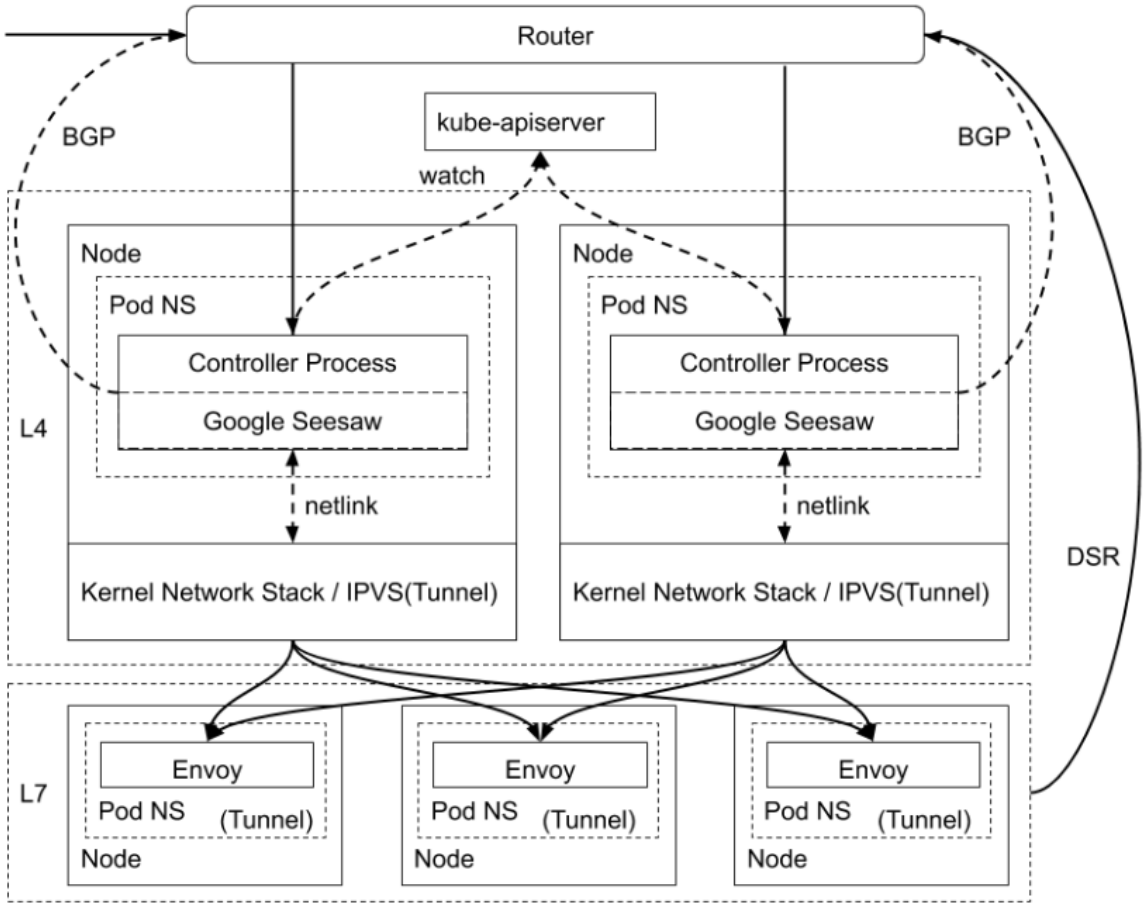
- 跨数据中心LB是为了在某个数据中心无法提供服务的时候，快速地将流量切换到其它数据中心，客服DNS的TTL延迟问题以及客户端缓存问题。
- 为什么要两级LB？因为如果只有一级的话，
    - 在POD数量非常多的情况下，POD状态的更新请求会压垮单级的负载
    - 要摘除某个数据中心，需要逐个POD的摘除，效率非常低；这相当于组织人多了，一级管理层级不行了，就要多级管理，类似于管理半径问题。

多集群部署

- 集群联邦下面有多个数据中心
- 数据中心下可能有多个可用区
- 每个可用区下可能有过个K8S集群
    - 一个可用区可设定一个网关集群，集中部署Istio primary；可以在网关集群使用ssl加速卡卸载TLS
    - 同可用区其它集群部署Istio remote。所有集群使用相同RootCA.



入站流量架构L4 + L7

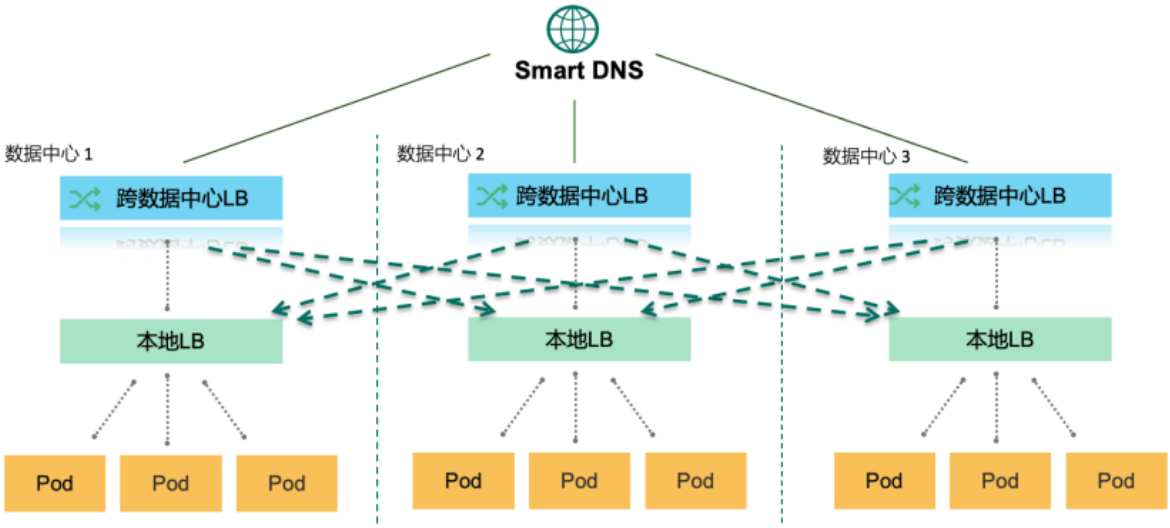


> 为了看懂上图，引用第8课的笔记
>
> 1. 上图中第1、2步骤中，路由器能把数据包路由给10.0.2.1的TOR，是因为IP为10.0.2.46的K82节点通过BGP把自己节点上可达的POD的路由关系交换给了TOR。
> 2. 为了进入K8S节点，根据service的原理，可能通过NodePort类型或者ClusterIP类型进入节点。到了K8S的节点，数据包被发送到网卡cali704c5adc642，这是一个veth pair设备，另外一头在Pod上，数据包进入容器内部。此时就需要POD之间的互通了，互通使用IPVS，过程就类似CNI的calico（参见模块7笔记《调度器和控制器》“Calico”小节）的隧道模式，就是上图中的第2步，外头再封装一层包头。

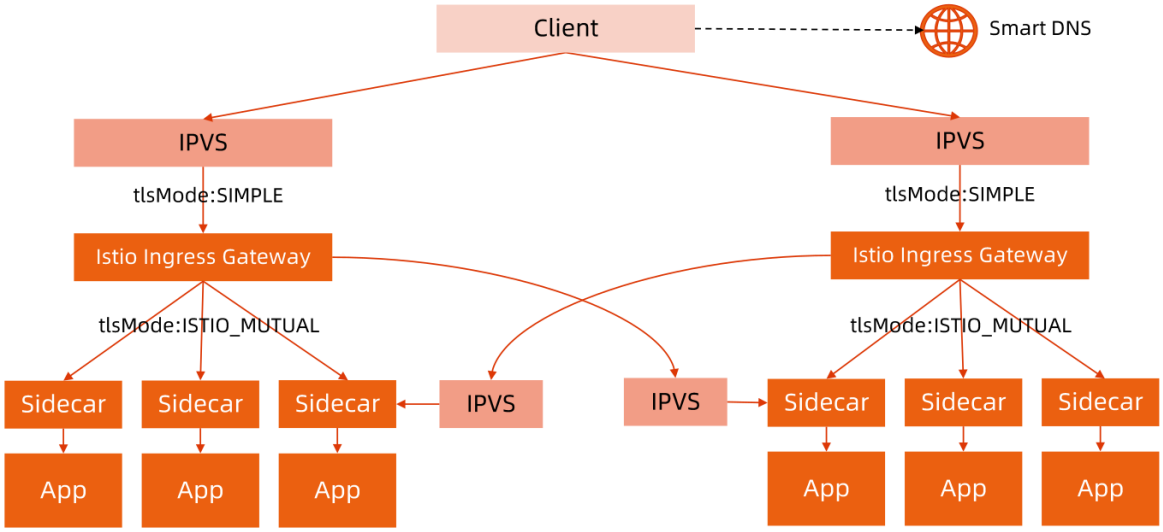
3. 到了Gateway Pod后，通过隧道端点的设备把外层包头去掉（上图第3步），进入Ingress组件的七层流量转发策略转到目的服务去处理。
4. 处理结果使用直接路由（DR）的形式回到10.0.3.1的TOR，最终通过路由器返回给客户端。

整体模型还是遵循**先经过service的4层流量控制，再到Ingress的7层流量控制。**

可以用Istio将下面架构减少一级LB设备



新架构如下：



具体做法如下：

- 上图中连接不同集群的下层IPVS指定义LoadBalancer类型的服务，提供集群外可访问的LoadBalancer IP。
- 然后在其它集群中定义指向远端集群LoadBalancer IP的WorkLoadEntry，以实现故障转移的目的。

> 参加第12模块笔记：
>
> WorkLoadEntry也是一个ServiceEntry，只是用workloadSelector指定真正的后端服务——也即WorkLoadEntry。总体上，相当于一个外部服务的“指针”或者“引用”

```yaml
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
 name: details-svc
spec:
 hosts:
  - details.bookinfo.com
 location: MESH_INTERNAL
 ports:
  - number: 80
    name: http
    protocol: HTTP
 resolution: STATIC
 workloadSelector:
  labels:
    app: details-legacy
```

```yaml
apiVersion: networking.istio.io/v1beta1
kind: WorkloadEntry
metadata:
 name: details-svc
spec:
 serviceAccount: details-legacy
 address: 2.2.2.2
 labels:
  app: details-legacy
  instance-id: vm1
```

指向其它数据中心LoadBalancer IP的WorkLoadEntry定义如下：

```yaml
apiVersion: networking.istio.io/v1beta1
kind: WorkloadEntry
metadata:
 name: foo
spec:
 address: foo.bar.svc.cluster2
 labels:
  run: foo
 locality: region1/zone1
```

- 定义ServiceEntry。ServiceEntry可以将本地POD和具有相同Label的WorkLoadEntry定义成相同的Envoy Cluster。

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: foo
spec:
  hosts:
  - foo.com
  ports:
  - name: http-default
    number: 80
    protocol: HTTP
    targetPort: 80
  resolution: STATIC
  workloadSelector:
    labels:
      run: foo
```

- 在VirtualService中使用ServiceEntry

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: foo
spec:
  gateways:
  - foo
  hosts:
  - foo.com
  http:
  - match:
    - port: 80
    route:
    - destination:
        host: foo.com
        port:
          number: 80
```

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: foo
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - foo.com
    port:
      name: http-default
      number: 80
      protocol: HTTP
```

- 为了在正常情况下优先访问本地服务，需要利用WorkLoad的Locality信息
  - Istio从如下配置中读取地域信息，基于它们可以为Istio中运行的所有workload添加地域属性

- Kubernetes Node 对象中的地域信息，所有 Pod 自动继承该 Locality 信息
  - region: topology.kubernetes.io/region
  - zone: topology.kubernetes.io/zone
  - subzone: topology.istio.io/subzone
- Kubernetes Pod 的 istio-locality 标签，可覆盖节点 Locality 信息
  - istio-locality: "region/zone/subzone “
- WorkloadEntry 的 Locality 属性
  - locality: region/zone/subzone

- 然后就可以利用地域信息进行流量转发了

Distribute

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: foo
spec:
  host: foo.com
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        distribute:
        - from: "*/*"
          to:
            region1/zone1/*: 99
            region2/zone2/*: 1
        enabled: true
    outlierDetection:
      baseEjectionTime: 10s
      consecutive5xxErrors: 100
      interval: 10s

  tls:
      mode: ISTIO_MUTUAL
```

Failover

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: foo
spec:
  host: foo.com
  trafficPolicy:
    loadBalancer:
      localityLbSetting:
        enabled: true
        failover:
        - from: region1/zone1

          to: region2/zone2
    outlierDetection:
      baseEjectionTime: 10m
      consecutive5xxErrors: 1
      interval: 2s
    tls:
      mode: ISTIO_MUTUAL
```

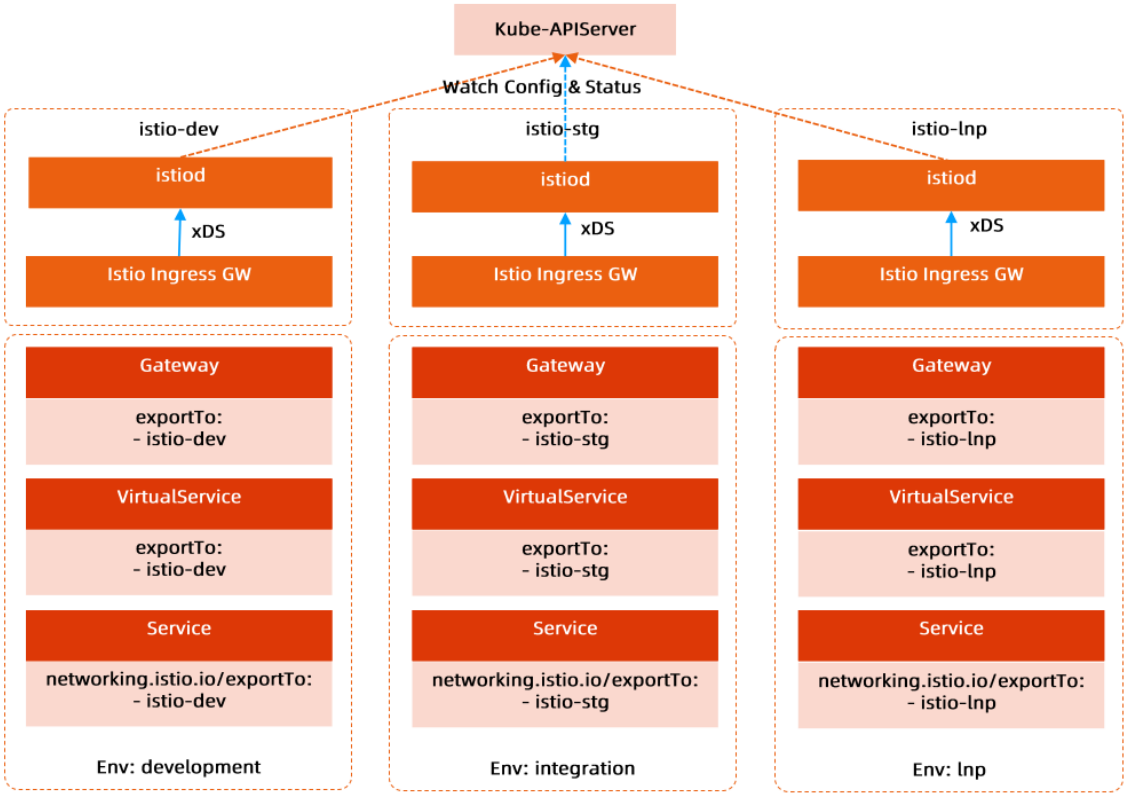
应对规模化集群挑战

通过Istio对象中的exportTO属性覆盖默认配置，只将服务发布到指定的空间中

```
defaultServiceExportTo:
- "."
defaultVirtualServiceExportTo:
- "."
defaultDestinationRuleExportTo:
- "."
```

Istio自身的规模控制

社区增加了discoverySelector的支持，允许Istiod只发现添加了特定label的namespaces下的Istio以及kubernetes对象，这样就可以实现单网关集群多环境支持



> 但因为kubernetes框架的限制，该功能依然要求Istiod接收所有配置和状态变更明细，并且在Istiod中进行对象过滤，在超大集群中，并未降低网络带宽占用和Istiod的处理压力。