

随着容器等轻量级高效率虚拟化技术的兴起与微服务理念普及，云计算正向着“云原生”（Cloud Native）的方向发展。为了适应这个趋势，网络也需要进行相应的改造以更好地支撑云原生平台大规模的弹性能力与服务自愈和的特性。开放网络的技术将在云原生场景中得以广泛应用。



本文根据周雍恺博士在NCDC大会上的演讲报告《云原生视角下的开放网络》整理而成。

## 何谓“云原生”

从2015年开始，微服务、CI/CD、DevOps、Serverless、SRE等词汇大量涌现，一场云原生的运动正式拉开大幕。

云原生，从广义上来说，是更好地构建云平台与云应用的一整套新型的设计理念与方法论，而狭义上讲则是以docker容器和Kubernetes（K8S）为支撑的CNCF技术生态堆栈正在革新整个IT架构。我关注到本次大会还设了一个OpenStack开发者论坛，但是其中所有演讲议题都以容器或K8S为主，由此也可以印证，云原生的潮流势不可挡。

以Docker为代表的轻量级容器虚拟化技术，将成为今后企业应用发布的标准形态，横跨众多Linux甚至是Windows平台。



## 01 云原生 | 以容器为代表的轻量级虚拟化技术



以Kubernetes (K8S) 为代表的云原生编排系统, 将成为分布式集群架构的核心操作系统。



## 01 云原生 | 以K8S为代表的云原生编排技术



### Kubernetes

- Google多年大规模生产运维基础设施服务Borg的开源版本
- 架构清晰, 功能完备
- 高度弹性伸缩, 秒级扩容数百个容器实例
- 管理超过100万台服务器, 跨越80个数据中心
- 每周运行20亿个容器

- 调度平台
- 应用弹性
- 监控管理
- 标准化工具
- 持续集成工具
- 快速部署平台
- 灰度升级

标准组件与服务

统一调度与弹性扩展

快速开发与部署

### 适用业务:

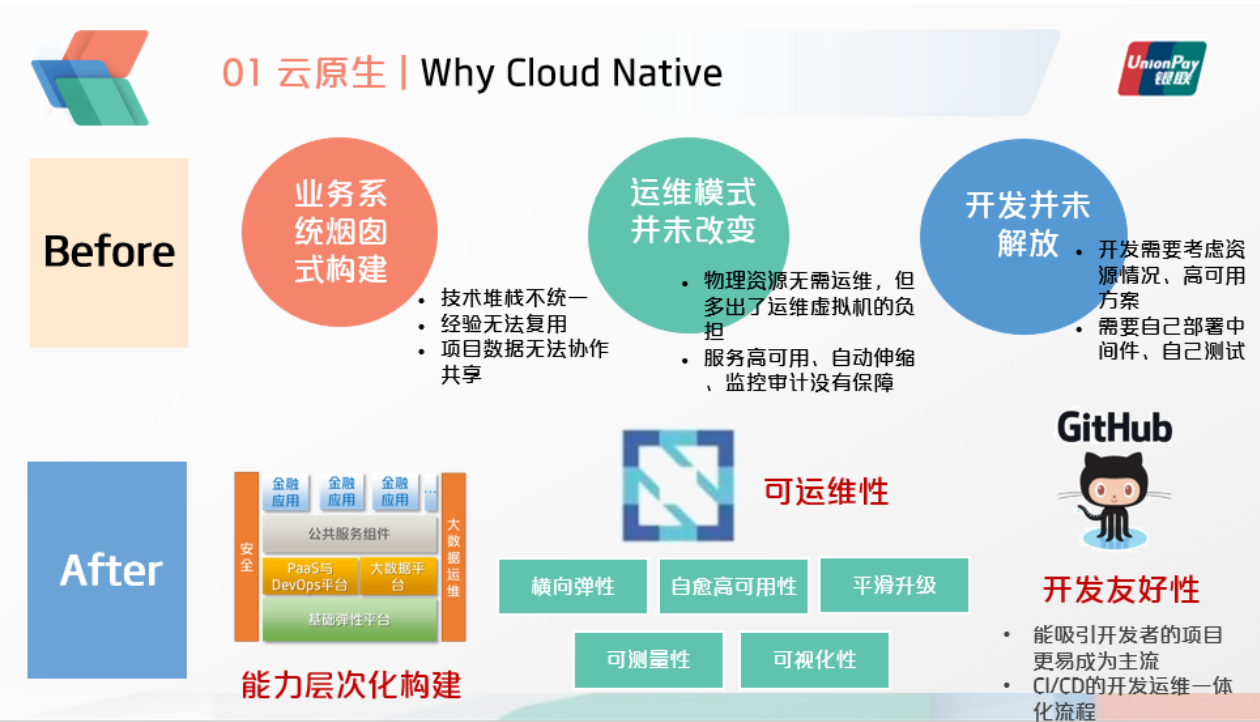
- 新建系统
- web服务系统
- 创新类业务
- 具有快速发布, 持续迭代等要求的业务



基于K8S/OpenShift之上构建现代化的IT开发与治理流程

当时硅谷的大佬们抛出云原生的提法, 说明之前用云的方式存在很大问题, 并不是原生的, 主要体现为: 1) 业务系统烟囱式的构建, 项目经验无法沉淀复用, 项目数据无法协作共享, IT治理难度较大 2) 运维模式未有本质性的改变, 虚拟化的出现使得物理资源无需运维, 但是多出了运维虚拟机的负担, 像服务高可用、自动伸缩、监控审计, 这些在平台方面是没有保障的, 还是需要人工介入 3) 开发并未得到解放, 开发人员在写程序时仍然需要考虑资源的使用情况, 高可用的方案, 还需要自己部署中间件, 自己进行测试。

而随着云原生的出现，它理想的场景，一是IT的能力可以最大化的复用，能力层次化地构建，体现IT治理的成效。其次是最重要的可运维性，它包括云原生所强调的大规模横向弹性，自愈高可用性，平滑升级，可测量、容器化等特性。最后是开发友好性，越来越多的项目表明，能够吸引开发者的项目更容易成为主流，此外，CICD开发运维一体化的流程可以大大节省集成部署的时间与难度，提升生产效率。



再从数据中心架构演进的视角来看，最早的云计算是IaaS资源池化，以Vmware与OpenStack为代表的IaaS编排系统，对网络计算以及存储及资源进行统一的管理。这一阶段主要解放的是资源的运维，但是对于应用的构建、部署的方式，弹性伸缩与高可用保障并无本质性的改变。



第二阶段是PaaS的服务化，强调的是服务的能力，需要具备弹性与高可用的保障，而不是简单的提供IT资源服务，云原生的应用能够更好的利用云平台所提供的能力。云原生是整个IT架构与组织架构的真正向分布式云化的重构，其终极的状态是Serverless，应用彻底无需关心底层的资源运维。

第三阶段是更加现代化的数据智能阶段，它强调的是数据的运营能力以及内外部数据整合对外提供服务的能力，其中数据治理与隐私数据保护是数据运营的重要保障，系统架构设计则是以数据流为核心的，也是真正将企业的核心数据转化为生产力与生产价值的重要阶段。

当前的系统架构主要聚焦于第二阶段，第三阶段的大数据智能也在逐步开展，当然也是要以PaaS服务化作为基础的。作为PaaS服务化的核心，云原生目标就是能力堆栈可以复用，业务可以在有保障的云平台中快速上线。

## 云原生网络

第二大部分周博士介绍了云原生网络。由于云原生的内涵非常丰富，要理清其中的脉络，绝非易事。在此，主要从两个角度来对云原生网络进行说明：一是网络如何为云原生应用提供支撑，二是如何用云原生的设计理念来改进网络。第一部分称之为**云原生组网**，第二部分称之为**网络的云原生化**。

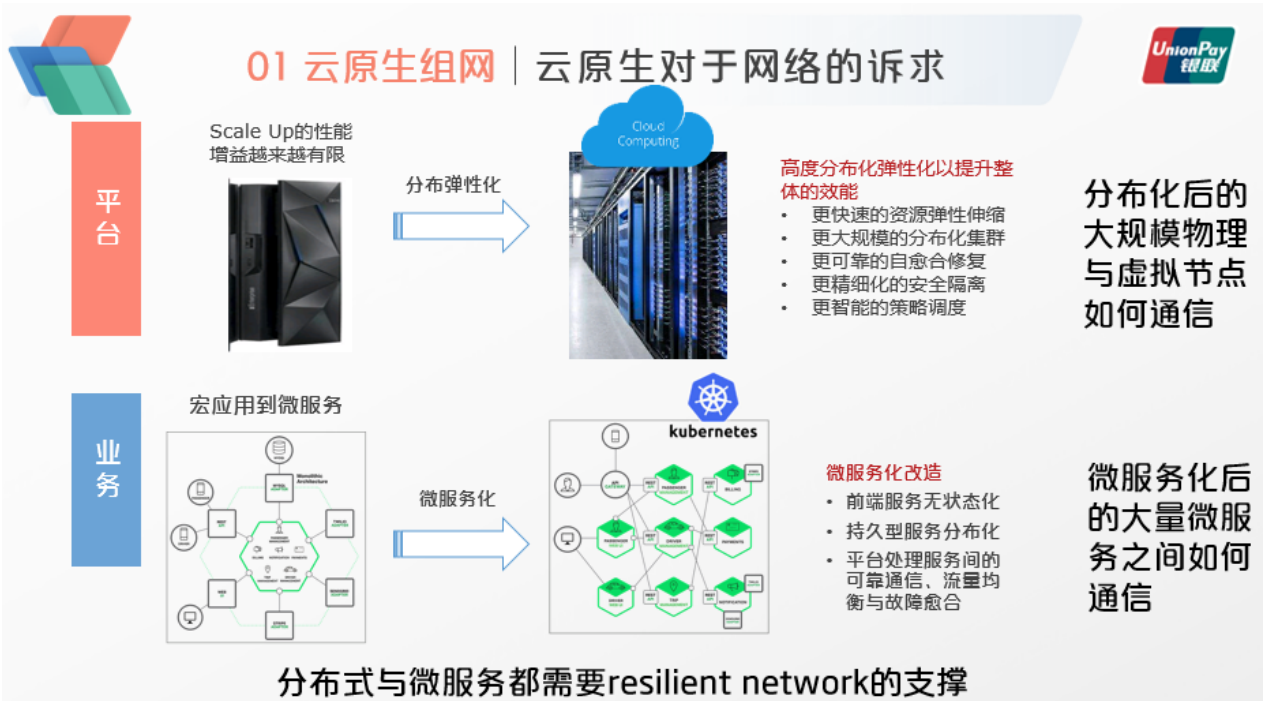
### 云原生组网

云原生对于网络的诉求，可以从平台以及业务两个方面来进行分析：

- 从平台方面而言，**scale up**的性能增益越来越有限，整个平台的体系向分布式弹性化的方向进行发展，在性能与可靠性上都可以有较高的保障。
- 而对于业务而言，整个的趋势则是从原来的单体应用向微服务化进行转变，从而能够更好的利用到云平台弹性可扩展的优势。前端的服务尽量无状态化，有状态的部分存储到后端的分布式存储之中，而至于服务间的可靠通信、流量均衡与故障自愈合等问题则交由平台来负责处理。

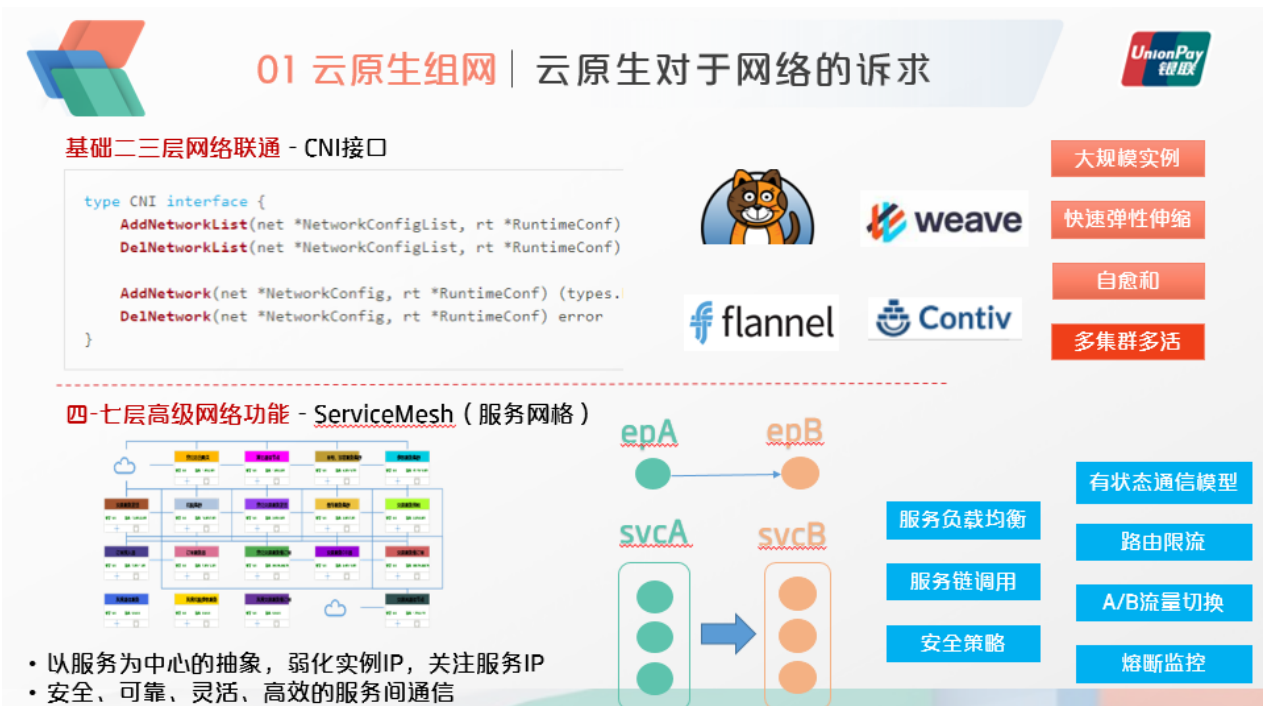
无论是平台的分布化以及业务的微服务化，都需要一个强大的网络来支撑大规模虚拟节点以及微服务之间的通信。






具体到技术层面，云原生对于网络的要求，一是基础的二三层网络联通，二是4~7层的高级网络功能。

二三层的网络主要实现K8S中CNI接口，CNI接口的定义相当简单，基本上就是创建一张二层的虚拟隔离网络，相当于IaaS中vpc网络的概念。具体的实现方案有calico, flannel, weave, contiv等，这些方案的介绍网上资料有很多。与以前虚拟机时代的vPC不同，云原生场景下，对于二三层网络的创建有很高的性能要求，每秒要创建上千个网络的end point，并且能够快速弹性伸缩，还要具备自愈合的能力，此外为了将服务部署到多个集群中，网络还要需要提供跨K8S集群多活组网的能力。

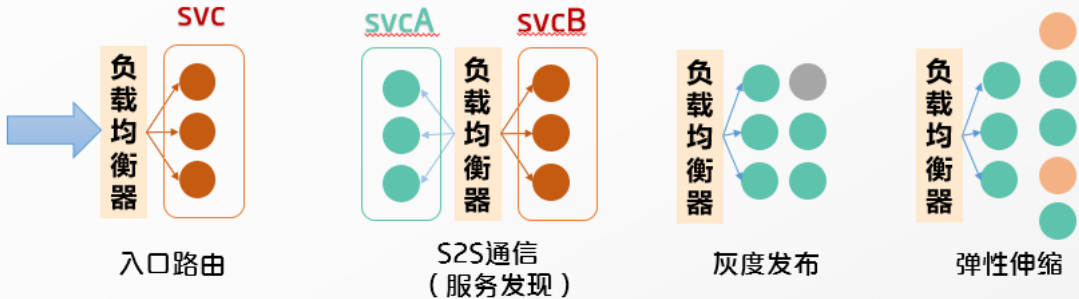


4~7层的组网，则是ServiceMesh，中文翻译为服务网格。顾名思义，ServiceMesh里的通信，不再是点到点的通信，而是抽象成服务到服务之间的通信，每个服务由众多的计算实例所组成。这种以服务为中心的抽象，将弱化每个实例的IP而更加关注的是服务的虚拟IP，ServiceMesh的本质是提供安全、可靠、灵活、高效的服务间通信。有文章将ServiceMesh称之为下一代的SDN，有一定的道理，但是它也离不开基础二三层网络的联通。ServiceMesh将会提供一些更加高级的网络功能，例如有状态的通信，路由限流，灰度流量切换，熔断监控等等，是整个微服务框架的基础支撑。

对于上述的这些功能，可以分解开来看：首先是负载均衡。负载均衡在K8S中的作用，无论如何高估都不过分，整个分布式架构的数据平面基本上都是靠负载均衡来进行维系的，比如入口的路由，就是一个7层的负载均衡器；而服务间的通信或者说服务发现，也是依靠负载均衡来让流量分发到另外一个服务中的具体实例；此外灰度发布的话，也是由负载均衡在新旧版本之间做切换，而弹性伸缩其实本质上就是一个负载均衡，将流量按比例逐步切换到新增的计算实例之中。




## 2.1 云原生组网 | 负载均衡的作用



### K8S (iptables/IPVS)

```
KUBE-SEP-5YXRASJHTFWMUXIW all -- * statistic mode
random probability 0.33332999982
KUBE-SEP-BVNV3X6U34LM7T7M all statistic mode
random probability 0.50000000000
KUBE-SEP-BJX4QFI64J2WHZVP all -- * * 0.0.0.0/0
0.0.0.0/0 /* tidb1/tidb:mysql */
```

### Istio (envoy sidecar)



- SideCar “劫持” 服务之间的通信流量
- Envoy的性能是关键

在K8S中，服务间的负载均衡实现是通过iptables中对于每条实例规则的概率匹配来实现的。而在服务网格Istio中的实现，则是靠一个sidecar边车容器来“劫持”两个服务容器之间的流量，流量一旦被“劫持”后，那就可以做负载均衡、熔断限流、灰度发布、安全检测、遥测上报等一系列的操作了。当然，在这种方案中，sidecar（envoy容器）的转发性能是关键所在。

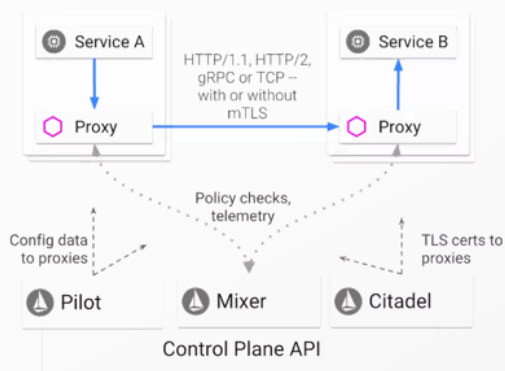
之前说到了Istio，它是当前CNCF主推的微服务框架，下面详细介绍一下它的实现架构。



## 2.1 云原生组网 | Istio服务网络



### Istio微服务治理架构



- 数据平面Sidecar无代码侵入
- 平台级的微服务框架
- Pilot下发控制，Mixer收集运行状态

### 丰富的后台工具形成完整闭环



#### Kiali

- 服务拓扑图
- 指标度量收集
- 配置校验
- 健康检查和显示
- 服务发现

#### Prometheus + Grafana

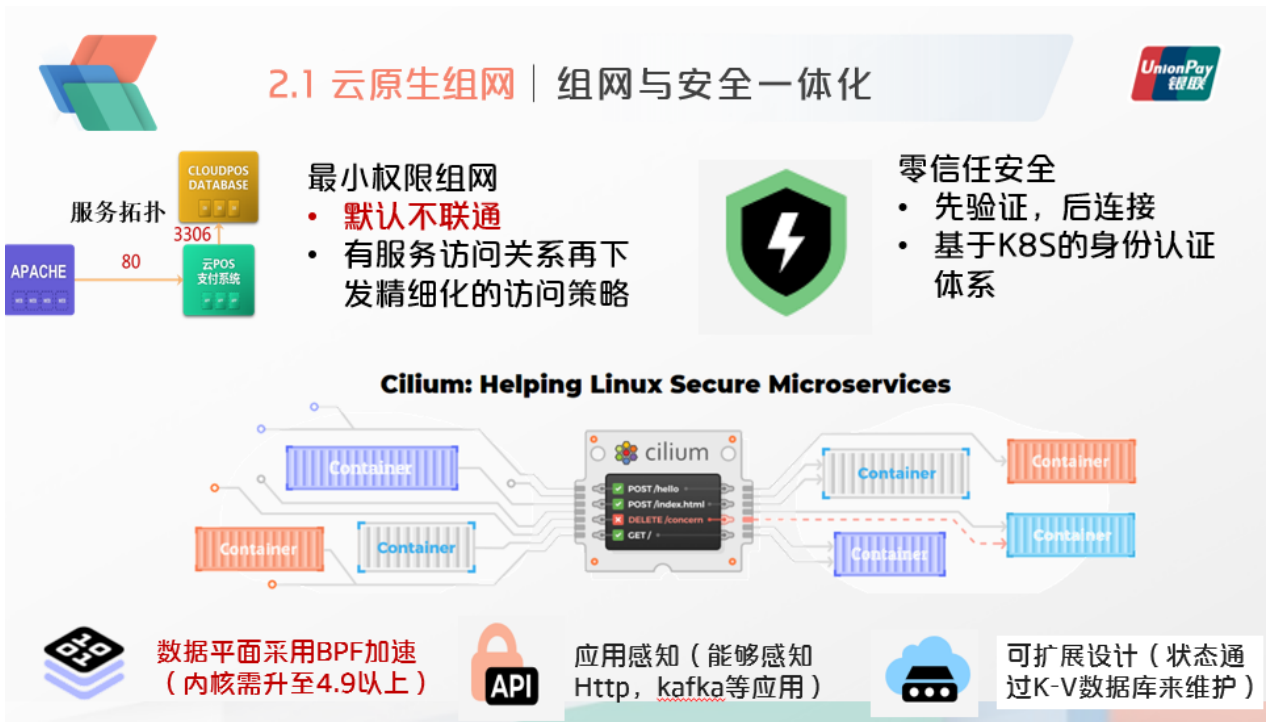
- 指标监控
- 服务监控
- 可视化呈现

在数据平面，它通过sidecar对于流量进行劫持，从而实现无代码侵入的服务网络。这是它的一大实现特点，相比于spring cloud，dubbo等框架，它可以被称作是平台级的微服务框架，也就是说其中微服务通信的所有功能都是靠平台来进行实现，而无需通过增加应用代码，或者Java应用框架进行支撑，这更加符合云原生的设计理念。

控制平面的两个组件中，pilot负责下发控制，mixer收集运行的状态，citadel则负责安全证书方面的处理。

此外，在另一个维度。Istio还有非常丰富的后台工具，能够对其运行进行闭环支撑。首先是Prometheus与Grafana，对于Istio的各项指标进行监控，并做可视化的呈现。其次是Jaeger和Kiali这样的分布式追踪工具，他们可以显示出微服务之间的调用拓扑，以及每个API完整调用中的子调用，相应的时长等，这对于发现微服务通信之间的网络异常以及性能瓶颈，有非常重要的作用。

最后再来看一下云原生组网的安全，这里强调的是组网与安全是一体化的。



具体而言，一是最小权限组网，相对于以往的默认互通，在服务化场景中，最优化的实现是默认不连通，只有当服务之间有调用关系时，才下发访问策略，从而实现精细化的网络访问控制。

其次是零信任安全，它强调的是，每次调用前先验证后连接，对于组网而言，可以基于K8S的身份认证体系进行认证。

这里再看一个比较有意思的实现案例，叫做Cilium，它也是CNCF CNI中的一个项目，但它主打的却是安全牌，首先在数据平面，采用了BPF机制（Berkeley Packet Filter，Cilium的整个开发团队就是BPF的原班人马），其次是对于应用的感知能力，基于BPF框架，Cilium能够以 service / pod / container 为对象进行动态地网络和安全策略管理，解耦控制面的策略管理和不断变化的网络环境，还做到 7 层能力，感知常见的 HTTP，Kafka等应用流量，从而实现对于流量的精确化控制，三是可扩展设计，它的状态通过etcd来进行维护，复用K8S的一些能力。

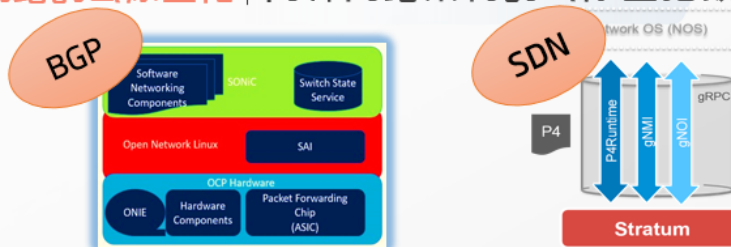
## 网络的云原生化

周博士主要以BGP与SDN两种典型的网络架构为例，分析了他们的云原生指数。其中BGP的实现以SONiC为例，SDN的实现以Stratum+ONOS为例，这两者都是开源的交换机操作系统。





## 2.2 网络的云原生化 | 两种网络架构的云原生指数



	SONiC	Stratum + ONOS
横向弹性可扩展	☹	☹
自愈高可用	☹	☹
配置友好性	☹	☹
可测量性	☹	☺
闭环可控性	☹	☺
容器化	☺	☹
开发友好性	☹	☹

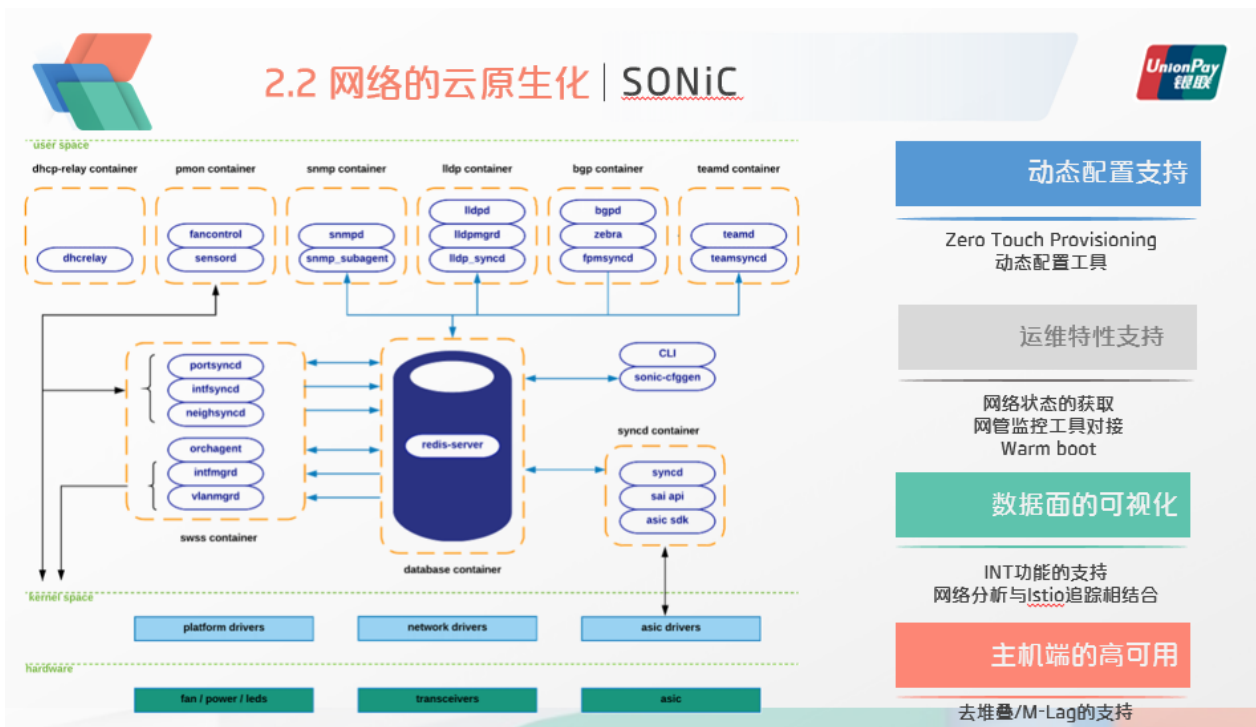
首先是云原生最重要的两个指标：横向的弹性可扩展，以及自愈和高可用，这一点上BGP分布式路由具有天然的优势，这在全球Internet大规模的场景中已经久经考验。而对于SDN式的集中式控制而言，控制平面的分布式可扩展以及高可用，需要花大量的笔墨进行设计。此外，一旦数据平面与控制平面的链路出现故障，是否能够快速恢复也是一大考虑因素。

在另外一方面，配置友好性、可测量性、闭环可控性，这些指标BGP并不占优势，经常出现BGP人为配置错误的报道，分布式网络的故障排查尤其困难，而这又反过来是SDN网络控制的一些优势。

在容器化支持方面，SONiC操作系统采用了先进的容器化、模块化的设计；而ONOS是基于Java OSGi的框架，尚无容器化的计划。最后在开发友好性方面，两者目前的状态都只能评价为一般。

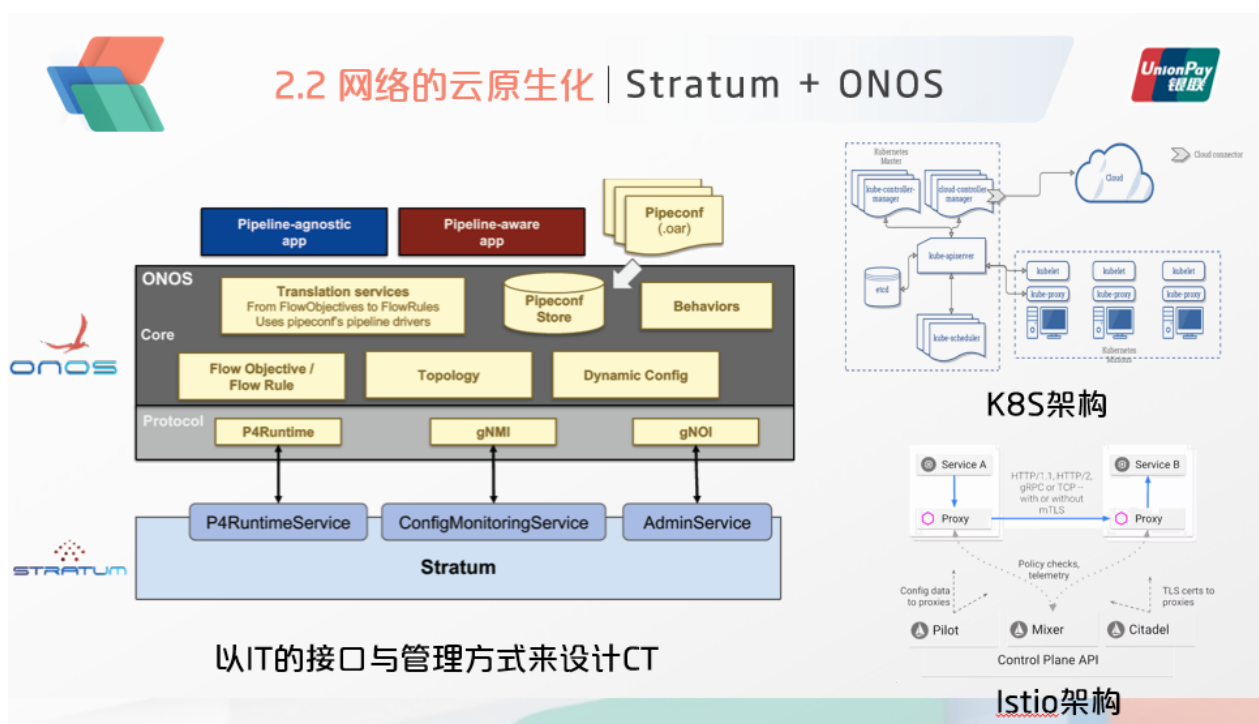
接下来周博士分享了这两个系统未来有可能的改进方向。

正如之前所说的，SONiC操作系统的设计采用了容器化、模块化的理念。其中心是一个用于存储状态的redis数据库，其他的功能组件都是围绕他而进行构建交互，其中有BGP路由的容器，以及SAI容器，负责与内核层硬件转发芯片的驱动打交道。



系统改进方面，一是需要有更多动态配置的支持，包括ZTP，动态配置工具等；二是非常重要的运维特性的支持，包括网络状态的获取，网管、监控工具对接，热启动等；三是数据平面的可视化，尤其是INT功能的支持，当前所有正式上线的SONiC集群，都会把INT功能作为其中的必选项加入，从而可以更好的跟踪网络的流向。当然仅仅从网络层面进行分析还不够，还必须要与上层业务系统的分布式追踪相结合才能达到更好的效果；最后是主机端的高可用，SONiC当前不支持主机双上联，这对于生产应用来说几乎很难接受，因此还需要加入mlog或者更为简洁的去堆叠功能。

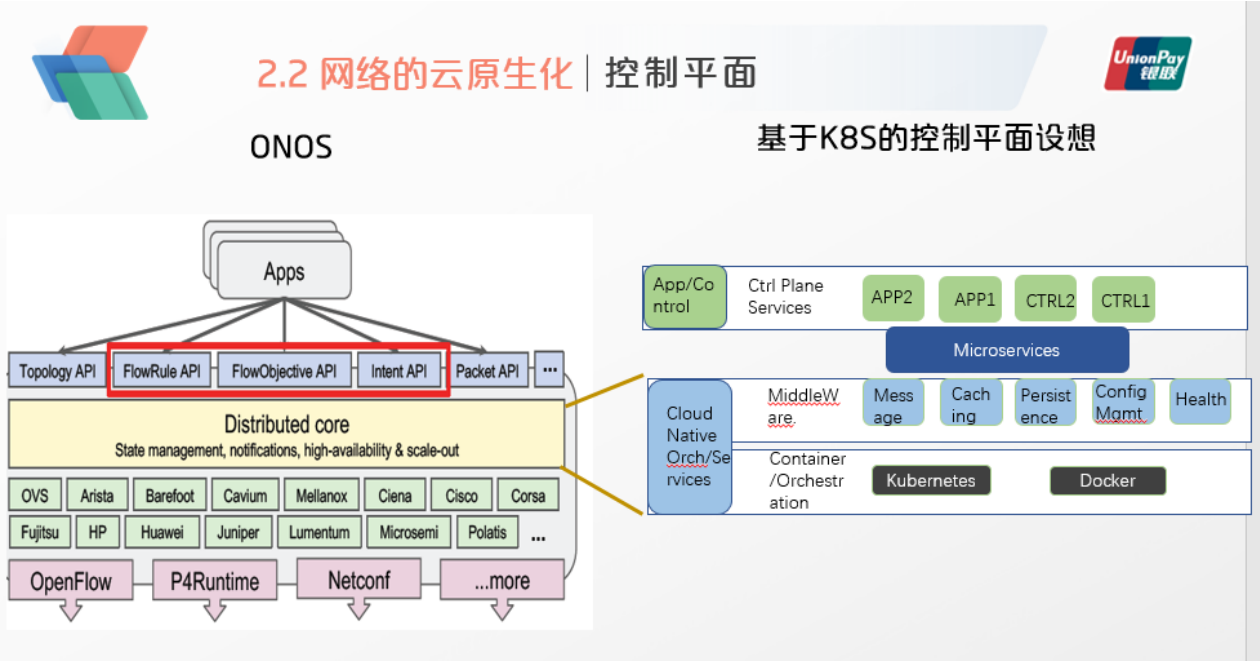
再看一下Stratum+ONOS架构的组网设计。



Stratum由谷歌与ONF进行开源运作，他整体的设计方案，贯穿了谷歌一贯的Controller-Worker分布式系统的设计模式。具体而言，P4Runtime负责下发控制，gNMI与gNOI负责网络管理与运行。三种接口都通过以gRPC Protobuf进行封装，从而可以提升接口调用的效率。

我们如果把Stratum+ONOS的架构，与K8S和Istio比较一下，可以发现是相当类似的。在K8S中，Master通过Kubelet向工作节点下发指令，同时节点状态也会上报存储到etcd数据库中，KubeProxy负责数据平面。

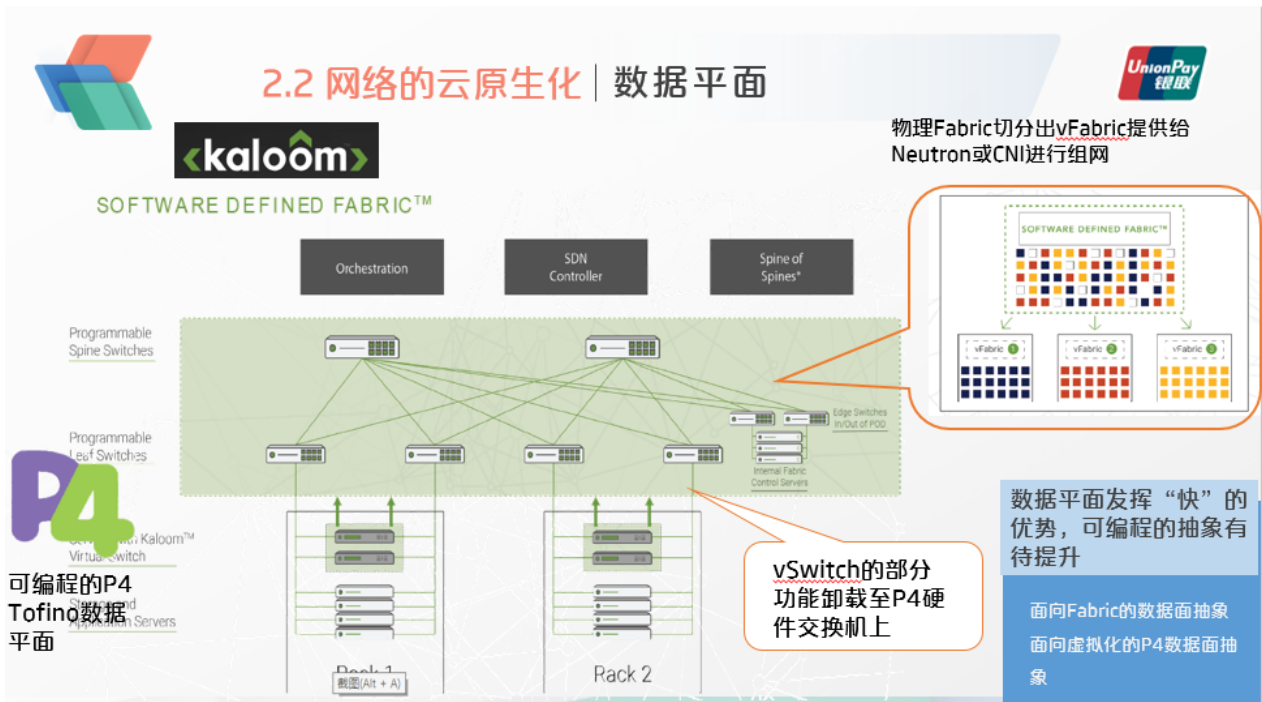
而Istio的架构中，Pilot负责发控制指令，Mixer负责收集运行的数据。Stratum与他们整体的管控方式都非常类似，也可以称之为是以IT的接口与管理方式来设计CT系统。



在控制平面，为了适应云原生化的架构，还是会有进一步的加固改进空间，在ONOS的实现框架中，它最重要的就是一个分布式核心（左图黄色部分），提供了整个集群的状态管理、消息通知、高可用以及弹性伸缩。

而这部分在云原生的体系中，正是K8S所处理的强项，当然还要配合以消息总线、缓存处理、持久化、配置管理、健康检测等辅助性的中间件功能，在之上则可以通过容器来部署各种各样的网络功能。这是采用K8S改造ONOS控制平面的一个设想，由此可以复用到K8S很多分布式核心平台的功能，保持CT与IT架构的统一，同时也可以降低ONOS整体的复杂性。

另外在数据平面也需要做一些虚拟化的可编程支持。

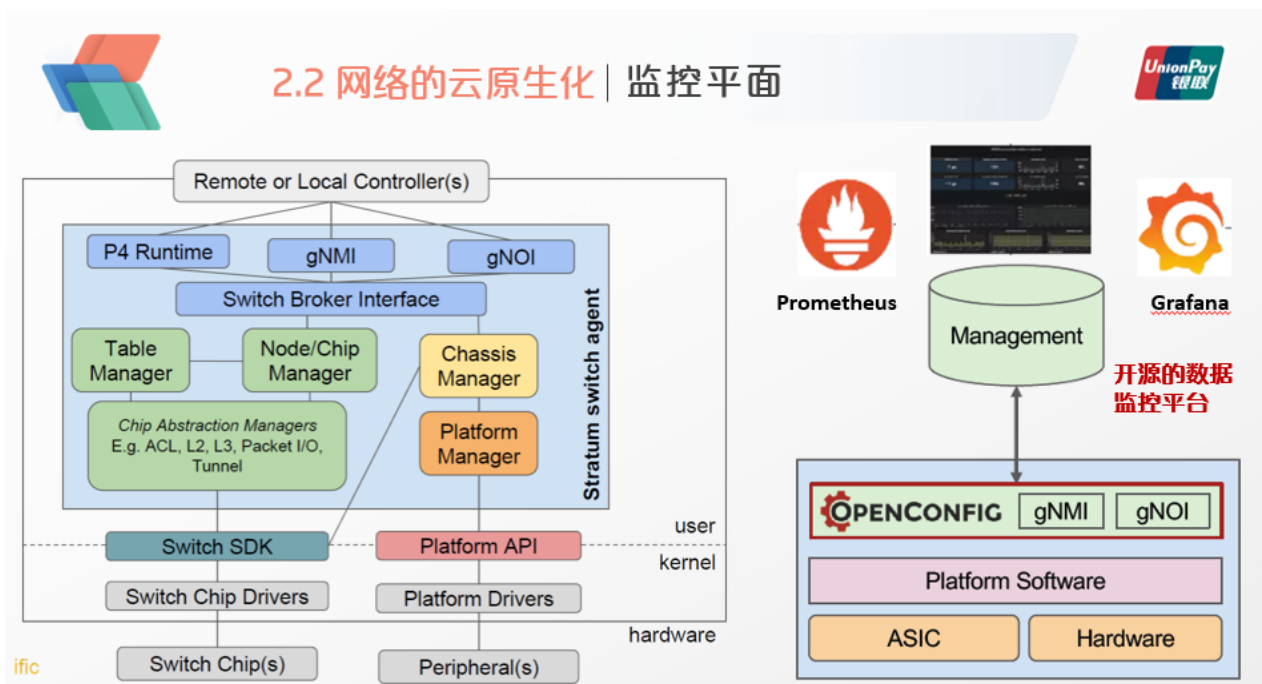


以硅谷的初创公司Kaloom为例，它主打的是软件定义fabric，数据平面由P4语言，以及可编程的交换机进行支撑。

整个物理的fabric通过网络控制，可以切分出多个虚拟的vFabric，提供给Neutron或者CNI进行vPC的组网。

因此数据平面在发挥速度快的优势的同时，可编程的抽象程度还有待提升，主要体现为面向fabric的数据平面抽象以及面向虚拟化的P4抽象，这些都需要进一步的研究努力。

另外还有构成闭环的监控平面，Stratum提供了gNMI与gNOI接口，可以对接外部的网管平台。



相对于传统的网管平台，我们最期望的仍然是采用开源的数据监控实现方案，当前Prometheus与Grafana基本已经是云原生系统的标准监控工具了，因此对于网络的监控也可以利用两个组件来进行对接，从而提升整个平台的可控与可运维能力。

最后再回到“当云原生遇上开放网络”的主题。周博士指出无论是之前所提到的SONiC，Stratum，ONOS，还是P4，可以说开放网络在整个云原生网络体系中扮演着重要的作用。



上图是参考自思科云原生网络功能白皮书所修改的，最底层是容器引擎与容器编排平台，数据平面有P4，OpenvSwitch，DPDK，VPP，SAI等优秀的开源工具，控制平面,有SONiC与Stratum之类的交换机操作系统，以及ONOS与ODL控制器。在此之上，我们都可以构建各种各样微服务形态的网络功能，例如负载均衡、网络遥测、网关等，此外通过整个CICD的编排以及DevOps的运维，让整个网络体系形成自动化自愈闭环的结构。

网络堆栈的开放为实现云原生网络的大规模系统工程提供了支撑，很难想象光靠封闭的盒子和大框可以堆出一个云原生的网络。

最后是几点总结：

- 云原生的场景下，网络是分布式与微服务化的重要支撑，网络的功能本质上是为应用提供服务的，因此应当是需求驱动的，网络的功能将会朝着更好地贴合云原生应用需求这个方向发展。
- 采用云原生的理念改造网络的设计，尤其是控制平面的设计，可以利用当前云原生IT体系建设的经验与成果，提升网络的可控性与高可用弹性能力。随着网工IT能力的提升以及ICT整合的加速，今后网络的设计与控制也会朝着云原生化进行过度。
- 网络的软件定义化以及网络堆栈的开源开放，已经发展到了一定的程度。在数据平面，随着OpenvSwitch与P4的出现，释放软件能力进入网络领域成为可能，整个转发处理流水线都可以进行软件化的定义，ONF，Linux Foundation以及CNCF等开源组织的一系列项目，为打通整个网络堆栈做了一系列重要的工作，这些都将有助于云原生网络目标的最终实现，而且未来开放网络会在云原生网络的体系中发挥



更大的作用！

由于时间有限，本报告所讨论的范围主要集中在数据中心内部的组网，其它诸如广域网、拥塞控制、AIOps等话题未有展开。

来源：<https://www.sdnlab.com/23359.html>