阿里巴巴云原生化应用管理实践

孙健波 阿里云技术专家

孙健波

阿里云智能事业部 技术专家

OAM项目 Maintainer
Kubernetes 项目社区成员

jianbo.sjb@alibaba-inc.com

目录

阿里巴巴云原生化路径

2011: 容器化

- LXC
- BU维度的资源池
- scripts

2015: docker容器化 + 智能调度

- Docker
- 内部自研调度系统

2017-18: k8s 基础编排

- “富容器”
- 一个Pod 一个container
- K8s API

2019: 云原生

- Containerd
- Pod + sidecars
- K8s 全栈

容器化
(围绕容器本身)

探索
(围绕K8s API)

云原生
(标准 + 开放)

我们遇到的新挑战

1. 应用描述、应用管理与底层运行模式耦合
   - 绑定了Deployment的产品（Flagger、Argo rollout ），无法扩展
   - 开发、运维分工不明确，开发无法理解K8s的yaml
2. 复杂应用交付困难
   - Operator可以自动化运维，但不包含完整的应用描述
   - 带状态的应用交付需要大量的运维能力支撑（节点亲和性、服务发现、实例唯一ID...）
3. 云上资源、运维能力众多，难以管理
   - 没有面向应用本身的编排：有面向资源的生命周期管理（Alibaba ROS、Terraform、AWS Cloud formation ），缺失应用生命周期管理，如升级、监控、报警..
   - 运维能力的管理缺失：编排、冲突管理
4. 云上、云下不统一
   - 同一种软件在云上云下API不同

案例 1：为什么某内部 PaaS，只允许研发设置 Deployment 的个别字段?

- 研发是否能确定实例数?
  - HPA 怎么接入?
  - Replica=几?
- allowPrivilegeEscalation 的意思是?
- 彻底把研发和运维的字段分开设置，能不能解决问题?

  Separate concerns 很重要

  但研发的诉求，也必须能传递给运维

  只有研发，才最懂应用!

```yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      deploy: example
  template:
    metadata:
      labels:
        deploy: example
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          securityContext:
            allowPrivilegeEscalation: false
```

案例 2：我们上线了 CronHPA

但是：

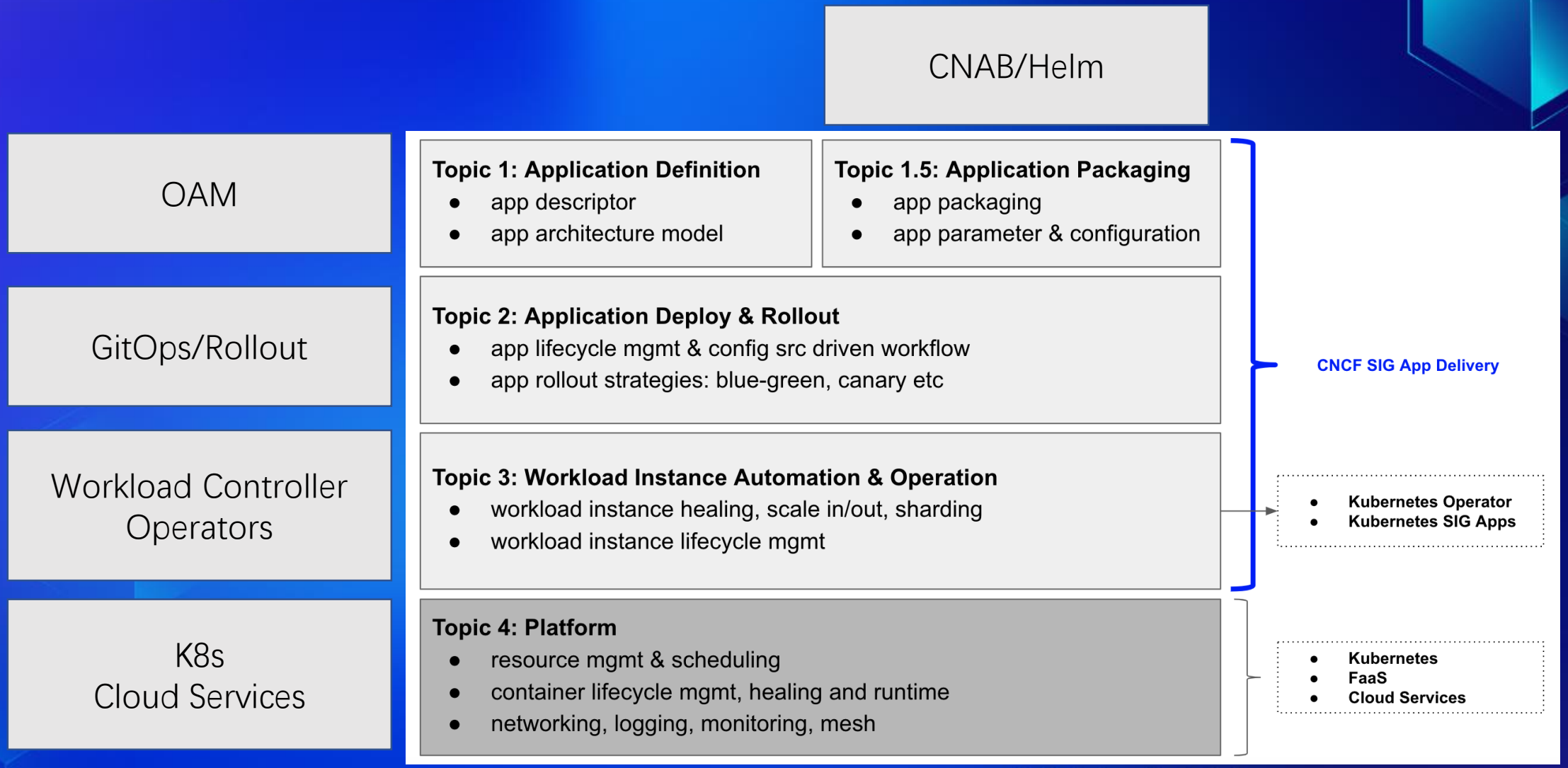
1. 运维如何确认这个 CronHPA 插件在某个集群安装了？
2. 运维如何知道这个 CronHPA 怎么用？
3. 运维如何确定这个 CronHPA 跟默认 HPA 会不会发生冲突？

K8s 中，大量的运维能力通过插件提供，但是 ...

真实故障：某运维将插件 A 和插件 B 绑定给同一个 Deployment，但却不知道这两个插件的能力是存在冲突的，应用大量失败。

```yaml
apiVersion: "app.alibaba.com/v1"
kind: CronHPA
metadata:
  name: cron-scaler
spec:
  timezone: America/Los_Angeles
  schedule:
  - cron: '0 0 6 * * ?'
    minReplicas: 20
    maxReplicas: 25
  - cron: '0 0 19 * * ?'
    minReplicas: 1
    maxReplicas: 9
  template:
    spec:
      scaleTargetRef:
        apiVersion: apps/v1
        kind: Deployment
        name: php-apache
      metrics:
      - type: Resource
        resource:
          name: cpu
          target:
            type: Utilization
            averageUtilization: 50
```

阿里巴巴正在推进云原生应用管理体系



为什么需要应用标准定义？

- K8s 项目没有“应用”的概念
  - 使用 YAML 文件（K8s API 资源）来描述用户的提交
- 各云提供商、云产品，也没有统一的“应用定义”的概念
- 但“应用定义”是：
  - 终端用户的主要关心点
  - 基于云搭建 Serverless/PaaS 的强需求与关键依赖
  - 应用分发的起点

- 社区现有的尝试
  - Docker Image:
    - 打包软件制品，比如： WAR 包 → 应用可能是多个执行文件的组合，比如：Pod
  - Helm/K8s Application Def:
    - 多个 K8s API 资源的组合 → 无法定义应用外部依赖，比如：SLB，VPC，RDS
无法定义运维动作，比如：应用的水平扩展策略
无法定义应用的运行时，比如：K8s，ECS，FaaS
  - CNAB:
    - 多个 Docker Image的组合 + 启动脚本 + 启动参数 → 黑盒，不利于协作；必须双方信任；没有规范性

OAM 开放云原生应用模型

- 云原生：描述和规范天然“生于云上，长于云上”的应用定义
  - 声明式
  - 自包含
  - 分层、松耦合
  - 弹性、可扩展
- 解决传统应用描述三个核心问题：
  - 问题 1：运行时锁定
    - 只能被 Docker/K8s/某云平台/某 PaaS 使用
  - 问题 2：不区分使用者角色
    - 举例：开发在描述应用时，希望去定义 K8s Volume 的细节吗?
  - 问题 3：所有概念耦合在一起（All-in-one），不能协作，不能复用
    - 一个配置文件 = 容器镜像 + 运行参数 + Ingress + 网络配置 + 存储配置 + 虚拟机配置 ...

OAM Component

1. **描述应用本身**
2. **描述应用如何运行的可扩展参数**
3. **描述可被覆盖的参数** (schemas)

```
apiVersion: core.oam.dev/v1alpha1
kind: Component
metadata:
  name: nginx
  annotations:
    version: v1.0.0
    description: >
      Sample component schematic that describes the
administrative interface for our nginx deployment.
spec:
  workloadType: Server
  osType: linux
  containers:
  - name: nginx
    image:
      name: nginx:1.7.9
      digest: <sha256:...>
  workloadSettings:
  - name: initReplicas
    value: 3
  - name: worker_connections
    fromParam: connections
  parameters:
  - name: connections
    description: "The setting for worker connections"
    type: number
    default: 1024
    required: false
```

OAM Component

关注点分离，但不等于完全割裂！

研发如何通过 *Component* 向运维表达诉求？

开发给运维的操作提示

可覆盖的字段列表

引用一个可被运维覆盖的参数

```
apiVersion: core.oam.dev/v1alpha1
kind: Component
metadata:
  name: nginx
  annotations:
    version: v1.0.0
    description: >
      Sample component schematic that describes the
administrative interface for our nginx deployment.
spec:
  workloadType: Server
  osType: linux
  containers:
  - name: nginx
    image:
      name: nginx:1.7.9
      digest: <sha256:...>
  workloadSettings:
  - name: initReplicas
    value: 3
  - name: worker_connections
    fromParam: connections
  parameters:
  - name: connections
    description: "The setting for worker connections"
    type: number
    default: 1024
    required: false
```

插件式的运维能力管理三大难题

- 正交：如Ingress管流量，Storage Class管存储。
- 编排：不同的运维能力可以用于同一个应用，如基于流量的灰度。
- 冲突：不同的运维能力有时候是互斥的，如HPA vs Cron HPA。

- 运维人员管理运维能力几乎很难提前预知风险

可发现、可管理的运维能力：OAM Traits System

```
1 kubectl get traits
2 NAME                AGE
3 cron-scaler         19m
4 auto-scaler         19m
```

发现运维能力

```
$ oamctl trait-list
```

NAME	VERSION	PRIMITIVES
autoscaler	0.1.0	Server, Worker
ingress	0.1.0	SingletonServer,

kubectl get traits crob-scaler –o yaml

```yaml
1  apiVersion: core.oam.dev/v1alpha1
2  kind: Trait
3  metadata:
4    name: cron-scaler
5  spec:
6    appliesTo:
7      - core.oam.dev/v1alpha1.Server
8    properties:
9      - name: timezone
10       description: "Time zone for this cron t
11       type: string
12       required: false
     - name: schedule
       description: "CRON expression for a sca
       type: string
       required: true
     - name: cpu
       description: "The CPU consumption thres
       type: int
       required: false
       ...
```

查看能力用法

kubectl apply -f example.yaml

```yaml
1  apiVersion: core.oam.dev/v1alpha1
2  kind: ApplicationConfiguration
3  metadata:
4    name: failed-example
5  spec:
6    components:
7      - name: nginx-replicated-v1
8        instanceName: example-app
9        traits:
10         - name: auto-scaler
11           properties:
12             minimum: 1
13             maximum: 9
14         - name: cron-scaler
15           properties:
16             timezone: "America/Los
17             schedule: "0 0 6 * * ?"
18             cpu: 50
```

提前暴露冲突

绑定能力给应用

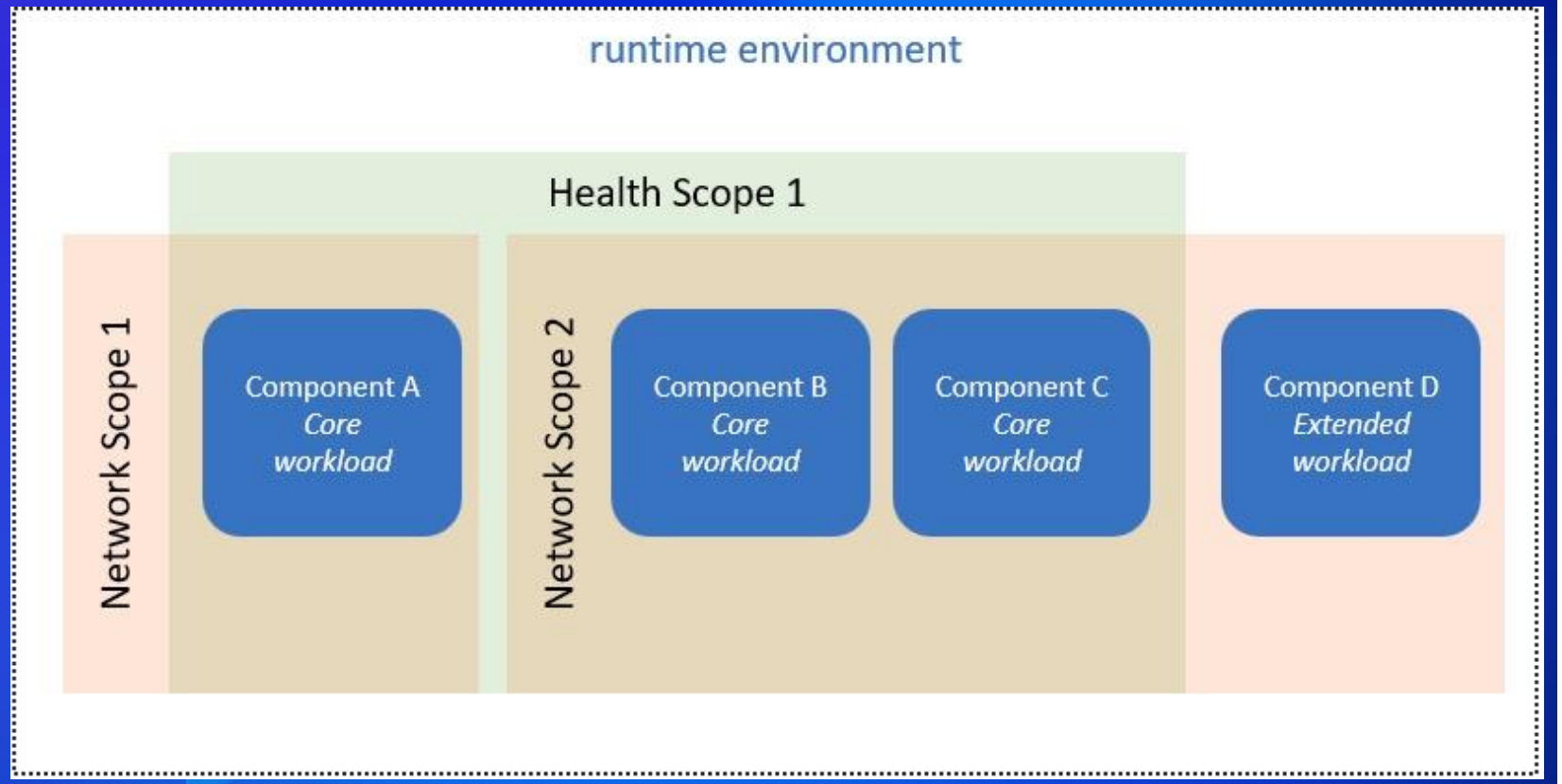
运维可以随意组合、搭配 Traits，就像乐高积木

```yaml
apiVersion: core.oam.dev/v1alpha1
kind: ApplicationConfiguration
metadata:
  name: my-awesome-app
spec:
  components:
    - componentName: nginx
      instanceName: web-front-end
      parameterValues:
        - name: connections
          value: 4096
      traits:
        - name: auto-scaler
          properties:
            minimum: 3
            maximum: 10
        - name: security-policy
          propoerties:
            allowPrivilegeEscalation: false
```

OAM 实现负责保证

1. Trait 按照定义的顺序绑定给应用
2. Trait 之间如果冲突，直接报错

有一种特殊的运维能力，叫做“边界”



1. Network scope
2. Health scope
3. Resource quota scope
4. Security scope
5. Identity scope
6. …

```yaml
apiVersion: core.oam.dev/v1alpha1
kind: ApplicationConfiguration
metadata:
  name: my-vpc-network
spec:
  variables:
    - name: networkName
      value: "my-vpc"
  scopes:
    - name: network
      type: core.oam.dev/v1alpha1.Network
      properties:
        - name: network-id
          value: "[fromVariable(networkName)]"
        - name: subnet-id
          value: "my-subnet"
---
apiVersion: core.oam.dev/v1alpha1
kind: ApplicationConfiguration
metadata:
  name: custom-single-app
  annotations:
    version: v1.0.0
    description: "Customized version of single-app"
spec:
  variables:
    - name: message
      value: "Well hello there"
    - name: domainName
      value: "www.example.com"
  components:
    - componentName: frontend
      instanceName: web-front-end
      parameterValues:
        - name: message
          value: "[fromVariable(message)]"
      traits:
        - name: Ingress
          properties:
            - name: host
              value: "[fromVaraible(domainName)]"
            - name: path
              value: "/"
      applicationScopes:
        - my-vpc-network

    - componentName: backend
      instanceName: database
      applicationScopes:
        - my-vpc-network
```

组装

App Ops

Trait

App Scope

Component

App Dev

Component

OAM Model

```
apiVersion: core.oam.dev/v1alpha1
kind: ApplicationConfiguration
metadata:
  name: my-awesome-app
spec:
  components:
    - componentName: frontend
      instanceName: web-front-end
      traits:
        - name: Ingress
          properties:
            - name: path
              value: "/"
      applicationScopes:
        - my-vpc-network
    - componentName: backend
      instanceName: database
      applicationScopes:
        - my-vpc-network
```

OAM YAML

+
Rudr

slb-7yt83n

container-8q93re

rds-1uy31m

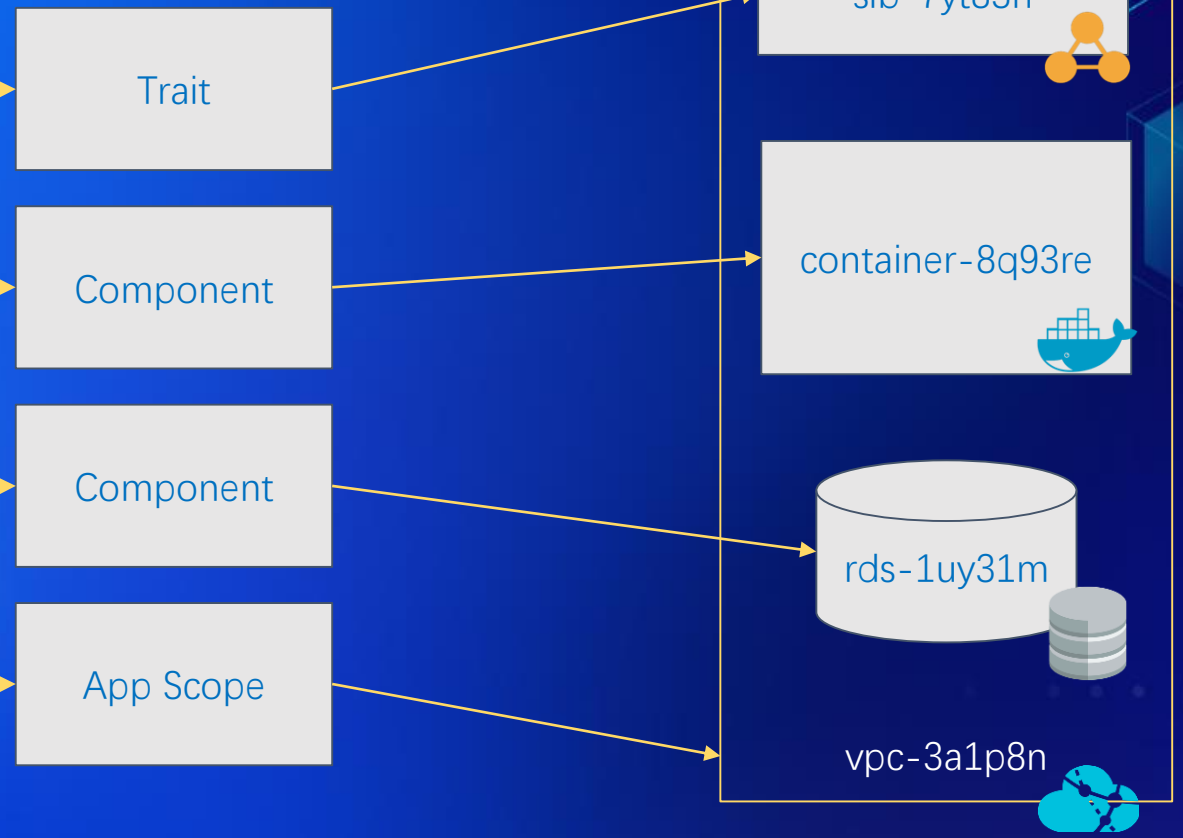
vpc-3a1p8n

Real-world instances

https://github.com/oam-dev/rudr

自包含

```
apiVersion: core.oam.dev/v1alpha1
kind: ApplicationConfiguration
metadata:
  name: my-awesome-app
spec:
  components:
    - componentName: frontend
      instanceName: web-front-end
      traits:
        - name: lngress
          properties:
            - name: path
              value: "/"
      applicationScopes:
        - my-vpc-network
    - componentName: backend
      instanceName: database
      applicationScopes:
        - my-vpc-network
```



OAM YAML

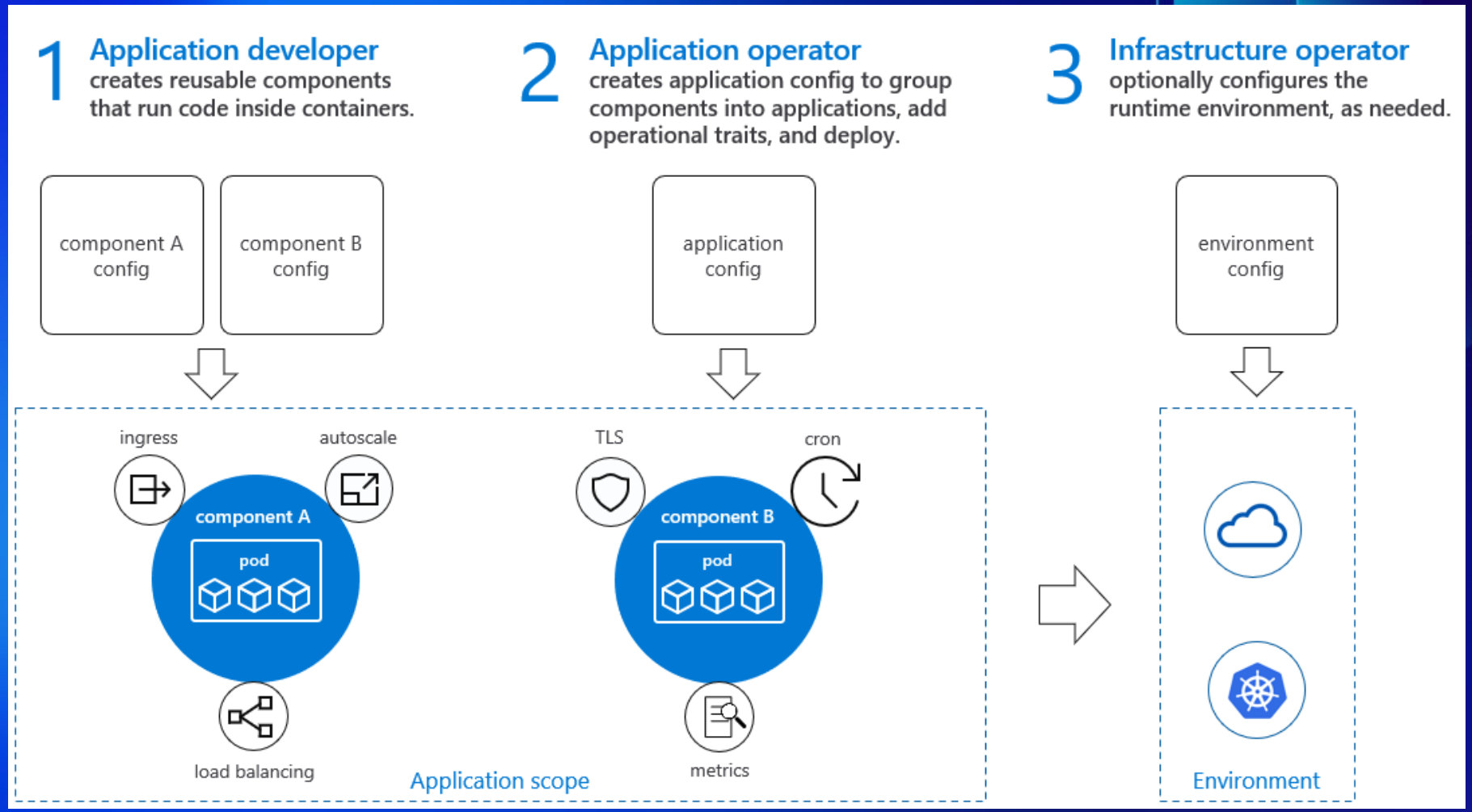
OAM Model

Real-world resources

基于 OAM 的应用管理工作流

https://openappmodel.io/

- 关注点分离：解耦不同角色
- 提供应用抽象：解耦应用定义与底层实现
- K8s 原生：跟云原生生态更好结合



Rudr：基于K8s的OAM实现：github.com/oam-dev/rudr

Thanks For Watching

本PPT来自2019携程技术峰会

更多信息请关注“携程技术中心”微信公众号~