GOPHER CHINA 2020

中国 上海 / 2020-11.21-22

Golang大规模云原生应用管理实践

刘洋（炎寻）

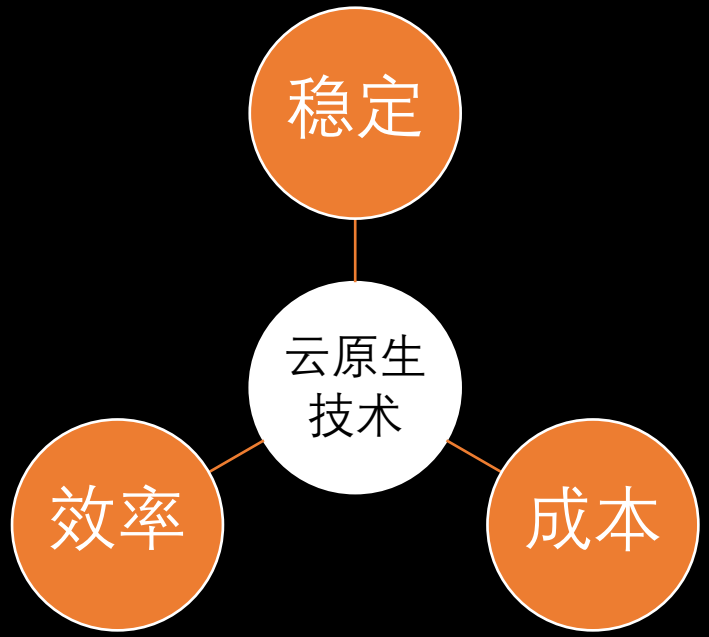
关于我

- 毕业于中国科学技术大学，定居杭州
- 就职于阿里云-云原生应用平台团队
- Problem Solver，聚焦中间件，容器，Kubernetes，PaaS平台...
- OAM社区成员

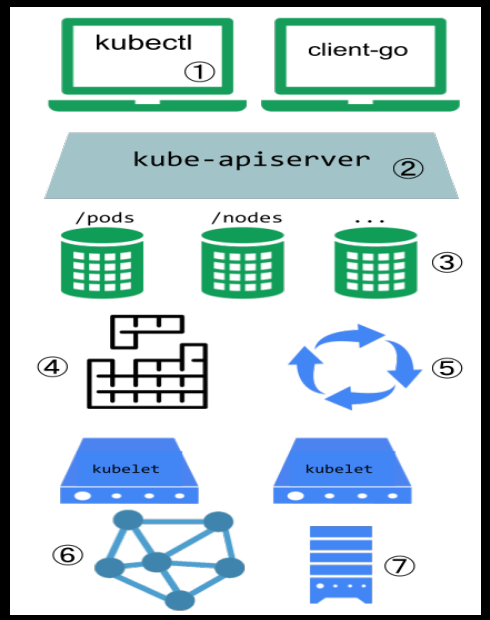
开局一张图

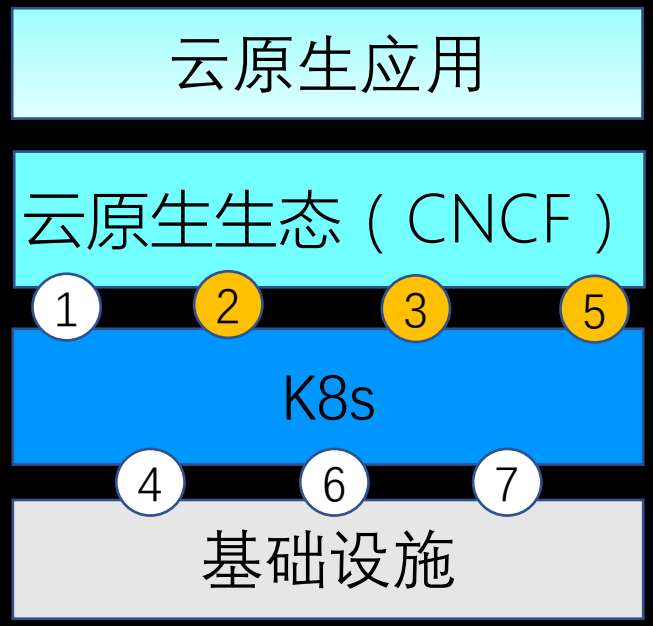
规模化应用交付效率对比去年 **提升1倍**

每万笔峰值交易的IT成本对比4年前 **下降80%**



云原生-程序员视角





非业务逻辑剥离，
提升交付效率

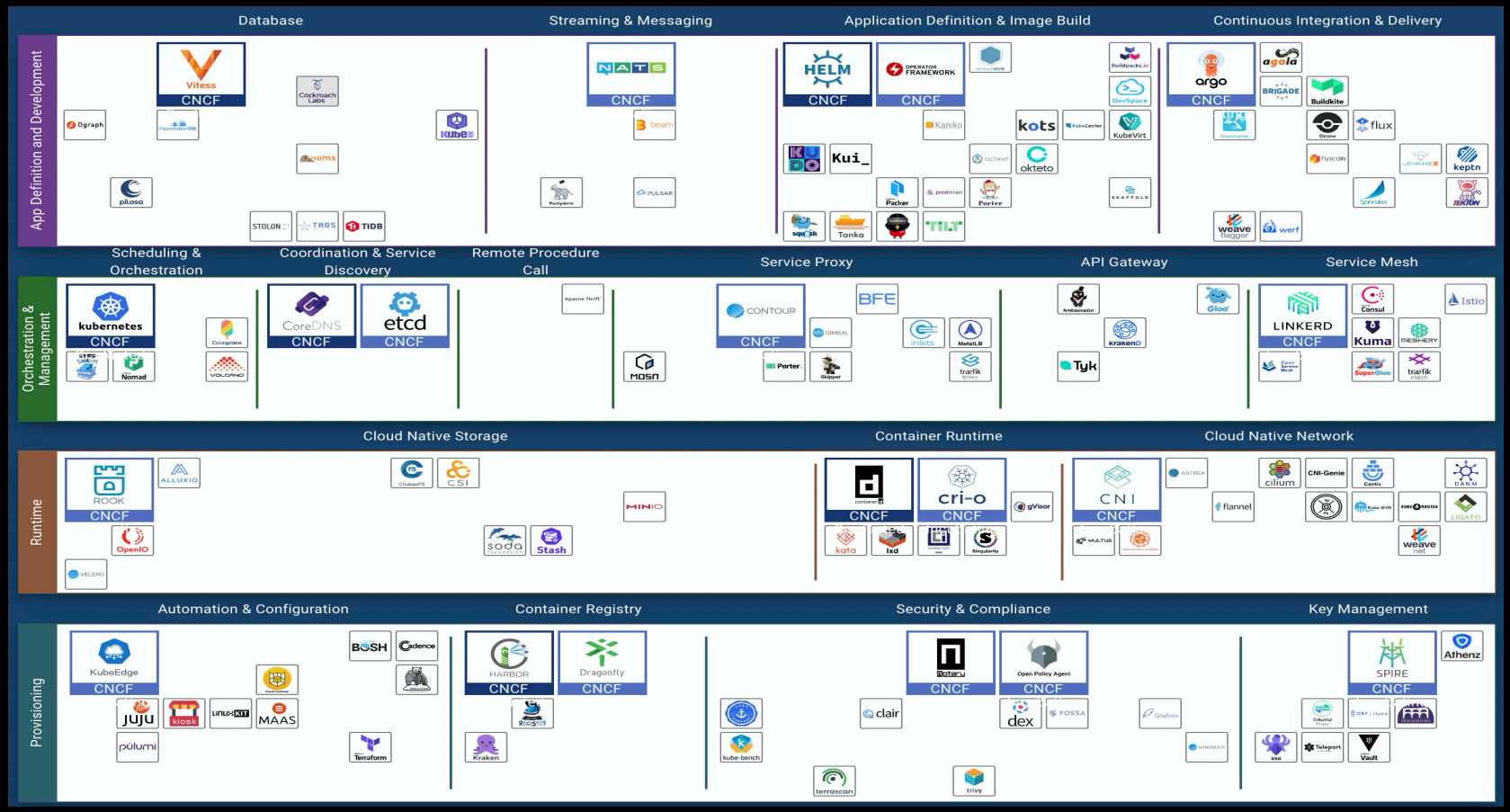
自动化运维，
提升稳定性

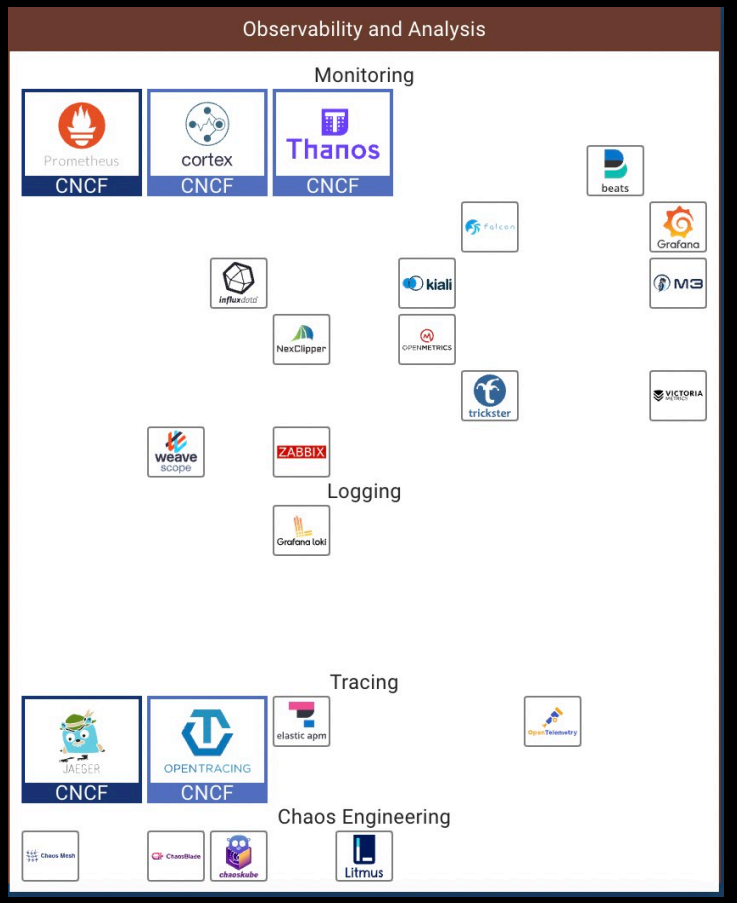
统筹规划，
降低成本

1. Kubectl plugins
2. Apiserver extension
3. Custom resources
4. Scheduler extension
5. Custom controller
6. Network plugins
7. Storage plugins

云原生是以容器技术为基础围绕着Kubernetes进行的一场技术标准化演进。通过标准可扩展的调度，网络，存储，容器运行时接口来提供基础设施；通过标准可扩展的声明式资源和控制器来提供运维能力。两层标准化推进了细化的社会分工，各领域进一步提升规模化和专业化，全面达到成本，效率，稳定性的优化。

Golang与云原生生态（CNCF）





项目数占比: 214/1512（14.2%） Github star数占比:1265737 / 2458072（51.5%）市值占比: $8.08T/$19.46T(41.5%)

GOPHER CHINA 2020
中国 上海 / 2020-11.21-22

https://landscape.cncf.io/format=card-mode&fullscreen=yes&grouping=no&language=Go

截止 2020.11.15

插入：策略（Policy）与机制（Mechanism）

策略是做事的一组概念和计划，关注要做什么事
“what”

- 员工进入公司需要验证是一个策略，人脸识别是机制；
- 从杭州到上海是策略，坐火车是机制；
- 接口是策略，实现是机制；
- 声明是策略，过程是机制；

机制是获取结果的过程，方法和系统，关注如何做事
“how”

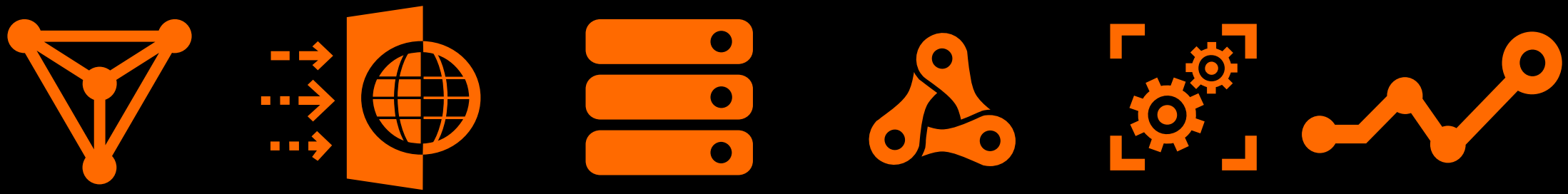
- 策略面向外部交互，机制面向内部实现；
- 策略追求开放标准，机制追求稳定可复用；
- 策略与机制要分离；
- 策略与机制随着层次的变化而变化；

应用管理的策略与机制

K8s及云原生生态给开发者提供的是机制

开发者直接使用K8s的失败故事

- 认知成本高：K8s功能强大却没有统一的使用方式，不得不学习复杂的声明字段和各种奇怪的Annotation；
- 稳定性不足：没有设置Pod的QoS等级，导致频繁被驱逐，没有设置反亲和性策略，导致节点流量不均;
- 扩展效率低：需要负责安装，升级丰富的云原生插件，无法解决插件的依赖，冲突和资源浪费问题；
- 运维成本高：Apiserver, etcd, Controller-Manager, Kubelet,等组件都具有一定复杂度，无法做到定期升级以维持安全，高可用，高性能的状态；
- ...

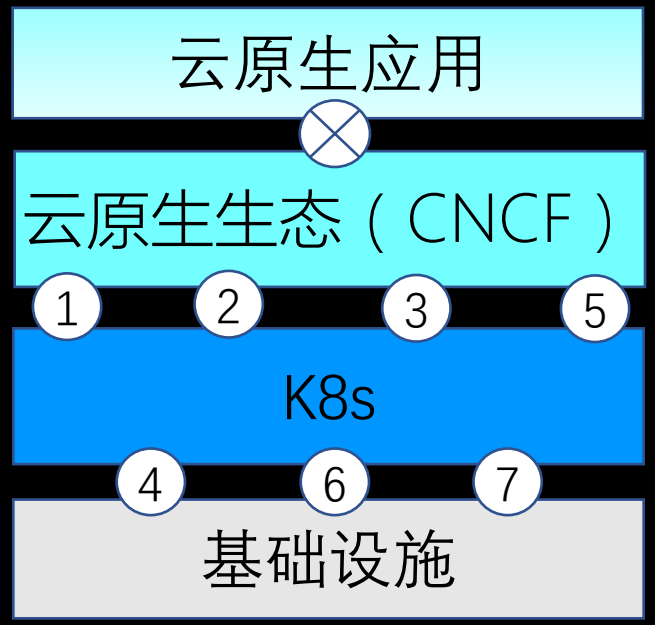


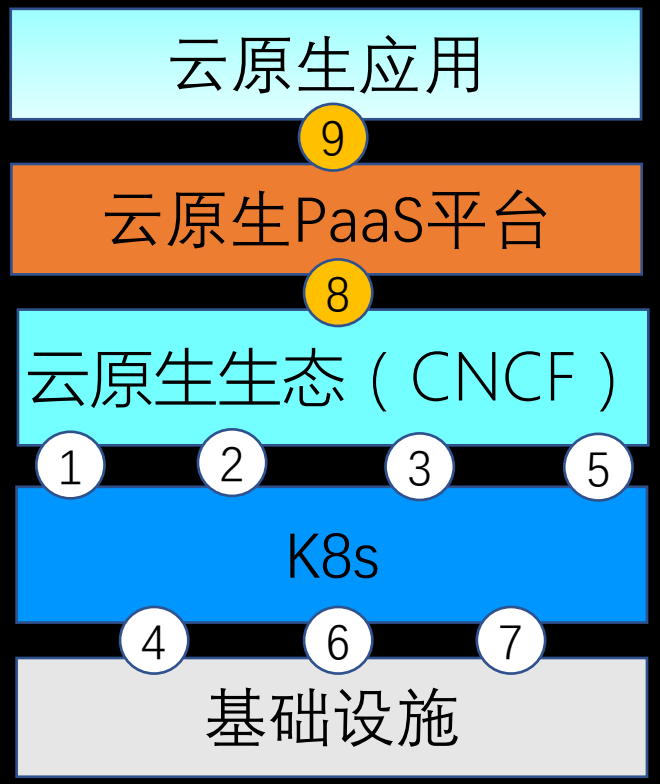
稳定　安全　基础设施　能力复用　自动化　可观测

开发者真正想要的是策略：大象无形的基础设施，坚如磐石的中间件，丰富高效的应用PaaS平台

云原生PaaS平台提供应用管理策略

1. Kubectl plugins
2. Apiserver extension
3. Custom resources
4. Scheduler extension
5. Custom controller
6. Network plugins
7. Storage plugins

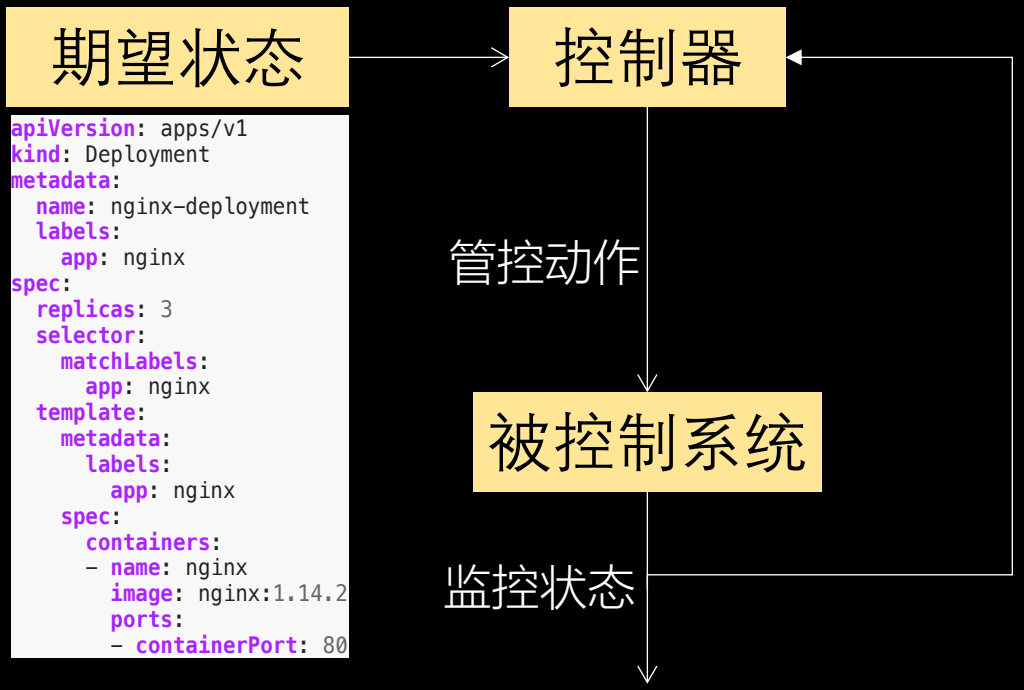




8 向下设计平台策略与机制融入云原生生态

9 向上提供应用管理策略与机制使用平台

插入：K8s核心机制-声明式资源与控制器




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

基于控制论原理

```go
func (dc *DeploymentController) syncDeployment(key string) error {
        ...
        deployment, err := dc.dLister.Deployments(namespace).Get(name)
        d := deployment.DeepCopy()

        // 检查rs
        rsList, err := dc.getReplicaSetsForDeployment(d)
        if err != nil {
                return err
        }

        // 获取所有属于该Deployment的Pod
        podMap, err := dc.getPodMapForDeployment(d, rsList)
        if err != nil {
                return err
        }

        // rolloutRolling发现没有和Deployment template一致的rs会创建新
的
        switch d.Spec.Strategy.Type {
        case apps.RecreateDeploymentStrategyType:
                return dc.rolloutRecreate(d, rsList, podMap)
        case apps.RollingUpdateDeploymentStrategyType:
                return dc.rolloutRolling(d, rsList)
        }
        ...
}
```

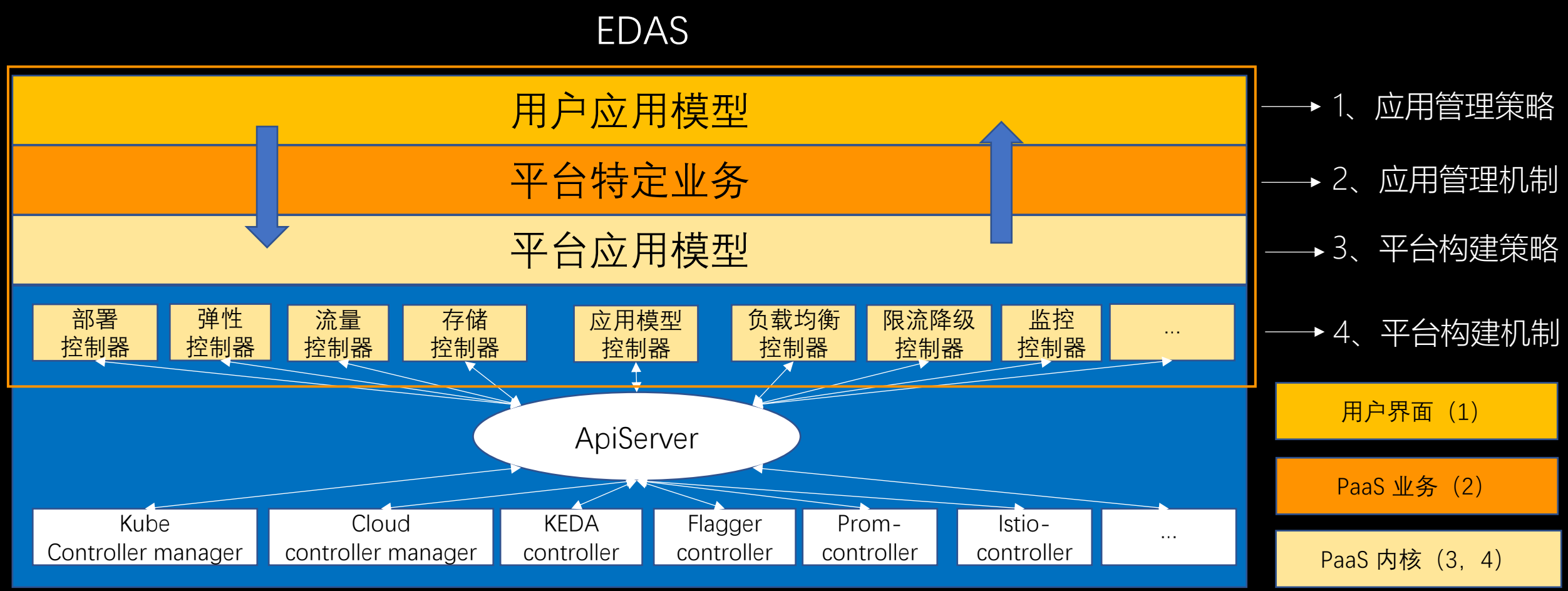
```go
func (dc *DeploymentController) processNextWorkItem() bool {
        key, quit := dc.queue.Get()
        if quit {
                return false
        }
        defer dc.queue.Done(key)

        err := dc.syncHandler(key.(string))
        dc.handleErr(err, key)

        return true
}
```

- 使用声明式K8s资源作为期望终态
- 循环控制器
- Label是一等公民
- 事件触发闭环反馈
- 多控制器组合

EDAS-阿里云云原生PaaS平台



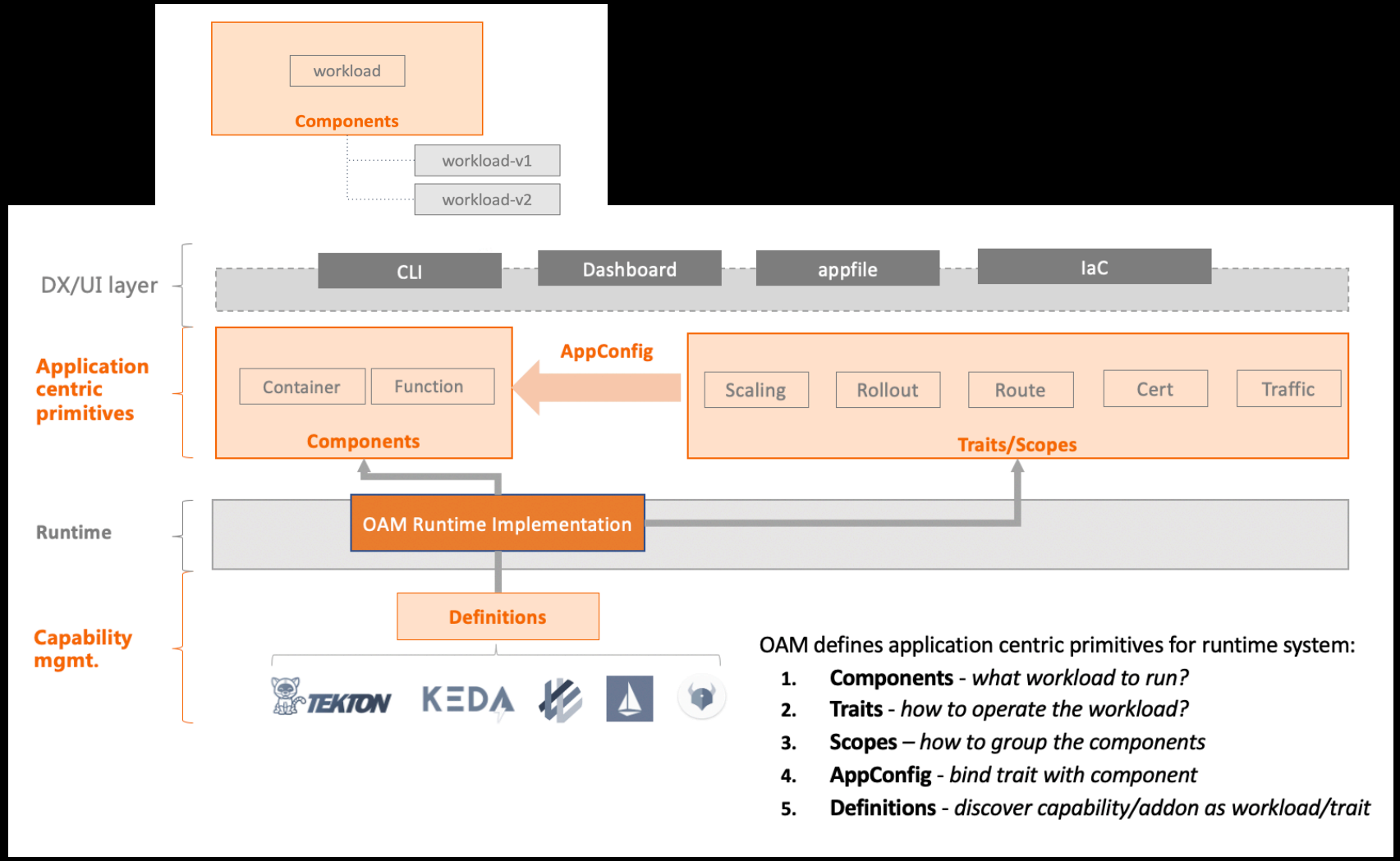
EDAS的平台构建策略-OAM应用模型

- 应用
  - 组件1（工作负载）
    - 运维特征1
    - 运维特征2
    - …
  - 组件2（工作负载）
    - 运维特征1
    - 运维特征2
    - …

    作用域
  - …

能力定义　　依赖编排

组件版本　　服务绑定



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

OAM应用模型

```yaml
apiVersion: core.oam.dev/v1alpha2
kind: TraitDefinition
metadata:
  name: routes.extended.oam.dev
  annotations:
    version: v0.3.0
spec:
  appliesTo:
    - serving.knative.dev/v1.Service
  definitionRef:
     name: routes.extended.oam.dev



apiVersion: core.oam.dev/v1alpha2
kind: TraitDefinition
metadata:
  name: scaledobjects.keda.k8s.io
  annotations:
    version: v0.3.0
spec:
  appliesTo:
    - serving.knative.dev/v1.Service
  definitionRef:
     name: scaledobjects.keda.k8s.io
```

```yaml
apiVersion: core.oam.dev/v1alpha2
kind: Component
metadata:
  name: web-service
  version: v0.3.0
  description: Knative workload
spec:
  workload:
    apiVersion: serving.knative.dev/v1
    kind: Service
    spec:
  template:
    metatdata:
      name: web-service-v1
    spec:
      containerConcurrency: 0
      containers:
      - env:
          - name: TARGET
            value: Knative
        image: helloworld-go:latest
```

```yaml
apiVersion: core.oam.dev/v1alpha2
kind: ApplicationConfiguration
metadata:
  name: helloworld
spec:
  components:
    - componentName: web-service
      instanceName: frontend
      dependencies:
        - backend
      traits:
      - trait:
          apiVersion: keda.k8s.io/v1alpha1
          kind: ScaledObject
          spec:
            maxReplicaCount: 4
            triggers:
            - type: redis
              ...
      - trait:
          apiVersion: extended.oam.dev/v1
          kind: Route
          spec:
            traffic:
            ...
      scopes:
      - scope:
          apiVersion: core.oam.dev/v1alpha2
          kind: Network
          name: my-vpc-network-X
    - instanceName: backend
      componentName: redis
      scopes:
      - scope:
          apiVersion: core.oam.dev/v1alpha2
          kind: Network
          name: my-vpc-network-X
```

- 可组合
- 可发现
- 标准可扩展
- 简单开放

能力定义

- 工作负载
- 运维特征
- 作用域

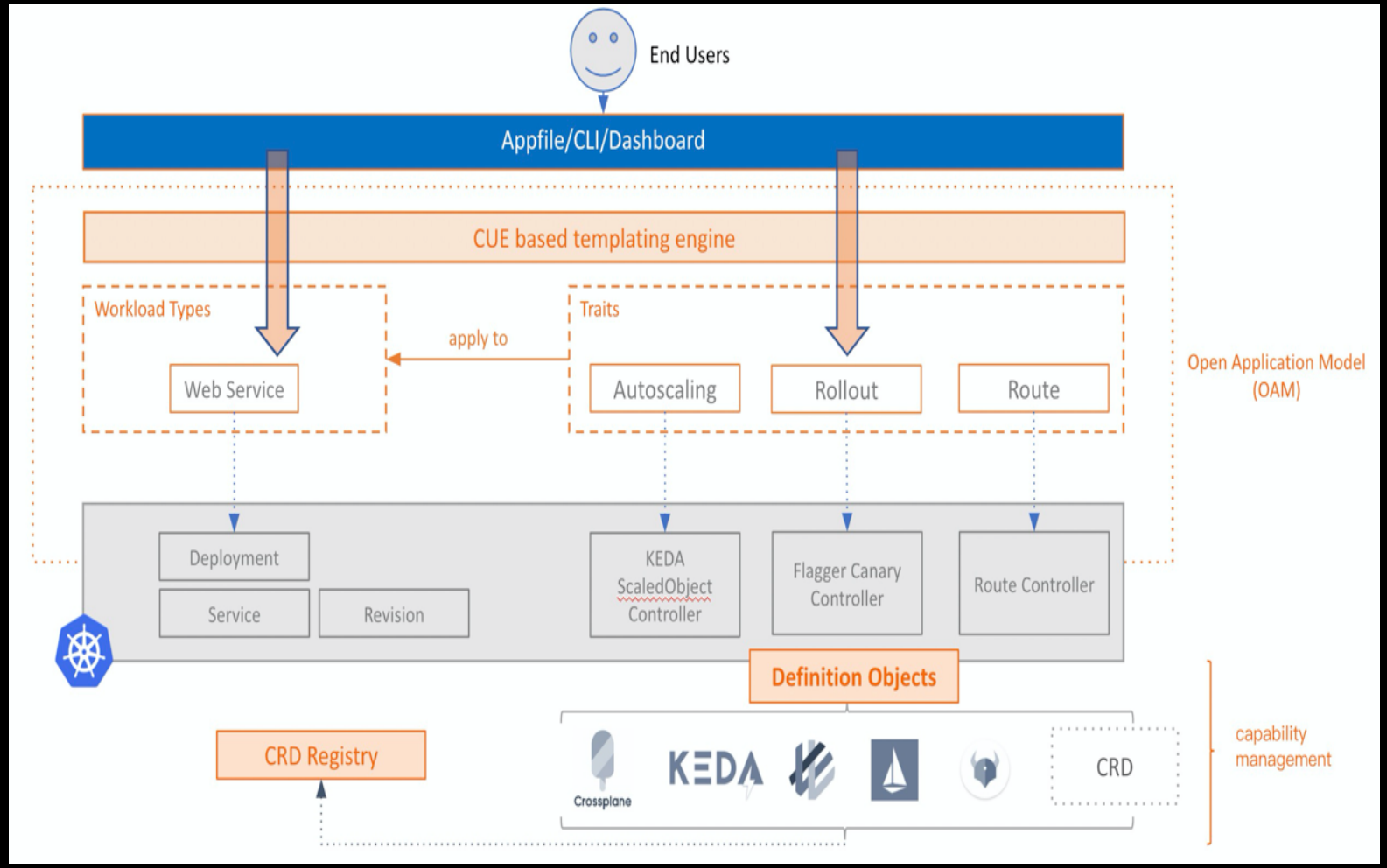
组件

应用配置 = 组件+运维特征+作用域

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

EDAS的平台构建机制-KubeVela

- OAM应用模型运行时
- 内置Workloads & Traits & Scopes
- Capability Management



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

KubeVela的核心机制-运行时

```
func Setup(mgr ctrl.Manager, args controller.Args, l logging.Logger) error {
    ...
    return ctrl.NewControllerManagedBy(mgr).
        Named(name).
        For(&v1alpha2.ApplicationConfiguration{}).
        Watches(&source.Kind{Type: &v1alpha2.Component{}},
&ComponentHandler{
            Client:      mgr.GetClient(),
            Logger:      l,
            RevisionLimit: args.RevisionLimit,
        }).
        Complete(NewReconciler(mgr, dm,
            WithLogger(l.WithValues("controller", name)),
    WithRecorder(event.NewAPIRecorder(mgr.GetEventRecorderFor(name)))))
}
```

控制器初始化

```
func (a *workloads) Apply(ctx context.Context, status []v1alpha2.WorkloadStatus, w
[]Workload, ao ...resource.ApplyOption) error {
    // they are all in the same namespace
    var namespace = w[0].Workload.GetNamespace()
    for _, wl := range w {
        if !wl.HasDep {
            err := a.patchingClient.Apply(ctx, wl.Workload, ao...)
            ...
        }
        for _, trait := range wl.Traits {
            if trait.HasDep {
                continue
            }
            t := trait.Object
            if err := a.updatingClient.Apply(ctx, &trait.Object, ao...); err != nil {
                return errors.Wrapf(err, errFmtApplyTrait, t.GetAPIVersion(),
t.GetKind(), t.GetName())
            }
        }
        workloadRef := runtimev1alpha1.TypedReference{
            APIVersion: wl.Workload.GetAPIVersion(),
            Kind:       wl.Workload.GetKind(),
            Name:       wl.Workload.GetName(),
        }
        for _, s := range wl.Scopes {
            if err := a.applyScope(ctx, wl, s, workloadRef); err != nil {
                return err
            }
        }
    }

    return a.dereferenceScope(ctx, namespace, status, w)
}
```

3.1 依赖满足，新增或更新workload

3.2 依赖满足，新增或更新trait

3.3 新增或更新workload的scope

3.4 删除workload的scope

```
func (r *OAMApplicationReconciler) Reconcile(req reconcile.Request) (result reconcile.Result, returnErr error) {
    ...
    ac := &v1alpha2.ApplicationConfiguration{}
    if err := r.client.Get(ctx, req.NamespacedName, ac); err != nil {
        return reconcile.Result{}, errors.Wrap(resource.IgnoreNotFound(err), errGetAppConfig)
    }
    acPatch := ac.DeepCopy()

    workloads, depStatus, err := r.components.Render(ctx, ac)
    if err := r.workloads.Apply(ctx, ac.Status.Workloads, workloads, resource.MustBeControllableBy(ac.GetUID())); err != nil {
    }
    ...
    for _, e := range r.gc.Eligible(ac.GetNamespace(), ac.Status.Workloads, workloads) {
        e := e
        ...
        if err := r.client.Delete(ctx, &e); resource.IgnoreNotFound(err) != nil {
            ...
        }
        ...
    }

    // patch the final status on the client side, k8s sever can't merge them
    r.updateStatus(ctx, ac, acPatch, workloads)

    ac.Status.Dependency = v1alpha2.DependencyStatus{}
    waitTime := longWait
    if len(depStatus.Unsatisfied) != 0 {
        waitTime = dependCheckWait
        ac.Status.Dependency = *depStatus
    }

    // the posthook function will do the final status update
    return reconcile.Result{RequeueAfter: waitTime}, nil
}
```

1 获取应用期望状态

2 解析期望的WTS状态

3 调谐动作-更新与新增

4 调谐动作-删除

5 调谐动作-状态同步

6 异步轮询

```
func (r *components) Render(ctx context.Context, ac *v1alpha2.ApplicationConfiguration) {
    workloads := make([]*Workload, 0, len(ac.Spec.Components))
    dag := newDAG()
    for _, acc := range ac.Spec.Components {
        w, err := r.renderComponent(ctx, acc, ac, dag)
        ...
        workloads = append(workloads, w)
    }

    ds := &v1alpha2.DependencyStatus{}
    res := make([]Workload, 0, len(ac.Spec.Components))
    for i, acc := range ac.Spec.Components {
        unsatisfied, err := r.handleDependency(ctx, workloads[i], acc, dag, ac)
        ...
        res = append(res, *workloads[i])
    }

    return res, ds, nil
}
```

2.1 渲染workload及其Traits，scopes

2.2 检查依赖

```
func (r *components) renderComponent(ctx context.Context, ...) {
    if acc.RevisionName != "" {
        acc.ComponentName = ExtractComponentName(acc.RevisionName)
    }
    c, componentRevisionName, err := util.GetComponent(...)
    ...
    p, err := r.params.Resolve(c.Spec.Parameters, acc.ParameterValues)
    w, err := r.workload.Render(c.Spec.Workload.Raw, p...)

    for _, ct := range acc.Traits {
        t, traitDef, err := r.renderTrait(ctx, ct, ac, acc.ComponentName, ref, dag)
        ...
    }
    if err := SetWorkloadInstanceName(traitDefs, w, c); err != nil {
        return nil, err
    }
    ...
    for i := range acc.Traits {
        traitDef := traitDefs[i]
        trait := traits[i]
        workloadRefPath := traitDef.Spec.WorkloadRefPath
        if len(workloadRefPath) != 0 {
            ...SetValue(workloadRefPath, workloadRef)...
        }
    }
    scopes := make([]unstructured.Unstructured, 0, len(acc.Scopes))
    for _, cs := range acc.Scopes {
        scopeObject, err := r.renderScope(ctx, cs, ac.GetNamespace())
        ...
        scopes = append(scopes, *scopeObject)
    }

    addDataOutputsToDAG(dag, acc.DataOutputs, w)

    return &Workload{...Workload: w, Traits: traits, RevisionEnabled:
isRevisionEnabled(traitDefs), Scopes: scopes}, nil
}
```

2.1.1 渲染workload

2.1.2 渲染trait

2.1.3 建立workload与trait的关系

2.1.4 建立workload与scope的关系

2.1.5 初始化依赖

EDAS的应用策略与机制

用户应用模型-OAM表单

```yaml
name: testapp

services:
  express-server:
    image: oamdev/testapp:v1
    build:
      docker:
        file: Dockerfile
        context: .

    cmd: ["node", "server.js"]
    port: 8080
    cpu: "0.01"

    route:
      domain: example.com
      rules:
        - path: /testapp
          rewriteTarget: /

    autoscale:
      min: 1
      max: 4
      cpuPercent: 5
```

平台业务

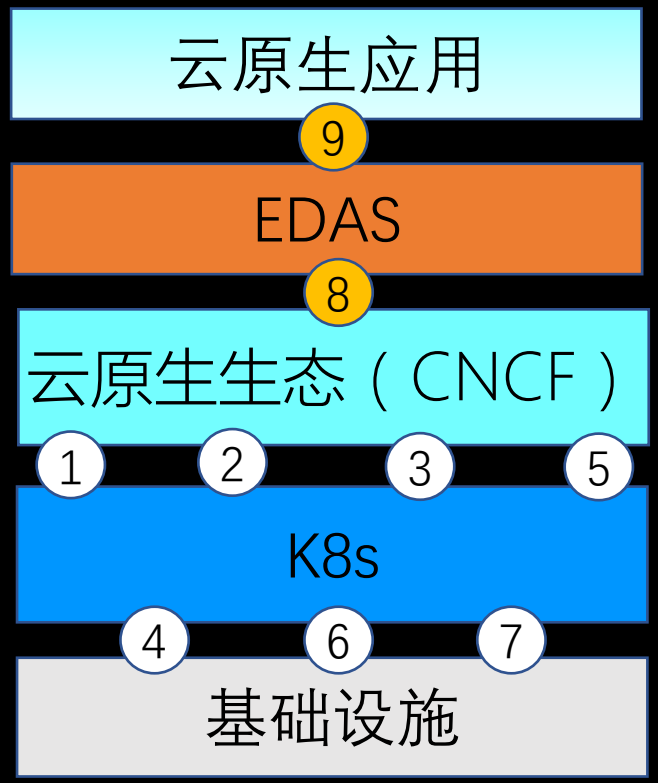
EDAS

平台应用模型-OAM资源

```yaml
apiVersion: core.oam.dev/v1alpha2
kind: ApplicationConfiguration
metadata:
  name: helloworld
spec:
  components:
    - componentName: web-service
      instanceName: frontend
      dependencies:
        - backend
      traits:
      - trait:
          apiVersion: keda.k8s.io/v1alpha1
          kind: ScaledObject
          spec:
            maxReplicaCount: 4
            triggers:
            - type: redis
              ...
      - trait:
          apiVersion: extended.oam.dev/v1
          kind: Route
          spec:
            traffic:
              ...
      scopes:
      - scope:
          apiVersion: core.oam.dev/v1alpha2
          kind: Network
          name: my-vpc-network-X
    - instanceName: backend
      componentName: redis
      scopes:
      - scope:
          apiVersion: core.oam.dev/v1alpha2
          kind: Network
          name: my-vpc-network-X
```

EDAS Review

1. Kubectl plugins
2. Apiserver extension
3. Custom resources
4. Scheduler extension
5. Custom controller
6. Network plugins
7. Storage plugins



9 OAM表单

8 OAM & KubeVela

简单开放

标准
可扩展

轻量化

新的复杂度-开发模式

```
for {
    actualState := GetResourceActualState(rsvc)
    expectState := GetResourceExpectState(rsvc)
    if actualState == expectState {
        // do nothing
    } else {
        Reconcile(rsvc)
    }
}
```

- 声明式资源设计不合理；
- Expect state变化导致大范围Pod重启；
- Reconcile逻辑混乱不堪，不可测试；
- 外部交互模式不匹配；

- 声明式资源设计（要什么）
  - Static
  - Measurable
  - Relevant
  - Attainable
  - Timebound
- 控制器设计（做什么）
  - 基于"可重构"状态机，开放的世界
  - 不要修改资源声明
  - 事件驱动+主动轮询
  - 重试 + 幂等
  - 自愈
- e2e测试
  - Ginkgo BDD
  - Kind本地K8s集群

新的复杂度-最终一致性

```
status:
  …
  phase: succeed
  …
```

- 过期的状态
- 版本冲突
- 业务及时性

```
status:
  currentBatch: 1
  lastPhaseTransitionTime: "2020-10-29T13:43:48Z"
  observedVersion: "5"
  phase: succeed
  reason: ""
  revision: 5
```

使用version和observedVersion做收敛同步

```
metadata:
    …
    resourceVersion:"3442346549"
    ….
```

使用resourceVersion做乐观并发

```
ownerReferences:
 - apiVersion: core.oam.dev/v1alpha2
   blockOwnerDeletion: true
   controller: true
   kind: ApplicationConfiguration
   name: oamhsf
   uid: 73eec338-c3c1-4936-bc5b-3c47f2eb63bc
```

使用ownerRef做事件触发

新的复杂度-可观测

应用为什么没有到终态？

- 关联资源太多
- 关联控制器太多
- 异步响应式

```
status:
  …
  workloads:
  - componentName: oamhsf-group-1
    componentRevisionName: oamhsf-group-1-v5
    status: Ready
    traits:
    - status: success
      traitRef:
        apiVersion: edas.aliyun.oam.com/v1
        kind: Rollout
        name: oamhsf-group-1-trait-5c4fdd8f6c
    - status: success
      traitRef:
        apiVersion: edas.aliyun.oam.com/v1
        kind: ImageBuilder
        name: oamhsf-group-1-trait-5bc99b979f
    workloadRef:
      apiVersion: apps/v1
      kind: Deployment
      name: oamhsf-group-1-v5
```

应用的status里面关联所有资源

```
status:
  conditions:
  - lastTransitionTime: "2020-11-17T18:33:02Z"
    reason: Successfully reconciled resource
    status: "True"
    type: Synced
  …
```

所有资源的status设计condition描述状态

```
LAST SEEN   TYPE     REASON               OBJECT                              MESSAGE
8m56s       Normal   RenderedComponents   applicationconfiguration/oamacree   Successfully rendered components
8m56s       Normal   AppliedComponents    applicationconfiguration/oamacree   Successfully applied components
```

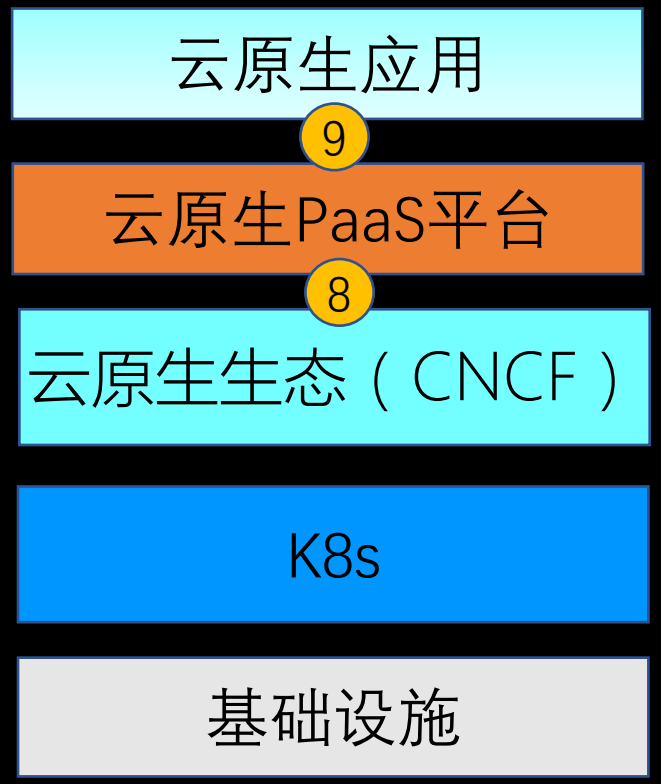
应用关联的事件与日志

新的复杂度-控制器运维

我们要管控大规模的集群，每个集群也会部署大量的控制器，控制器本身的运维成为问题

- 控制器管控平台
  - 升级
  - 回滚
  - 灰度
  - 重启
- 观测性
  - Prometheus
  - 统一日志收集
  - 事件中心
  - 告警
- 能力管控
  - 版本管理
  - 依赖满足
  - 健康检查

云原生PaaS平台的发展趋势



9 标准化用户应用模型

8 标准化平台应用模型与机制

- 开发者体验
  - Serverless = FaaS + BaaS
  - GitOps-自动化
  - ServiceMesh-下一代分布式应用编程模式
  - 应用运维可编程
  - 多元化（云，工作负载，服务）
- 平台
  - 可观测是重中之重
  - 智能化-AIOps
  - 一体化-扩展屏蔽基础设施
  - 轻量化-下沉
  - 自动化-端到端

GOPHER CHINA 2020

中国 上海 / 2020-11.21-22

OAM 社区交流群

1685人

扫一扫群二维码，立刻加入该群。

Thanks

EDAS K8s/Serverless K8s 交...

780人

扫一扫群二维码，立刻加入该群。