云原生社区 Cloud Native Community 火线 CODING CLOUD DEVELOPMENT

云原生社区 meetup
第七期·深圳站

KubeVela:
多云交付的云原生应用交付系统

邓洪超

阿里云高级技术专家

CNCF 应用交付小组联席主席

CLOUD NATIVE COMPUTING FOUNDATION THE LINUX FOUNDATION Broadview www.broadview.com.cn 腾讯游戏 Tencent Games

KubeVela: The Modern App Delivery System for Multi-Cloud Workloads

Hongchao Deng
Staff Engineer @ Alibaba Cloud
Co-Chair @ CNCF App Delivery TAG

What is KubeVela?

- An **easy-to-use** yet **highly extensible** app deployment system targets on today's hybrid, multi-cloud environments.

Tell me more!

- The core component behind web-scale app delivery/management platforms in Alibaba.
- Brings *consistent app delivery workflow* to hybrid (clouds and on-prem) infrastructures in a scalable approach.
- The solution of how Alibaba is adopting **GitOps** and **IaC**.
- It's open source and a CNCF project:
  - https://github.com/oam-dev/kubevela

CLOUD NATIVE
COMPUTING FOUNDATION

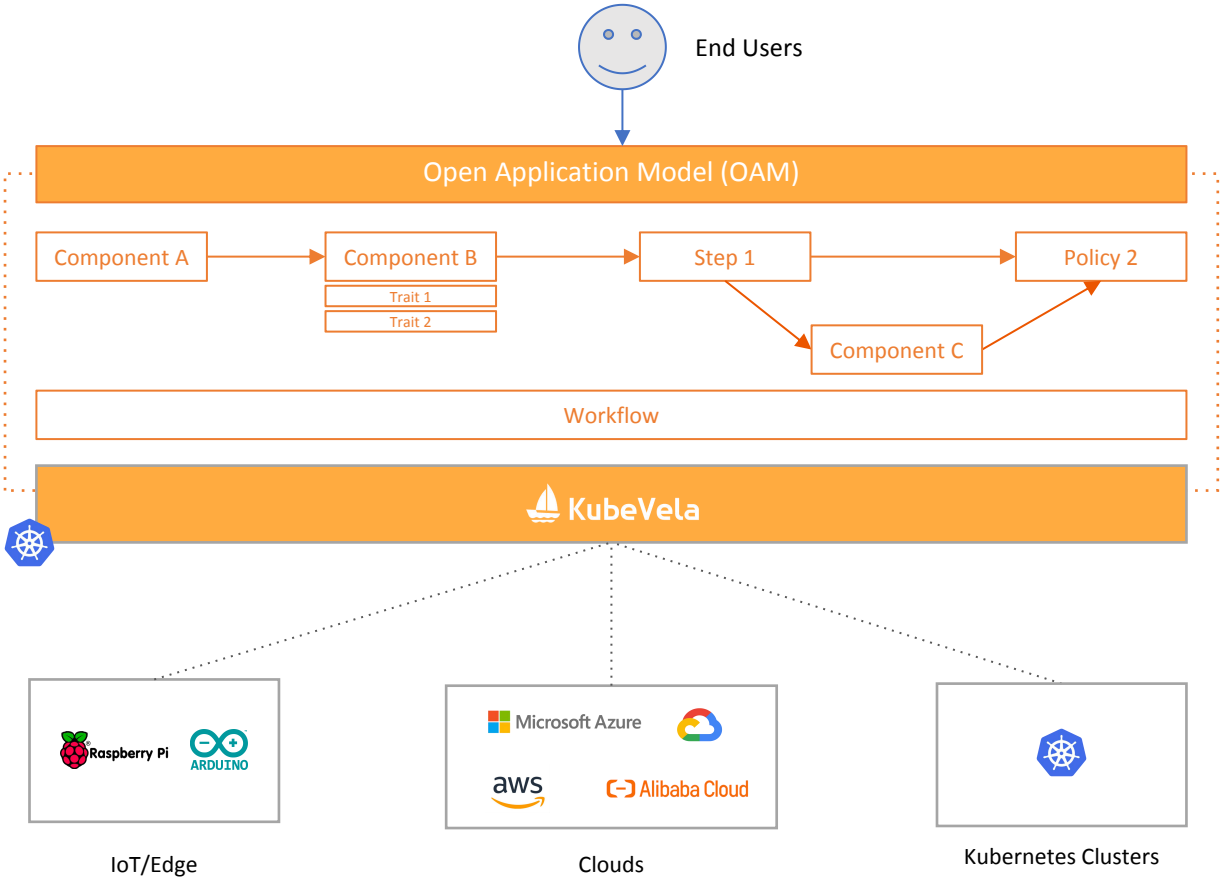
Design Principles

- Application Centric
  - Consistent yet higher level abstraction to capture the full deployment of microservices on top of hybrid environments.
  - No infrastructure level concerns, simply deploy.
- Programmable Workflow
  - Simple yet extensile application deployment workflow.
    - Define the deployment workflow as DAG, with all application's needs and steps glued together in programmable approach (via CUE).
    - No restrictions, natively extensible.
- Runtime Agnostic
  - Works as an application delivery control plane, not a runtime plugin.
    - Can deploy and operate any workload per your needs (e.g. containers, cloud functions, databases, or even EC2 instances).

Architecture

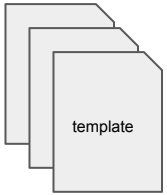
Design deployment topology, policy and workflow via OAM.

Distribute components to target cloud, IoT/Edge device, or Kubernetes cluster, following your policy and workflow.

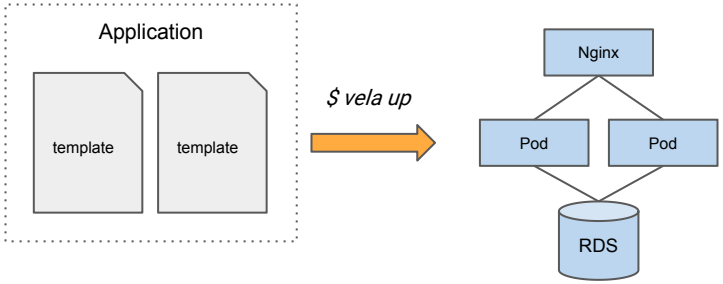


We endorses a team-centric workflow

- *Platform Team*
  - maintaining components, traits, policy, workflow as CUE **templates**



- *App Team*
  - **choosing** templates, **assembling** them into an ***Application*** deployment plan!



X-Definitions

- The “LEGO bricks” to build your deployment plan
  - ComponentDefinition
    - Helm, Kustomize, Terraform, CloudFormation, ROS ...
  - TraitDefinition
    - Canary, Autoscaler, Route ...
  - PolicyDefinition
    - Security, Health, Multi-Env ...
  - WorkflowDefinition
    - Blue-green, Traffic Shifting ...
- X-Definitions are fully programmable
  - 100% CUE templates.
  - Superglue of *EVERYTHING*.
    - K8s, Terraform, REST API, CUE Actions...
  - Provided by specialists to abstract away low level infrastructure details and expose easy-to-use parameters

```yaml
apiVersion: core.oam.dev/v1beta1
kind: TraitDefinition
metadata:
  annotations:
    definition.oam.dev/description: "expose the app"
  name: expose
spec:
  appliesToWorkloads:
    - deployments.apps
  podDisruptive: true
  schematic:
    cue:
      template: |
        patch: {spec: template: metadata: labels: app: context.name}
        outputs: service: {
            apiVersion: "v1"
            kind:       "Service"
            metadata: name: context.name
            spec: {
                selector: app: context.name
                ports: [
                    for k, v in parameter.http {
                        port:       v
                        targetPort: v
                    },
                ]
            }
        }
        parameter: {
            http: [string]: int
        }
```

CUE

Application

- The single source-of-truth to **model a full application deployment.**
- A *composition* object.

```yaml
kind: Application
spec:
  components:
    - name: express-server
      type: webservice
      properties:
        image: demo/hello-world
        port: 8000
      traits:
        - type: ingress
          properties:
            domain: testsvc.example.com
            http:
              "/": 8000
  policies:
    - type: security
      properties:
        audit: enabled
        secretBackend: vault
    - type: deployment-insights
      properties:
        provider: arms | promethues
        leadTime: enabled
        frequency: enabled
        mttr: enabled
  workflow:
    - type: blue-green-rollout
      stage: post-render
      properties:
        partition: "50%"
    - type: traffic-shift
      properties:
        partition: "50%"
    - type: rollout-promotion
      propertie:
        manualApproval: true
        rollbackIfNotApproved: true
```

***Components* to deploy**

e.g. a Helm char, a Kustomize pkg, a Cloud Formation template, a Terraform module … literally *anything*!

***Traits* for day 2 operations**

e.g. ingress/route rules, auto-scaling rules … operational behaviors attached to components!

***Policies* to enforce**

e.g. security scopes, healthy check policies, firewall rules … any policies to enforce before deployment happen!

***Workflow* of the deployment**

e.g. blue-green deploy, progressive traffic shifting, manual approval … any pipeline style delivery steps!

How to Use

App Team

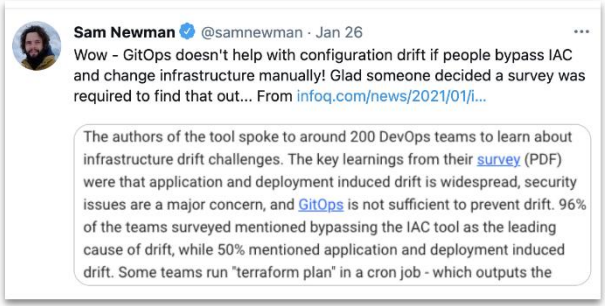
1. Declare an **application** deployment plan in a single file.
2. *$ kubectl apply -f app.yaml*
3. Done!

Deployment plan is the **ONLY** concept users need to learn in KubeVela, and as a K8s custom resource, it works seamlessly with any CI/CD or GitOps tools.

```yaml
kind: Application
spec:
  components:
    - name: express-server
      type: webservice
      properties:
        image: demo/hello-world
        port: 8000
      traits:
        - type: ingress
          properties:
            domain: testsvc.example.com
            http:
              "/": 8000
  policies:
    - type: security
      properties:
        audit: enabled
        secretBackend: vault
    - type: deployment-insights
      properties:
        provider: arms | promethues
        leadTime: enabled
        frequency: enabled
        mttr: enabled
  workflow:
    - type: blue-green-rollout
      stage: post-render
      properties:
        partition: "50%"
    - type: traffic-shift
      properties:
        partition: "50%"
    - type: rollout-promotion
      propertie:
        manualApproval: true
        rollbackIfNotApproved: true
```

Wait, why KubeVela is not built as a pure IaC tool?

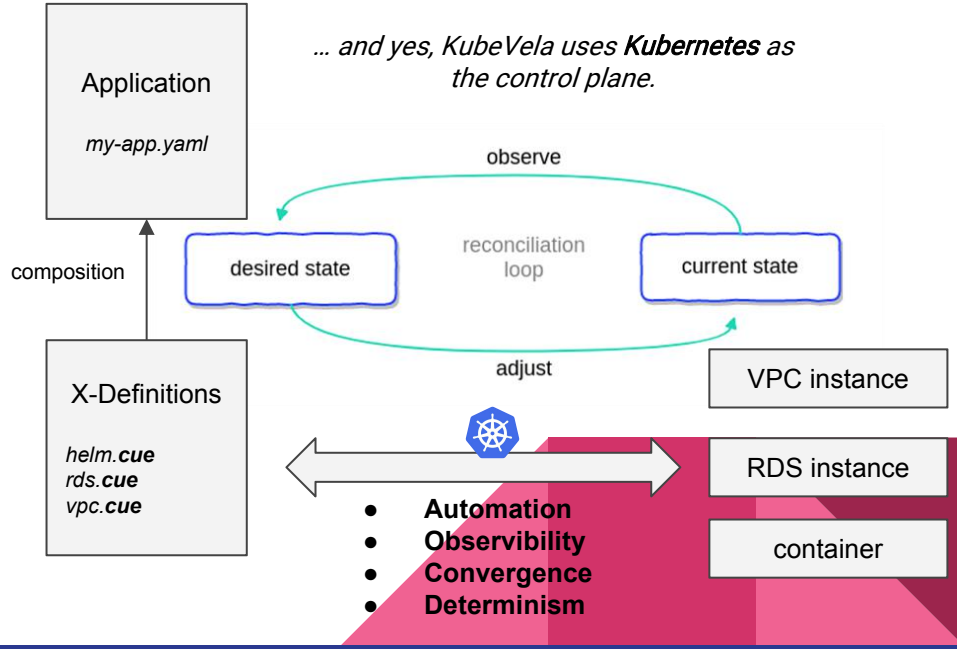
- IaC are perfect in agility, reusability and extensibility.
- But IaC also mean *configuration drift*, difficulty in *maintaining*, *tracing* and *auditing*
  - They can **not** be fixed by “GitOps”.
  - They are **nightmares** in web-scale deployment ...



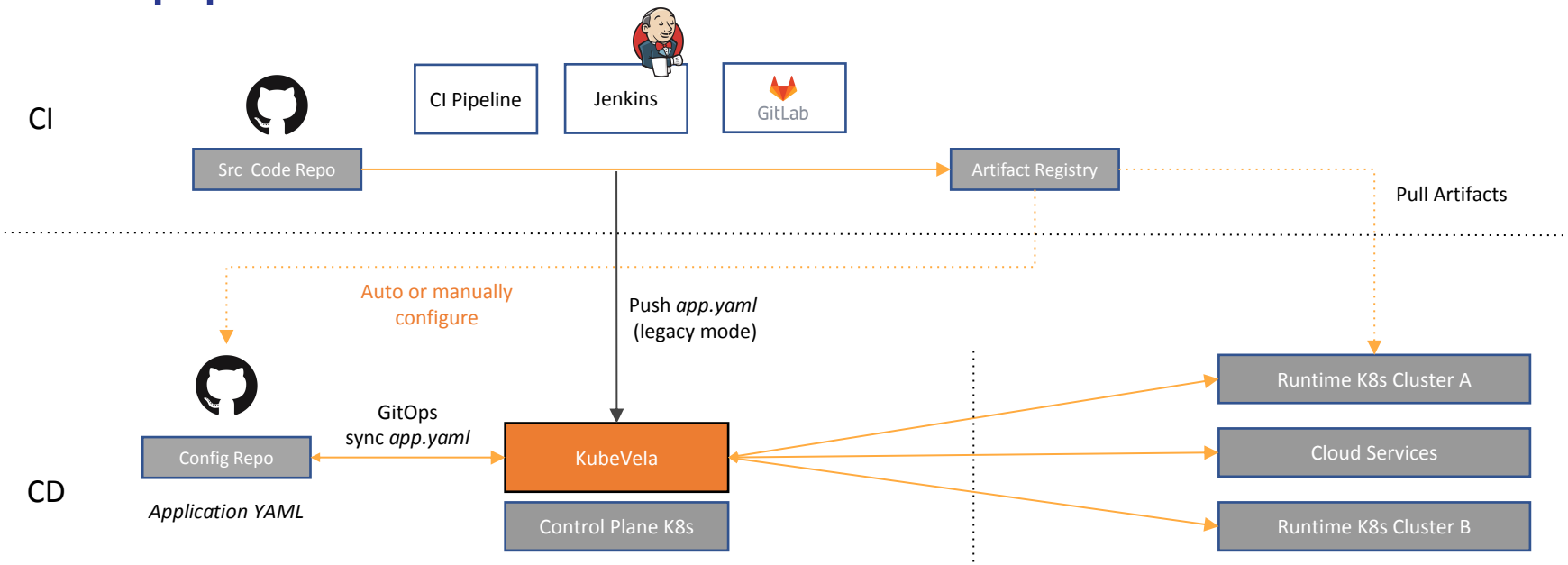
State of infrastructure drift 2021

Solution: with intention to keep all benefits of IaC (via CUE), we need a ***control plane*** to control the app delivery workflow.

*... and yes, KubeVela uses **Kubernetes** as the control plane.*



Our pipeline

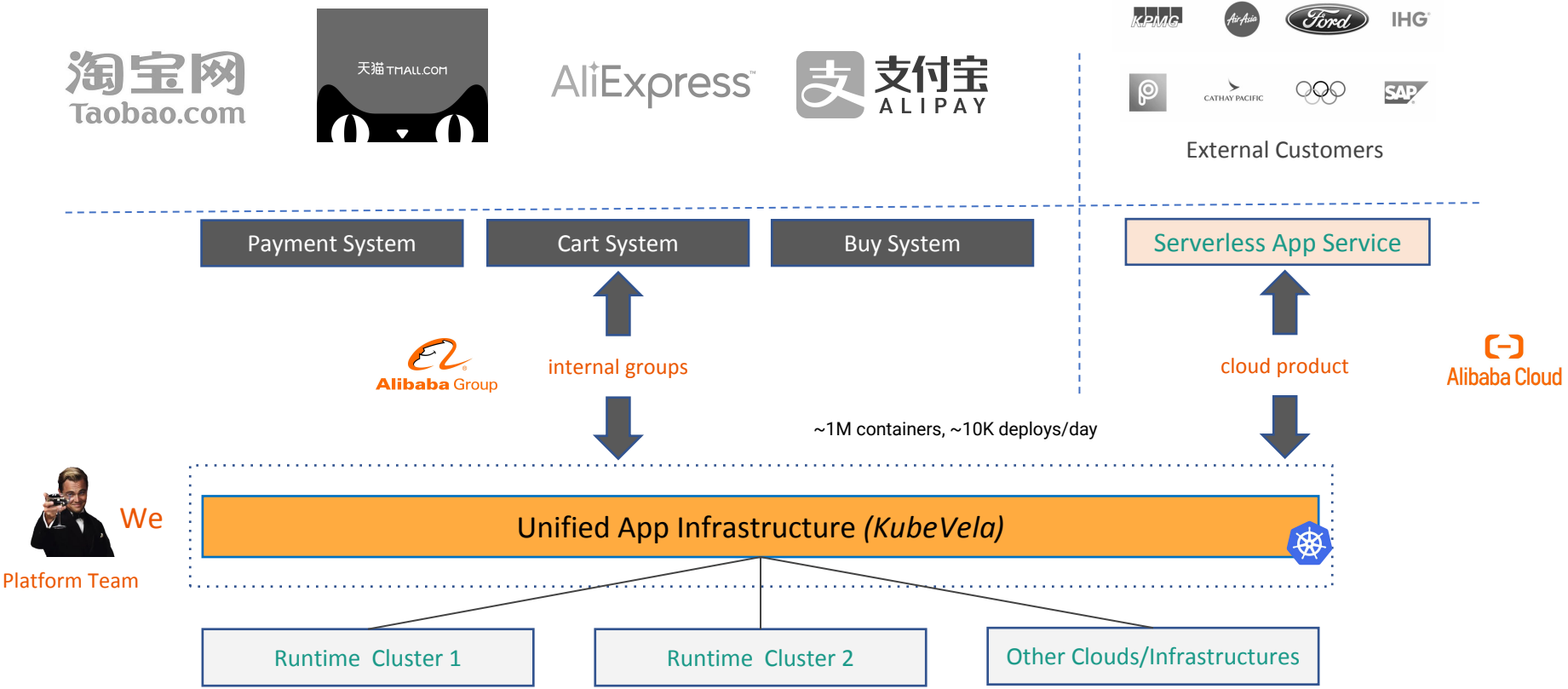


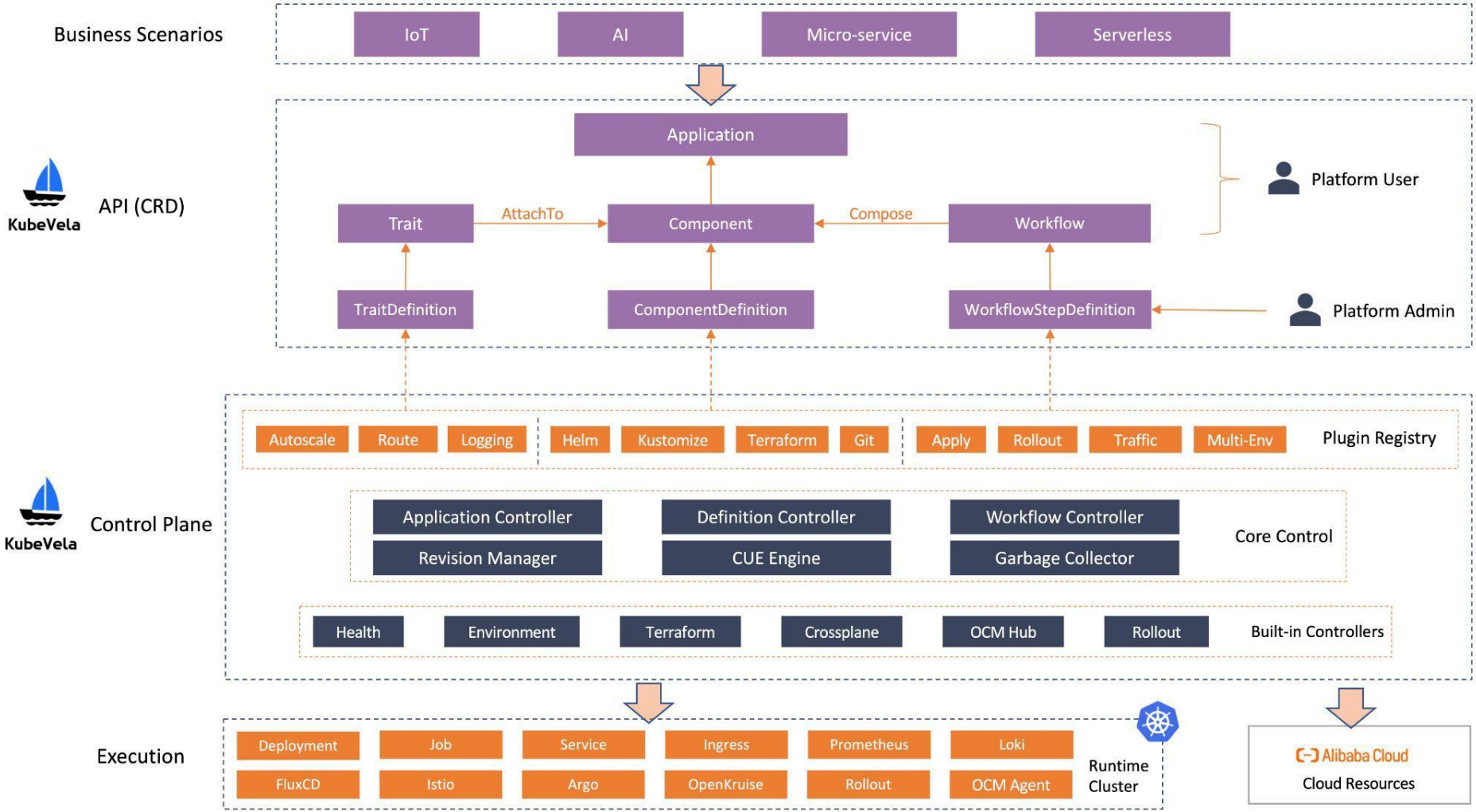
For given app, it now only needs 1 manifest to describe the application plan (across multiple environments), and KubeVela will take over the delivery workflow with full automation and determinism.

Subscription Channel

KubeVela will set up subscription channel for runtime environments to **pull** app metadata and sync.

Alibaba's Application Infrastructure




Business Scenarios
IoT
AI
Micro-service
Serverless
KubeVela
API (CRD)
Application
Trait
AttachTo
Component
Compose
Workflow
Platform User
TraitDefinition
ComponentDefinition
WorkflowStepDefinition
Platform Admin
Autoscale
Route
Logging
Helm
Kustomize
Terraform
Git
Apply
Rollout
Traffic
Multi-Env
Plugin Registry
KubeVela
Control Plane
Application Controller
Definition Controller
Workflow Controller
Revision Manager
CUE Engine
Garbage Collector
Core Control
Health
Environment
Terraform
Crossplane
OCM Hub
Rollout
Built-in Controllers
Execution
Deployment
Job
Service
Ingress
Prometheus
Loki
FluxCD
Istio
Argo
OpenKruise
Rollout
OCM Agent
Runtime Cluster
Alibaba Cloud
Cloud Resources

Case Study 1:

Multi-Cluster Application Deployment

Multi-Cluster App Deployment

Step 1: Register cluster

Step 2: Deploy Application

Step 3: Check health and approve

Join cluster swarm

```
vela cluster join stage-cluster.kubeconfig --name cluster-staging
vela cluster join prod-cluster.kubeconfig --name cluster-prod
```

List clusters

```
$ vela cluster list
CLUSTER         TYPE    ENDPOINT
cluster-prod    tls     https://47.88.4.97:6443
cluster-staging tls     https://47.88.7.230:6443
```

Multi-Cluster App Deployment

Step 1: Register cluster

Step 2: Deploy Application

Step 3: Check health and approve

Application yaml composition

The service component

```
components:
  - name: hello-world-server
    type: webservice
    properties:
      image: crccheck/hello-world
      port: 8000
```

The workflow specifies three steps:
1. deploy to staging; 2. pause and verify; 3. deploy to production

```
workflow:
  steps:
    # deploy to staging env
    - name: deploy-staging
      type: deploy2env
      properties:
        policy: example-multi-env-policy
        env: staging

    # manual check
    - name: manual-approval
      type: suspend

    # deploy to prod env
    - name: deploy-prod
      ...
```

- The env-binding policy defines how to select target cluster to deploy to, and the config patch for the target.
- The health policy defines how to check healthiness of all Application components

```
policies:
  - name: example-multi-env-policy
    type: env-binding
    properties:
      envs:
        - name: prod
          placement:
            clusterSelector:
              name: cluster-prod
          patch: # overlay patch on above components
            components:
              - name: hello-world-server
                type: webservice
                traits:
                  - type: scaler
                    properties:
                      replicas: 3

        - name: staging
          ...

  - name: health-policy-demo
    type: health
    properties:
      probeInterval: 5
      probeTimeout: 10
```

Full Application Yaml

```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: example-app
  namespace: default
spec:
  components:
    - name: hello-world-server
      type: webservice
      properties:
        image: crccheck/hello-world
        port: 8000
      traits:
        - type: scaler
          properties:
            replicas: 1
    - name: data-worker
      type: worker
      properties:
        image: busybox
        cmd:
          - sleep
          - '1000000'
```

```yaml
  policies:
    - name: example-multi-env-policy
      type: env-binding
      properties:
        envs:
          - name: staging
            placement: # selecting the cluster to deploy to
              clusterSelector:
                name: cluster-staging
            selector: # selecting which component to use
              components:
                - hello-world-server

          - name: prod
            placement:
              clusterSelector:
                name: cluster-prod
            patch: # overlay patch on above components
              components:
                - name: hello-world-server
                  type: webservice
                  traits:
                    - type: scaler
                      properties:
                        replicas: 3

    - name: health-policy-demo
      type: health
      properties:
        probeInterval: 5
        probeTimeout: 10
```

```yaml
  workflow:
    steps:
      # deploy to staging env
      - name: deploy-staging
        type: deploy2env
        properties:
          policy: example-multi-env-policy
          env: staging

      # manual check
      - name: manual-approval
        type: suspend

      # deploy to prod env
      - name: deploy-prod
        type: deploy2env
        properties:
          policy: example-multi-env-policy
          env: prod
```

Multi-Cluster App Deployment

Step 1: Register cluster

Step 2: Deploy Application

Step 3: Check health and approve

The second step in workflow will suspend to wait for manual verification

```
$ kubectl get application example-app -o yaml
NAME          COMPONENT            TYPE         PHASE                HEALTHY   STATUS
example-app   hello-world-server   webservice   workflowSuspending   true      Ready:1/1
```

Check Application status to verify it is running successfully in target environment

```yaml
status:
  services:
  - env: staging
    healthy: true
    message: 'Ready:1/1 '
    name: hello-world-server
    scopes:
    - apiVersion: core.oam.dev/v1alpha2
      kind: HealthScope
      name: health-policy-demo
      namespace: test
      uid: 6e6230a3-93f3-4dba-ba09-dd863b6c4a88
    traits:
    - healthy: true
      type: scaler
```

Resume workflow and continue deployment process (to production environment)

```
$ vela workflow resume example-app
Successfully resume workflow: example-app

$ kubectl get application example-app
NAME          COMPONENT            TYPE         PHASE     HEALTHY   STATUS
example-app   hello-world-server   webservice   running   true      Ready:1/1
```

Case Study 2:

Make Istio Simpler

Make Istio Simpler

Step 1: Enable Istio Addon

Install Istio plugins into the cluster via KubeVela Addon

```
vela addon enable istio
```

Step 2: Rollout new version and split traffic

Label `default` namespace to enable Istio auto-injection

```
kubectl label namespace default istio-injection=enabled
```

Step 3: Rollback

Make Istio Simpler

Step 1: Enable Istio Addon

Step 2: Rollout new version and split traffic

Step 3: Rollback

Component section has the image updated

```yaml
components:
  - name: reviews
    type: webservice
    properties:
      image: docker.io/istio/examples-bookinfo-reviews-v3:1.16.2
```

The workflow has three steps:

1. Rollout first batch and split 10% traffic to the new version
1. Pause and verify the new version
1. Rollout the rest of batches

The workflow abstracts users from low-level infra details

```yaml
workflow:
  steps:
    - name: rollout-1st-batch
      type: canary-rollout
      properties:
        # just upgrade first batch of component
        batchPartition: 0
        traffic:
          weightedTargets:
            - revision: reviews-v1
              weight: 90 # 90% shift to new version
            - revision: reviews-v2
              weight: 10 # 10% shift to new version

    # give user time to verify part of traffic shifting to newRevision
    - name: manual-approval
      type: suspend

    - name: rollout-rest
      type: canary-rollout
      properties:
        # upgrade all batches of component
        batchPartition: 1
        traffic:
          weightedTargets:
            - revision: reviews-v2
              weight: 100 # 100% shift to new version
```

Make Istio Simpler

Step 1: Enable Istio Addon

Step 2: Rollout new version and split traffic

Step 3: Rollback

If we want to rollback to the old version, it is as simple as just putting a rollback step to the workflow

```
workflow:
  steps:
    - name: rollback
      type: canary-rollback
```

It will rollback all replicas and traffic to the old version

Demo Time!

https://www.bilibili.com/video/BV1xq4y1P7A3/

Roadmap

- Support standalone mode
  - Deploy in docker/binary without relying on K8s
- Enhanced dashboard
  - App delivery workflow, addon center, environment, etc.
- Enhanced CLI
  - vela config, vela doctor, vela wfstep
- Observability: application monitoring
- Stability & debuggability
- ArgoCD integration
- Machine learning case study

Thank You!

- KubeVela 文档:
  https://kubevela.io
- 点击 **Star**:
  https://github.com/oam-dev/kubevela
- 用户注册登记:
  https://github.com/oam-dev/kubevela/issues/1662

用户讨论钉钉群

想了解更多，请关注我们的公众号或点击

https://cloudnative.to

云原生社区

Cloud Native Community