
KubeCon
CloudNativeCon
Europe 2023
TiKV

Challenges of Modern Application Delivery: A Retrospection of KubeVela Highlight Technologies

Jianbo Sun, Alibaba Cloud
Da Yin, Alibaba Cloud

Speaker

KubeCon | CloudNativeCon Europe 2023

Jianbo Sun

Staff Engineer, KubeVela Maintainer

Alibaba Cloud

Da Yin

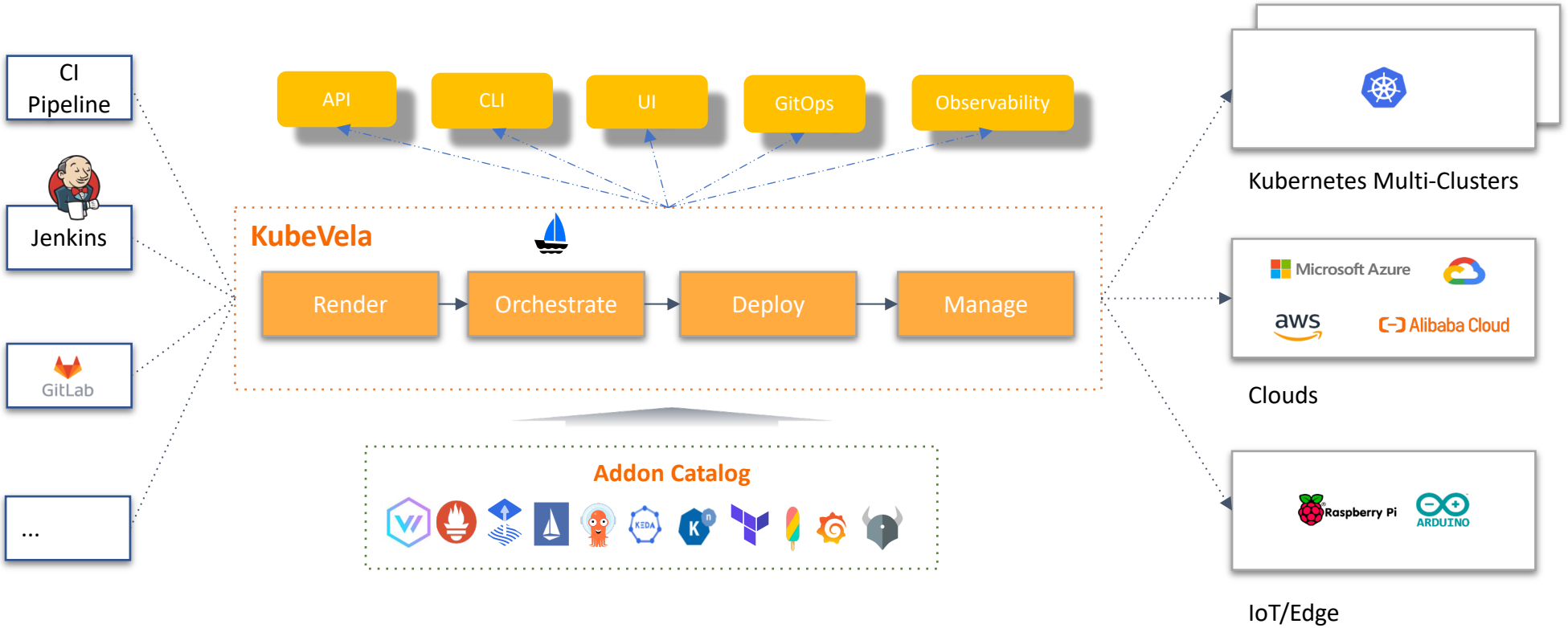
Senior Engineer, KubeVela Maintainer

Alibaba Cloud

What is KubeVela

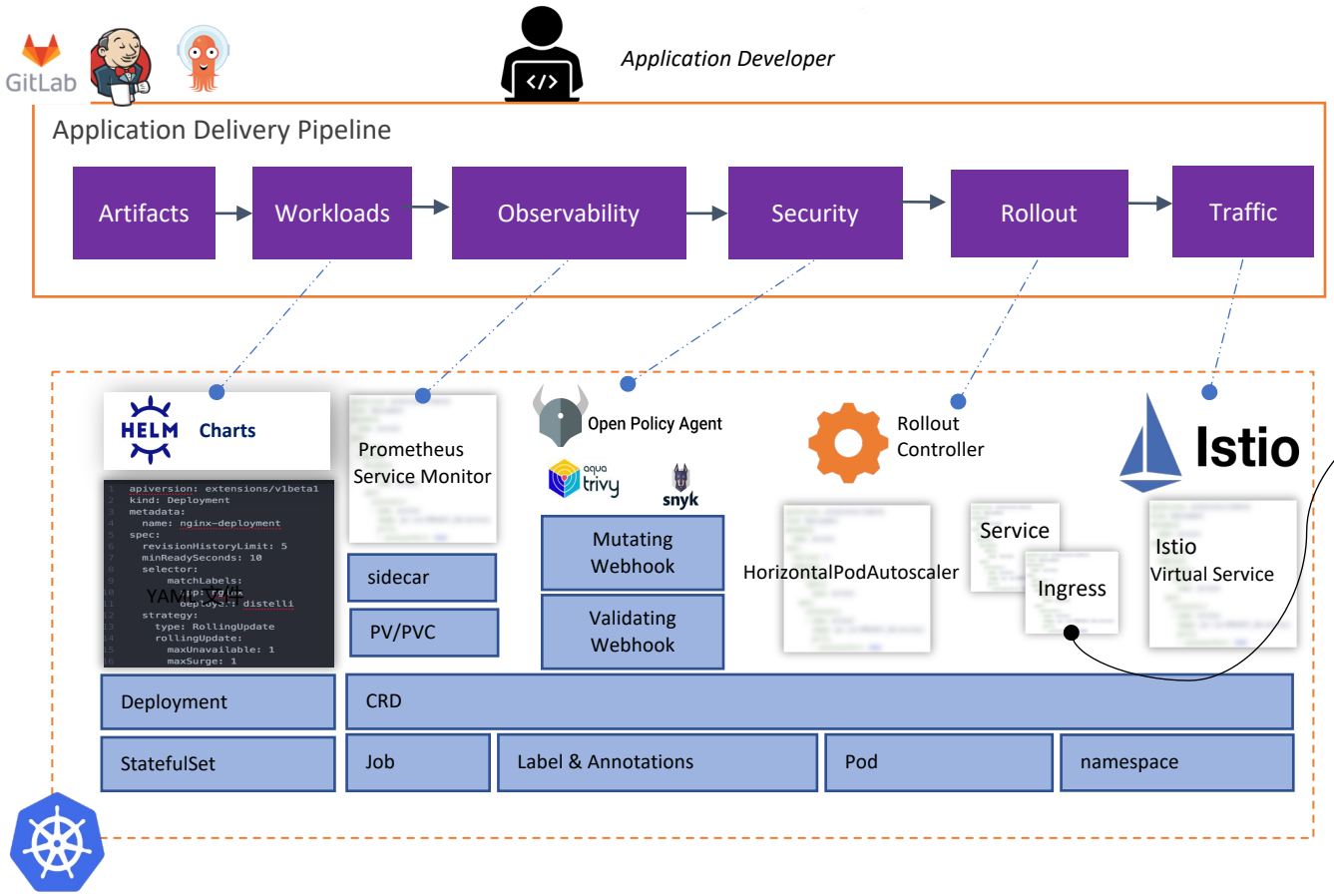
KubeVela is a modern software platform that makes delivering and operating applications across today's hybrid, multi-cloud environments easier, faster and more reliable.

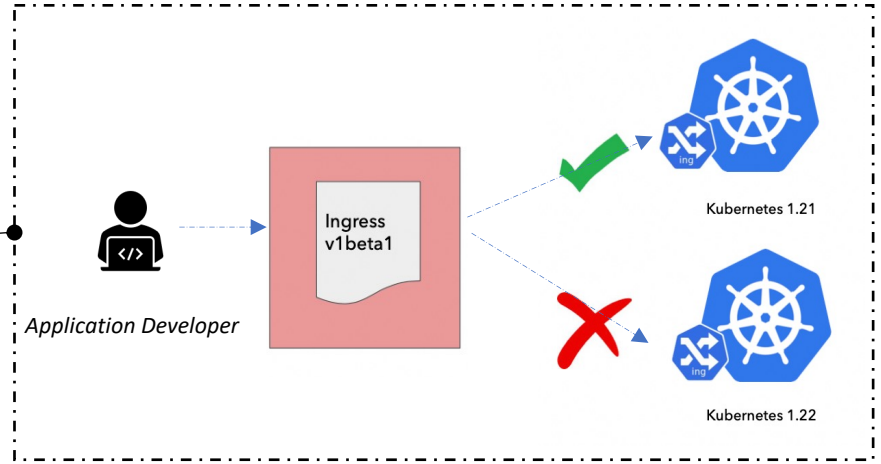
- ❑ Infrastructure agnostic
- ❑ Programmable
- ❑ Application-centric



Why KubeVela

- Kubernetes and cloud complexities are exposed directly to application developers.



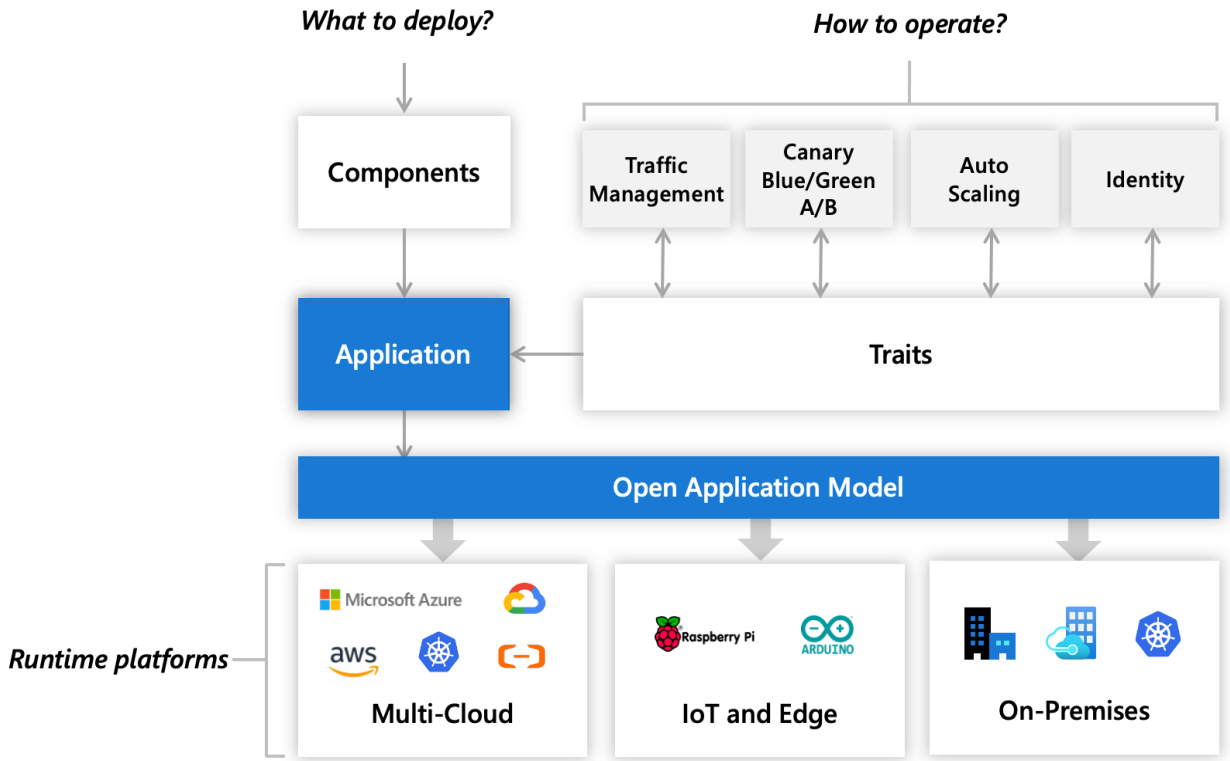


e.g. Kubernetes deprecations are exposed.

https://thenewstack.io/how-intuits-platform-engineering-team-chose-an-app-definition/

Open Application Model


```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: podinfo
spec:
  components:
    - type: webservice
      name: podinfo
      properties:
        image: stefanprodan/podinfo
      traits:
        - type: scaler
          properties:
            replicas: 3
        - type: gateway
          properties:
            class: traefik
            http:
              /: 9898
```



- Consistent Model for Application Delivery.

Open App Model | KubeVela

The application delivery model behind KubeVela is Open Application Model(OAM).

How could the model adapt to so many complex application delivery scenarios?

The model are fully extensible


```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: podinfo
spec:
  components:
    - type: webservice
      name: podinfo
      properties:
        image: stefanprodan/podinfo
      traits:
        - type: scaler
          properties:
            replicas: 3
        - type: gateway
          properties:
            class: traefik
            http:
              /: 9898
```

Component Definition

CUE

```
webservice: {
    annotations: {}
    attributes: workload: definition: {
        apiVersion: "apps/v1"
        kind:       "Deployment"
    }
    type: "component"
}

template: {
    parameter: {
      image: string
    }
    output: {
        apiVersion: "apps/v1"
        kind:       "Deployment"
        spec: {
            selector: matchLabels: "app.oam.dev/component": context.name
            template: {
                metadata: labels: "app.oam.dev/component": context.name
                spec: containers: [{
                    name:  context.name
                    image: parameter.image
                }]
            }
            ...
        }
    }
}
```

Trait Definition

CUE

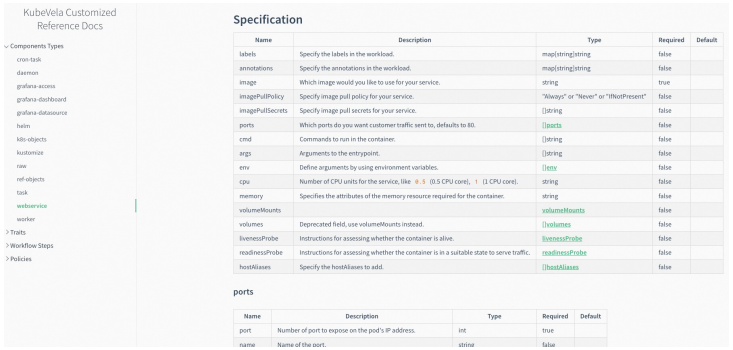
```
scaler: {
    type: "trait"
    annotations: {}
    labels: {}
    description: "Manually scale K8s pod for your workload"
    attributes: {
        appliesToWorkloads: ["deployments.apps", "statefulsets.apps"]
    }
}
template: {
    parameter: {
        // +usage=Specify the number of workload
        replicas: *1 | int
    }
    // +patchStrategy=retainKeys
    patch: spec: replicas: parameter.replicas
}
```

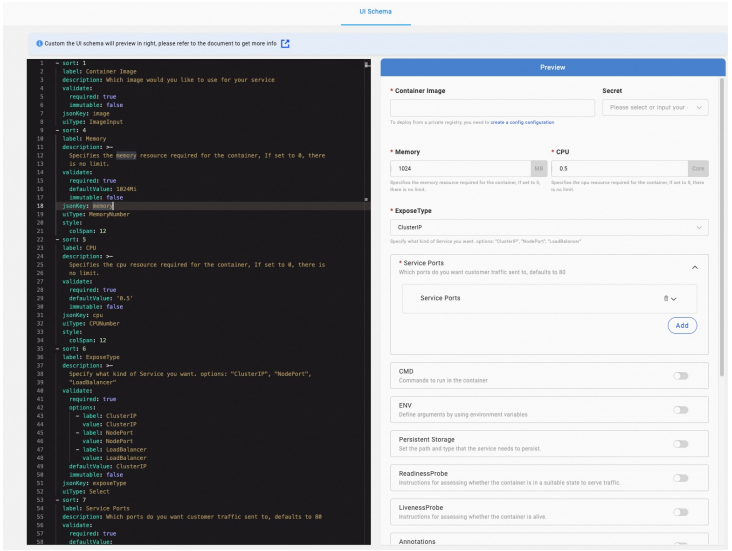
- ❑ Leverage Kubernetes API and CRD ecosystem.
- ❑ Programmable with **CUE** Configuration.
- ❑ Abstraction from Best Practices.
- ❑ Infrastructure Implementation agnostic.

How to balance the extensibility and user experience?

Balance Extension and User Experience


```
~ vela show --web
```






```
~ vela def gen-api webservice.cue
```

On the road...

https://github.com/kubevela-contrib/kubevela-go-sdk

Docs

UI

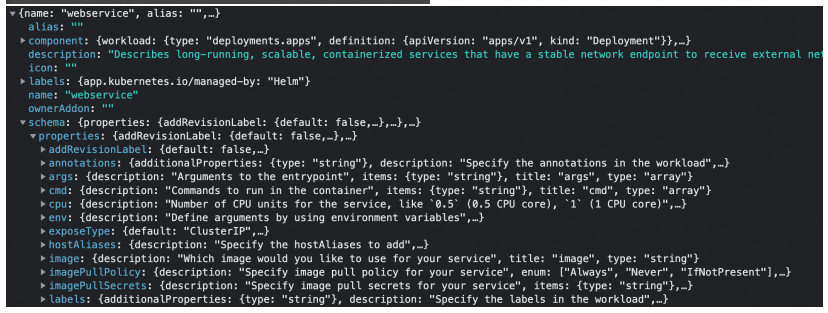
SDK

API

Observability

Automatically generated from customized definition to

webservice?type=component
/api/v1/definitions

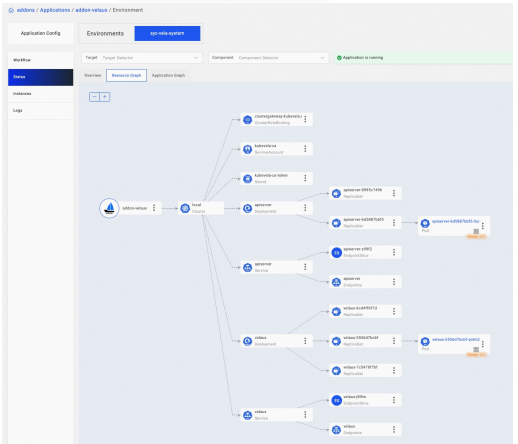


https://kubevela.stoplight.io/docs/kubevela/b8aer09sc4q9e-kube-vela-api-1-7

KubeVela

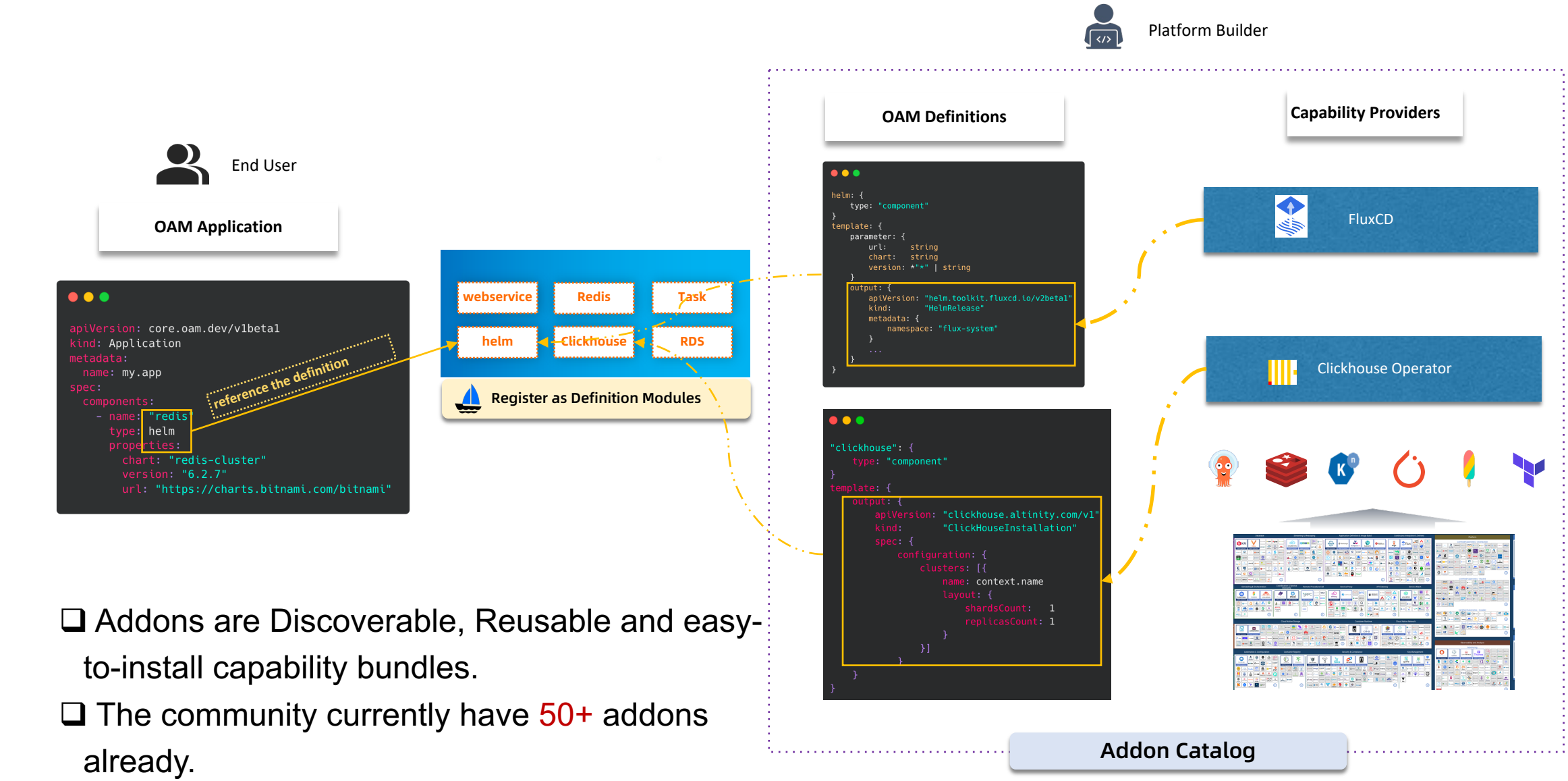
X - Definition

```
webservice: {
    annotations: {}
    attributes: workload: definition: {
        apiVersion: "apps/v1"
        kind:       "Deployment"
    }
    type: "component"
}

template: {
    parameter: {
        image: string
    }
    output: {
        apiVersion: "apps/v1"
        kind:       "Deployment"
        spec: {
            selector: matchLabels: "app.oam.dev/component": context.name
            template: {
                metadata: labels: "app.oam.dev/component": context.name
                spec: containers: [{
                    name:  context.name
                    image: parameter.image
                }]
            }
            ...
        }
    }
}
```



How to discover extension from the community?

Addon Registry: catalog for extensions



- ❑ Addons are Discoverable, Reusable and easy-to-install capability bundles.
- ❑ The community currently have 50+ addons already.

https://github.com/kubevela/catalog

KubeVela is the best practice for Platform Engineering

❑ Flexible, Extensible and Programmable.

Platform Team

Enforce best practices and deployment confidence

End Users

Enjoy PaaS-like experience

Component Types: Web Service, Cron Job, Database

Traits: Autoscale, Rollout, Route

Capability Templates: X-Definition

Staging Cluster, Production Cluster

Initialize Environments: Addons

register

Application: Web Service, Database, Route

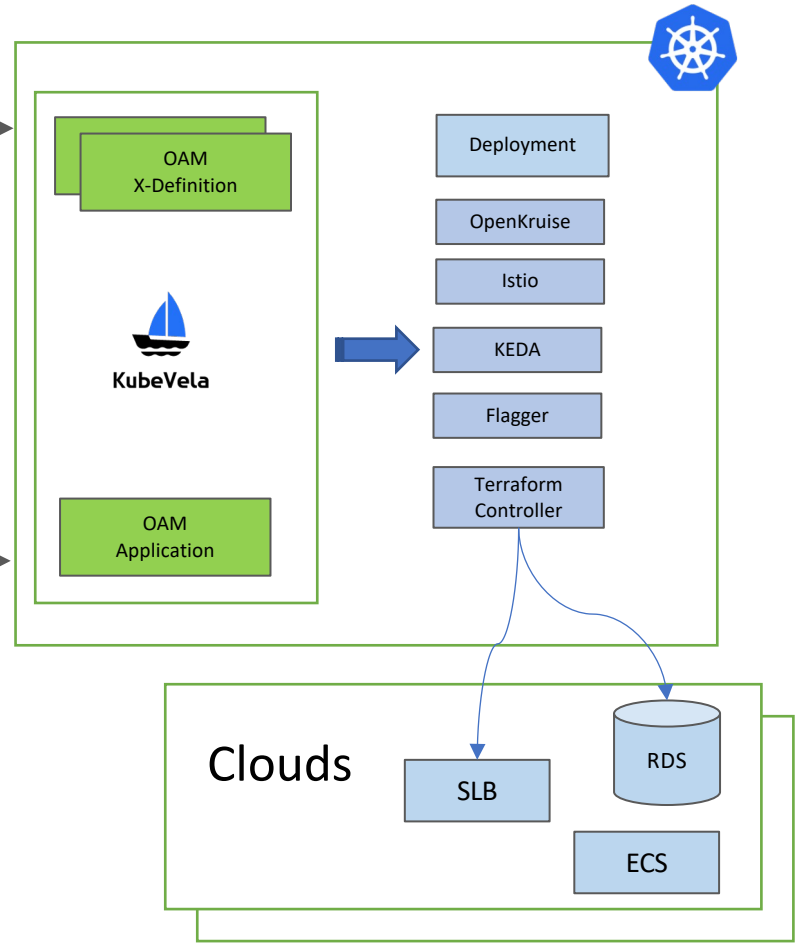
Production Cluster

deploy

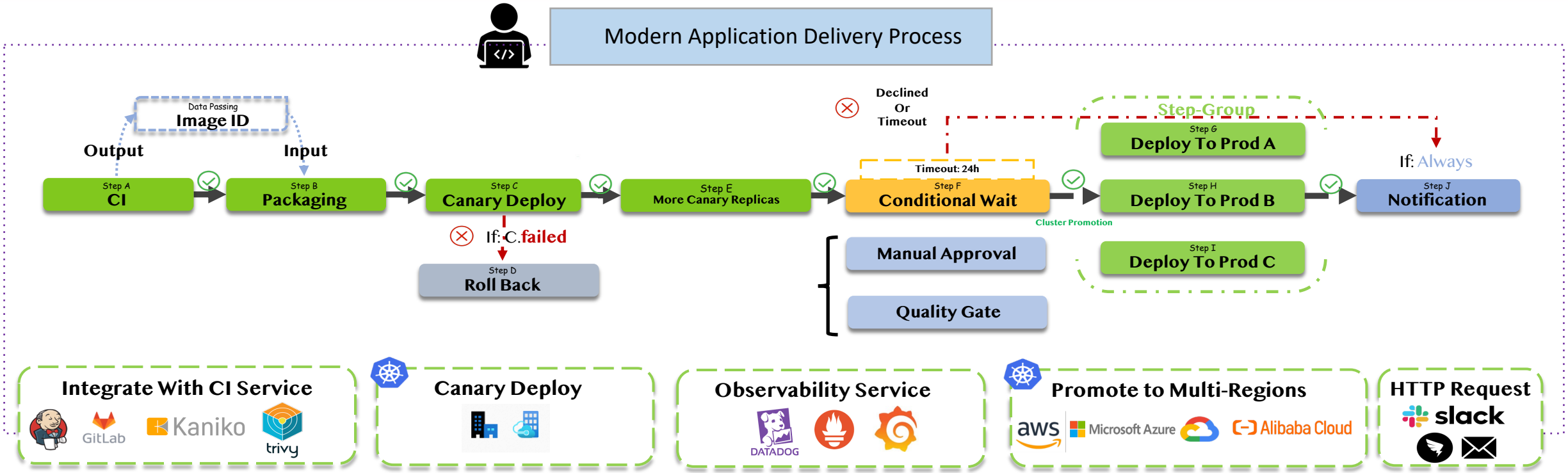
Choose target *Environment*

Choose capability *Templates*, fill in values, assemble them into an app

e.g.
image=app
replicas=2
host=myapp.io



Why KubeVela

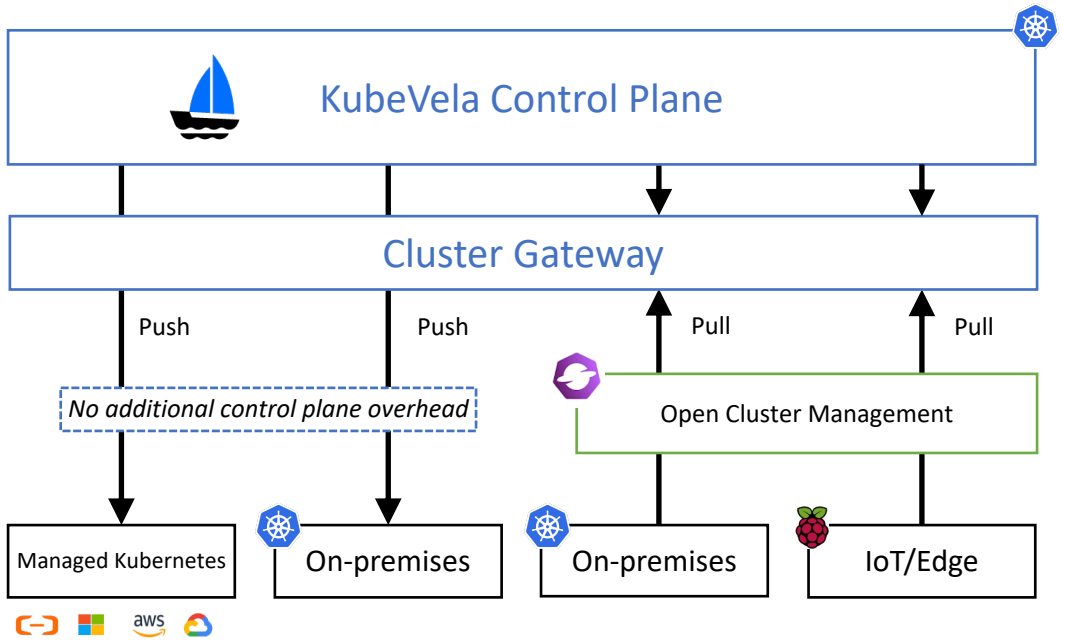


- ❑ The modern app delivery contains **not only containers but also integration of cloud resources and SaaS APIs**.
- ❑ The modern app delivery contains **promotion for multi-clusters and hybrid clouds**.
- ❑ The modern app delivery needs **observability, notifications and more customized service integration**.

How to orchestrate and manage them in a united way?

Multi-cluster as first class citizen in KubeVela


```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: deploy-with-override
  namespace: examples
spec:
  components:
    - name: nginx-with-override
      type: webservice
      properties:
        image: nginx
  policies:
    - name: topology-hangzhou-clusters
      type: topology
      properties:
        clusterLabelSelector:
          region: hangzhou
    - name: topology-local
      type: topology
      properties:
        clusters: ["local"]
        namespace: dev
    - name: override-high-availability
      type: override
      properties:
        components:
          - type: webservice
            traits:
              - type: scaler
                properties:
                  replicas: 3
```



- ❑ Natively supports multi-clusters with rich placement strategy.
- ❑ Support both Push and Pull model for cluster management.
- ❑ Runtime agnostic, adopts any plugins and manage them only in the control plane.

Customize delivery process with programmable workflow

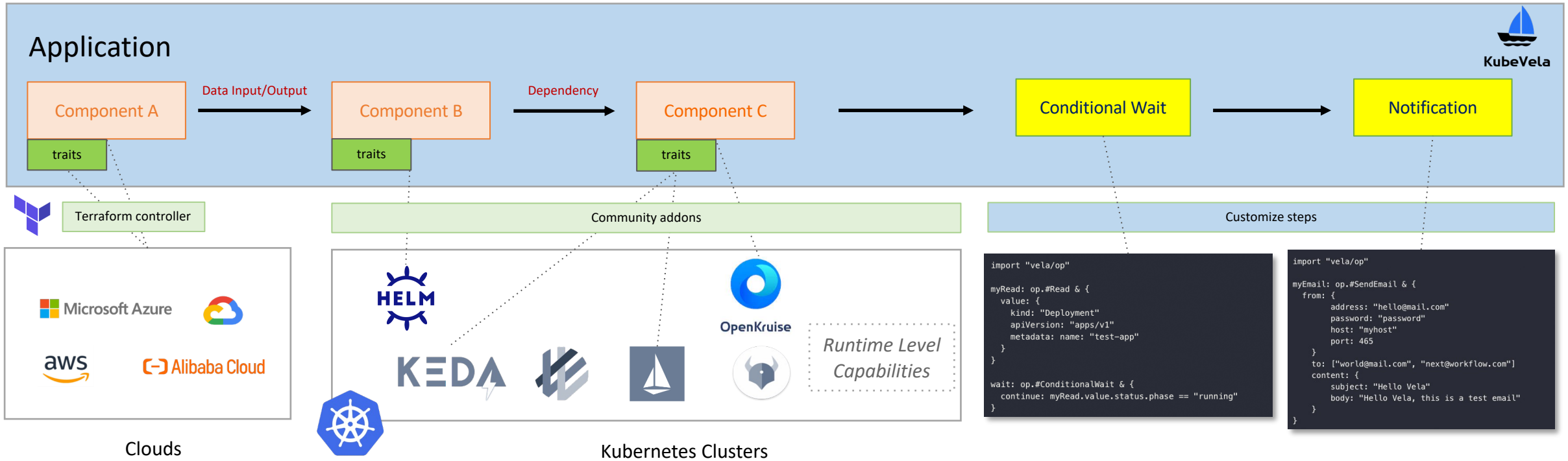

```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: my-blog
spec:
  components:
    - type: webservice
      name: my-wordpress
      properties:
        image: wordpress
    - type: alibaba-rds
      name: my-db
      properties:
        databases:
          - name: wordpress
  workflow:
    steps:
      - type: apply-component
        name: apply-db
        properties:
          component: my-db
      - type: apply-component
        name: apply-wordpress
        properties:
          component: my-wordpress
      - type: notification
        name: send-slack-message
        properties:
          slack:
            message:
              text: "deploy succeed"
```

CUE

```
import (
    "vela/op"
)

"apply-component": {
    type: "workflow-step"
    description: "Apply component to cluster."
}
template: {
  import ("vela/op")
  parameter: {
     component: string
  }

  // load component from application
  load: op.#Load

  // apply workload to kubernetes cluster
  apply: op.#ApplyComponent & {
      value: load.value[parameter.component]
  }

  // wait until workload.status equal "Running"
  wait: op.#ConditionalWait & {
     continue: apply.output.status.phase =="Running"
  }
}
```

- Rich Capabilities & Process Control
- Schema Checksum & Secure Execution
- Fast & Lightweight
- Extensible, Programmable, Reusable

A unified, powerful app delivery control plane



The Application Delivery behind KubeVela a **consistent, programmable, declarative** workflow!

Speaker

KubeCon | CloudNativeCon
Europe 2023

Da Yin

Senior Engineer, KubeVela Maintainer

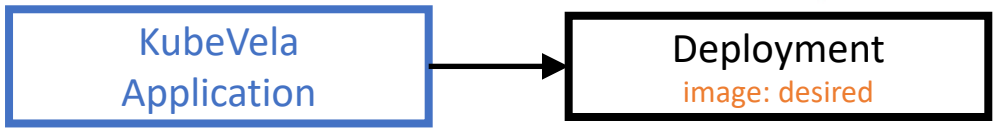
Alibaba Cloud

Why KubeVela?

KubeVela provides a unified Day-2 application management capability for all its extensibility.

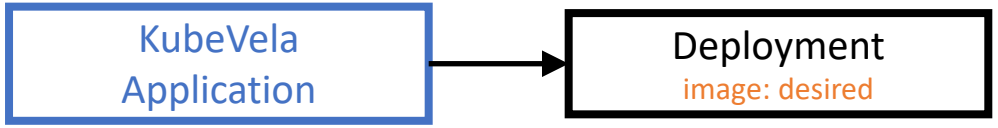
- Configuration Drift Prevention
- Resource Sharing
- Customized Observability
- Automated Garbage Collection
- Version Control
- Stability & Scalability

Resource Management



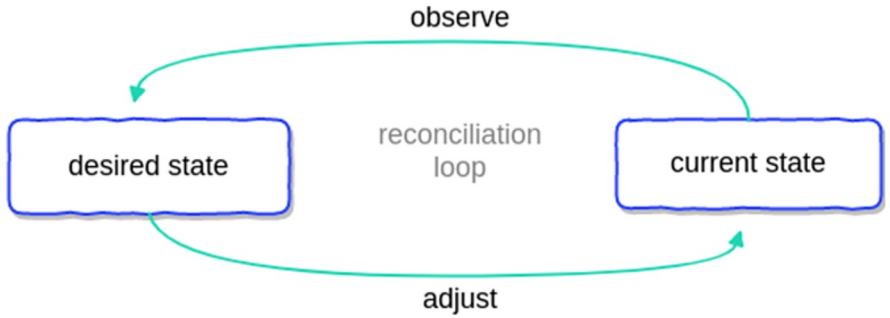
Deployment image edited by anonymous.

Application recovers the deployment to desired.



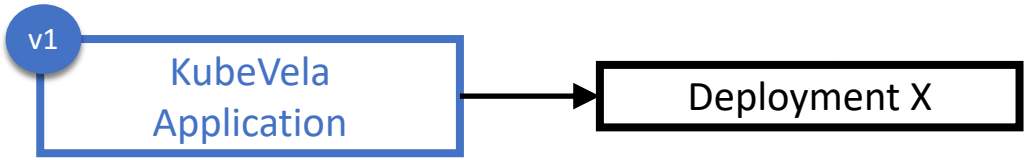
❑ No configuration drift.

The KubeVela Application repeatedly checks if managed resources are always in accord with the spec declared during the delivery process. It can effectively prevent configuration drift.

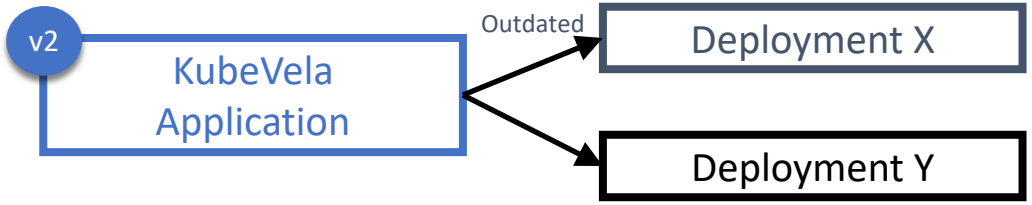


Resource Management

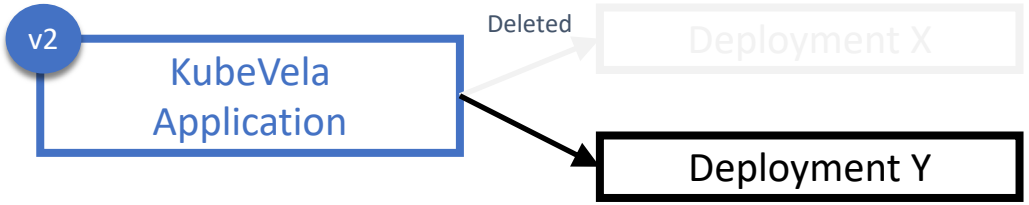
❑ Automated garbage collection.



Application Upgraded.

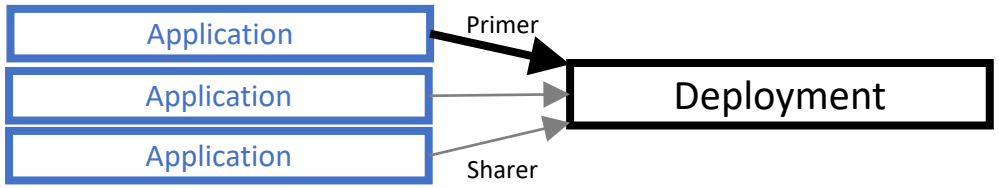


Application recycles outdated resources.

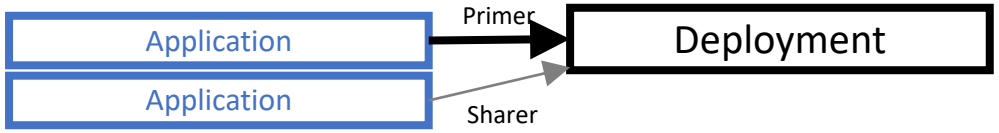


The KubeVela Application recycles resources when the application itself is updated or deleted. Users can configure various strategy for outdated resources, such as keeping them or removing them.

❑ Resources sharing across applications.



Primer application deleted.



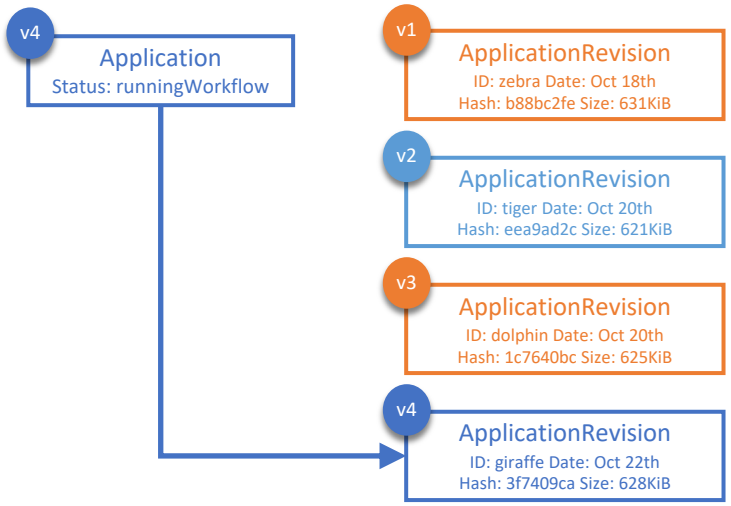
Last application deleted.

Resources can be shared by multiple applications. Shared resources are editable by the primer application and readable by all sharers. The last exit application is responsible for recycling them.

Application Version Control

KubeCon CloudNativeCon Europe 2023

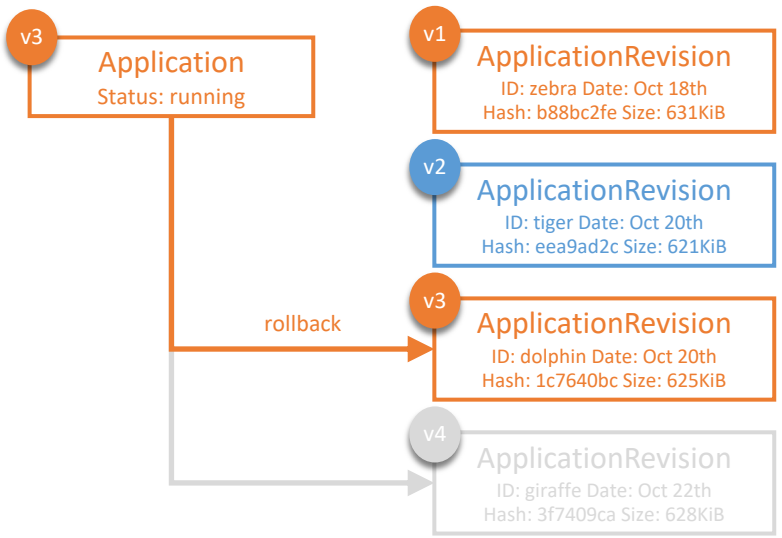
❑ History version recorded.




```
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: podinfo
  annotations:
-   app.oam.dev/publishVersion: v3
+   app.oam.dev/publishVersion: v4
    ...
```

Each KubeVela Application keeps limited history versions. Each version is a snapshot for the past delivery. Both the application and related definitions are recorded.

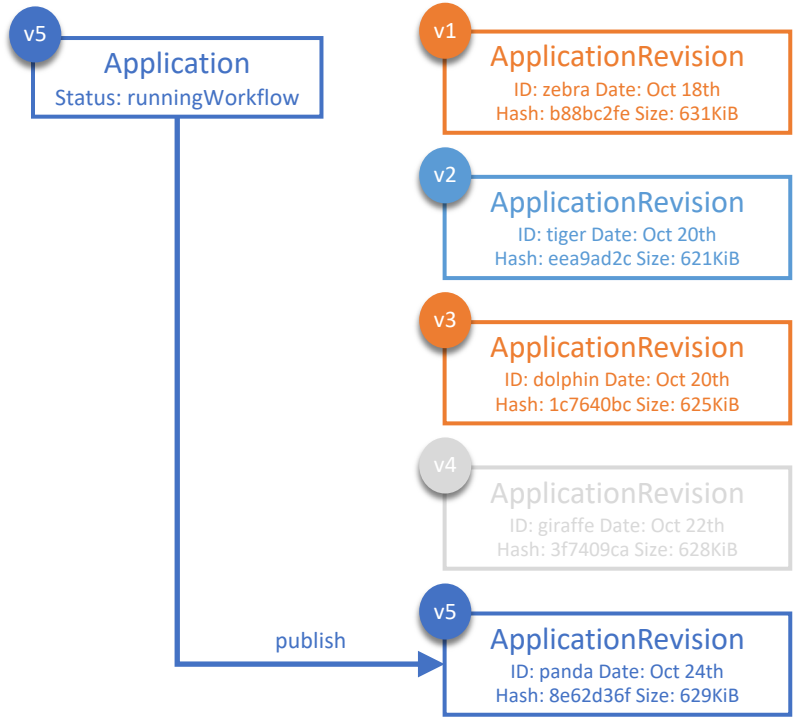
❑ Rollback to succeeded version.




```
$ vela live-diff my-app
    - type: webservice
      name: webapp
      properties:
-       image: webapp:dolphin
+       image: webapp:panda
```

The KubeVela Application supports rolling back to history succeeded versions when new publish failed. Inspecting differences across versions is available as well.

❑ Inspect changes across versions.

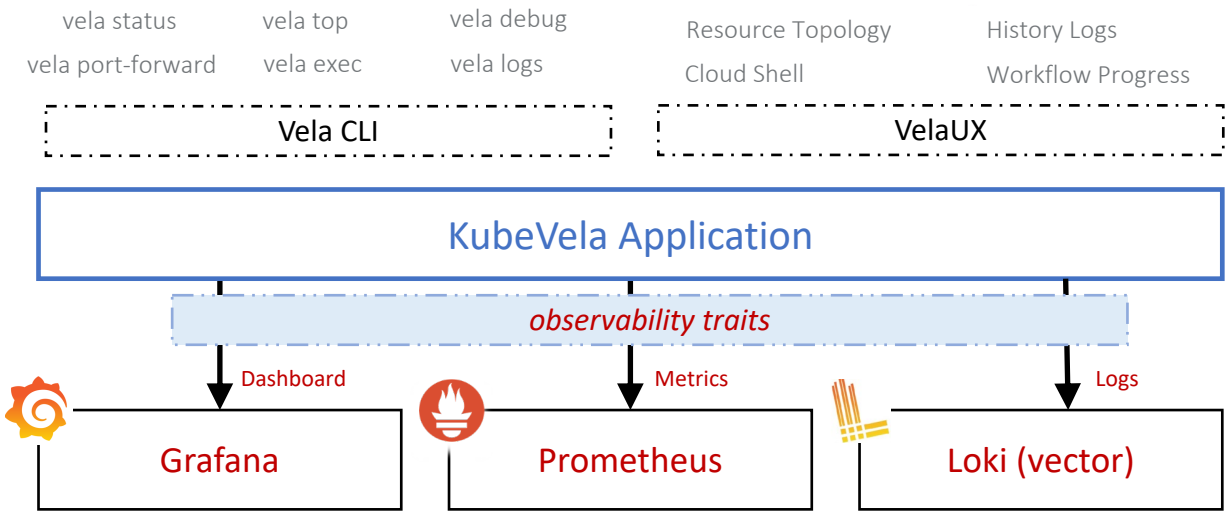


While KubeVela application usually automatically publishes new versions on spec changes, it is also possible to manually control the version publish, which allows users to edit application first and commit changes later.

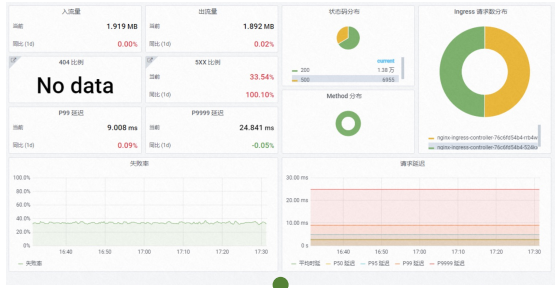
Observability as first class citizen

KubeCon CloudNativeCon Europe 2023

❑ Observability for all extended resources.




```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: nginx-app-2
spec:
  components:
    - name: nginx-comp
      type: webservice
      properties:
        image: nginx:1.14.2
        ports:
          - port: 80
            expose: true
      traits:
        - type: stdout-logs-collector
          properties:
            parser: nginx
            redirect_unknown_logs: true
            output_field:  "parsed"
            output_type:  "loki"
            loki_endpoint:  http://my-loki-svc:3100/
            loki_user_labels:
                job: "ingress_logs"
                env: "prod"
```



❑ Application Centric Observability.

With the use of customized Components and Traits, users can define how to monitor applications, for example, the way logs are collected and the dashboards metrics are plotted on.

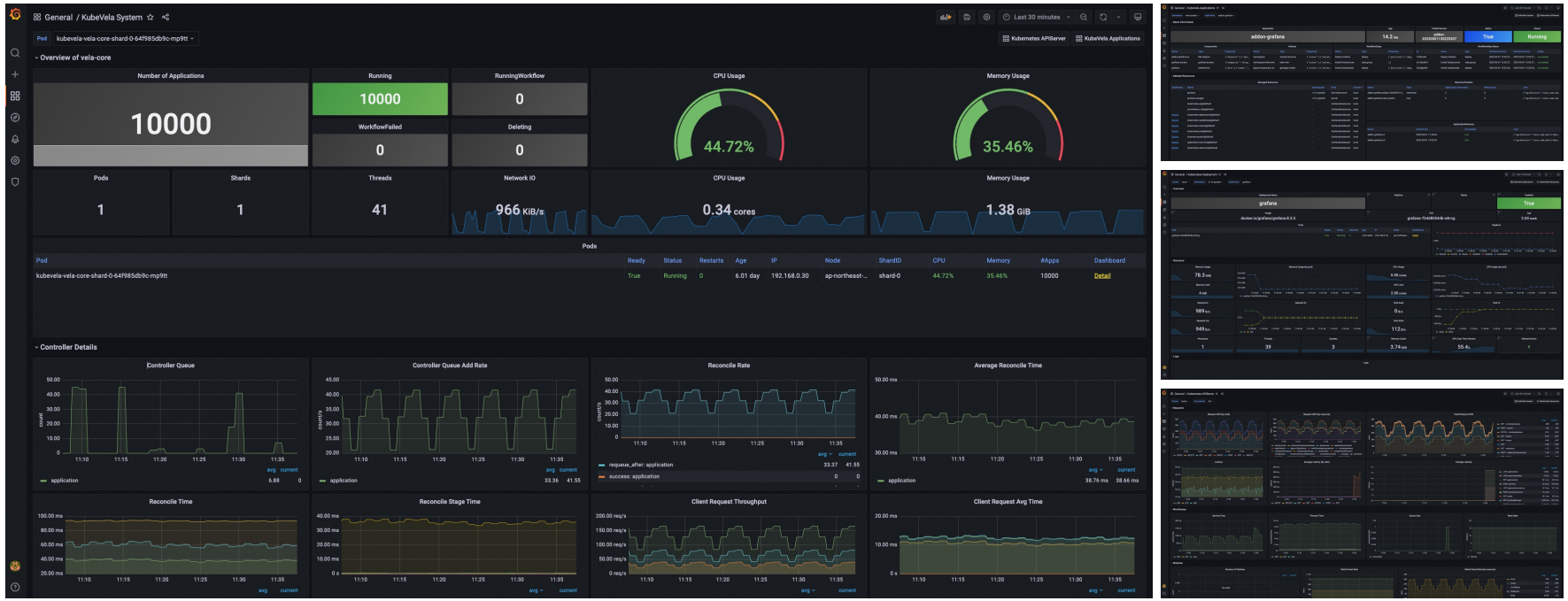
❑ Observability as Code.

The Observability rules for applications are managed in declarative ways. It makes updates and migrations more convenient and controllable. Developers can leverage the power of underlying monitoring infrastructures without the need of learning varying complex syntax.

Stability

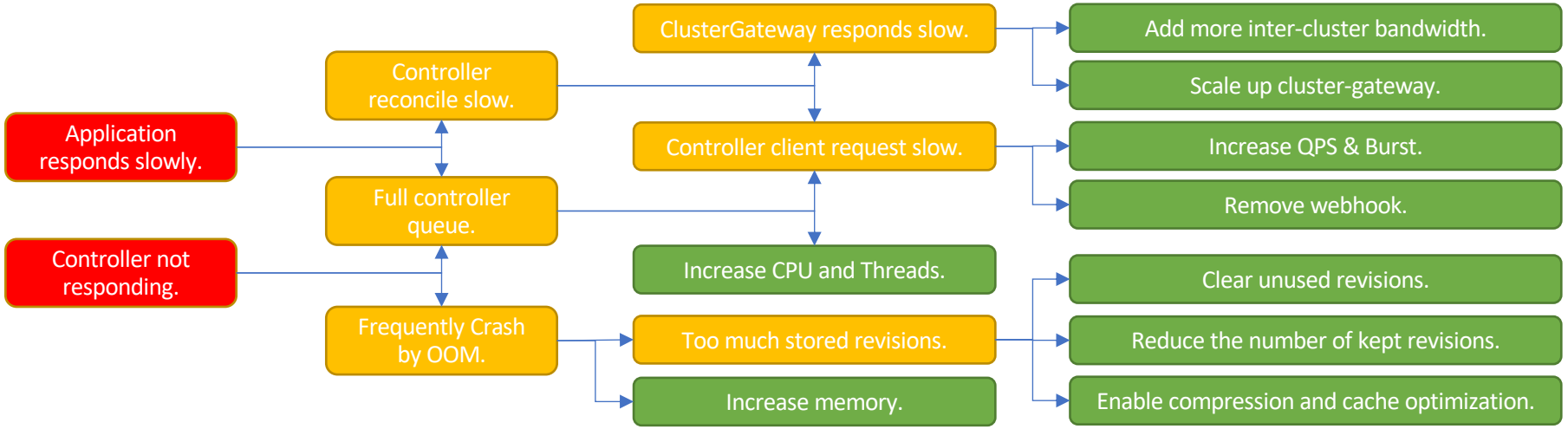
System Monitoring

The observability infrastructures include the necessary tools for monitoring the health status of KubeVela control plane, such as the controller's throughput, the latency of apiserver, etc. Exceptions and performance bottlenecks will be exposed by these metrics and dashboards.

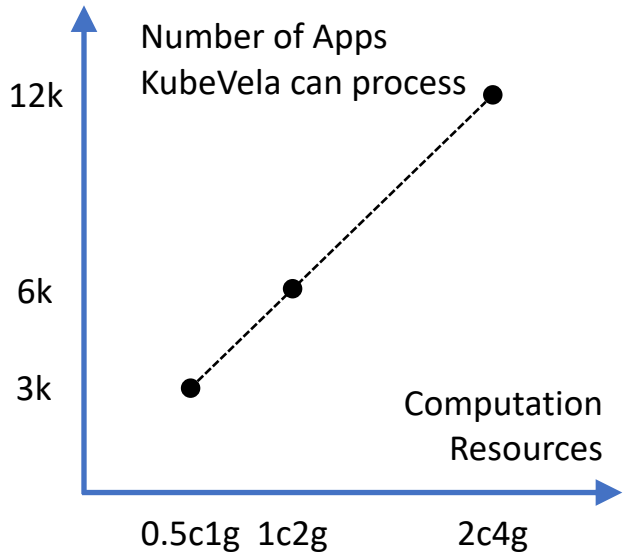


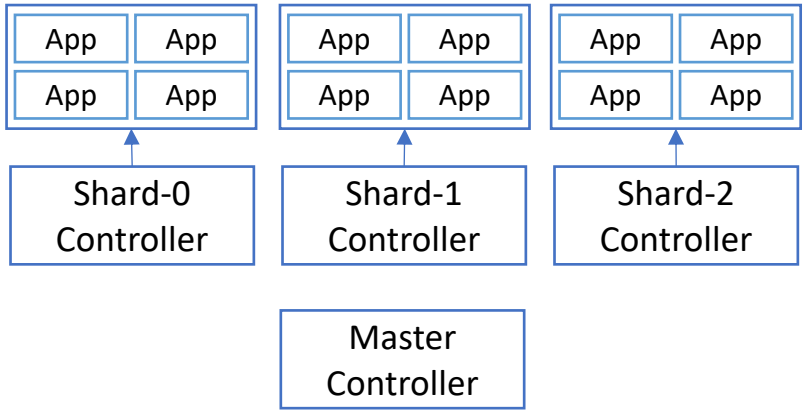
Customized Tunning

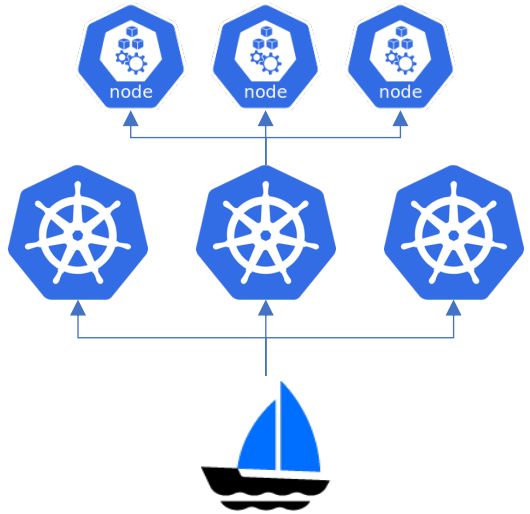
Depending on the states displayed by system monitoring tools, operators are able to make proper tunning strategy to the system and improve the robustness of the system.



Scalability







Vertical Scaling

Load tests have demonstrated that KubeVela is able to process thousands of applications with limited resources. The system capacity is linearly correlated with given computation resources.

Horizontal Scaling

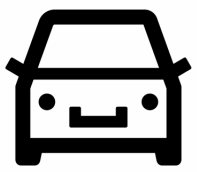
KubeVela controller supports sharding. By adding more shards of controllers to the control plane, it is easy to scale out the application capacity of the system. It can also help provide soft isolation to the runtime and therefore reduce the affected zones of disasters.

Large Scale

The application capacity of KubeVela system is nearly irrelevant to the number of Kubernetes clusters and nodes it manages. With proper tuning, it is shown to be possible to manage 200k+ application across hundreds of clusters in one KubeVela control plane.

Adopters

Commercial Banks



Car Manufacturers

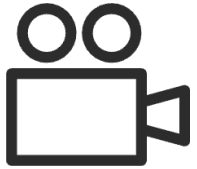
Cloud Providers

Game Companies

Insurance Cooperation

Software Developers

Trading Platforms



Entertainment Industry

KubeVela is applied across various areas to help manage application systems, especially high-tech industries.

https://github.com/kubevela/community/blob/main/ADOPTERS.md

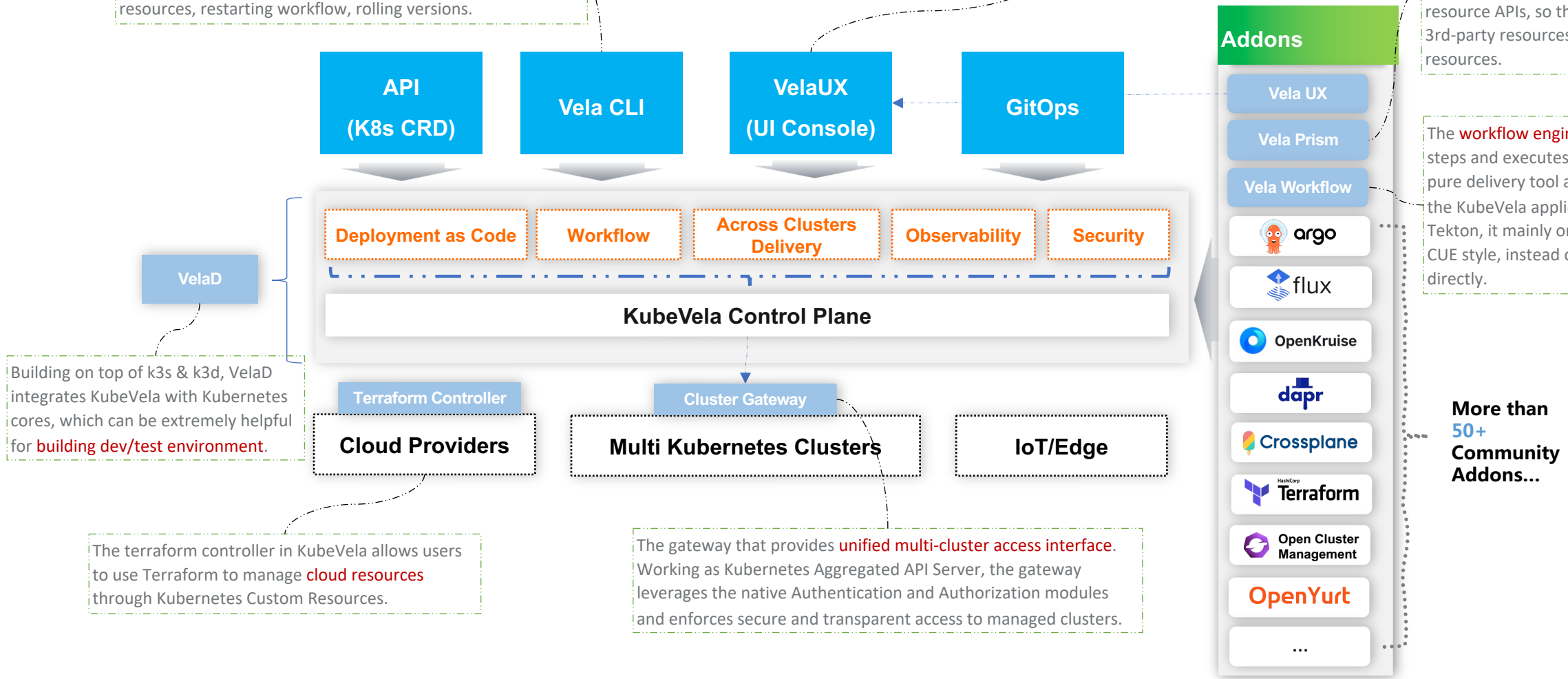
KubeVela Eco-system: Tools

KubeCon CloudNativeCon Europe 2023

KubeVela CLI provides various commands that helps you to operate applications, such as managing definitions, viewing resources, restarting workflow, rolling versions.

VelaUX is the Web UI for KubeVela. Besides, it incorporates business logics into fundamental APIs and provides out-of-box user experiences for non-k8s-expert users.

The extension API server for KubeVela built upon the Kubernetes Aggregated API Server. It projects native APIs like creating dashboards on Grafana into Kubernetes resource APIs, so that users can manage 3rd-party resources as Kubernetes native resources.



The workflow engine translates CUE-based steps and executes them. It works as a pure delivery tool and can be used aside by the KubeVela application. Compared to Tekton, it mainly organize the process in CUE style, instead of using Pods and Jobs directly.

Building on top of k3s & k3d, VelaD integrates KubeVela with Kubernetes cores, which can be extremely helpful for building dev/test environment.

More than 50+ Community Addons...

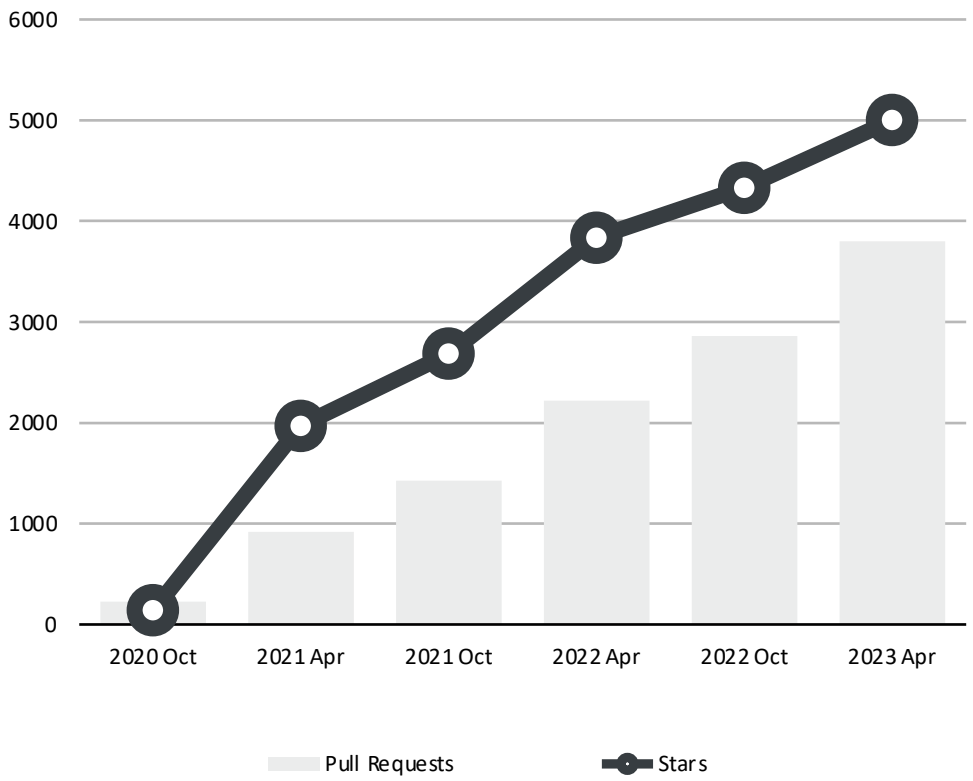
The terraform controller in KubeVela allows users to use Terraform to manage cloud resources through Kubernetes Custom Resources.

The gateway that provides unified multi-cluster access interface. Working as Kubernetes Aggregated API Server, the gateway leverages the native Authentication and Authorization modules and enforces secure and transparent access to managed clusters.

https://github.com/kubevela/

KubeVela Community

KubeVela attracts world-wide contributors and continuously evolves.



Contributors

KubeVela has attracted over 300 contributors from various countries, including China, USA, India, Germany, Korea, Spain, etc.

Issues

KubeVela received over 1,600 issues and has solved 80% of them.

Biweekly Community Meetings

KubeVela holds bi-weekly community meetings and has recorded 40+ English meetings on YouTube.

https://github.com/kubevela/community