KUBERNETES COMMUNITY DAYS SHENZHEN 2023

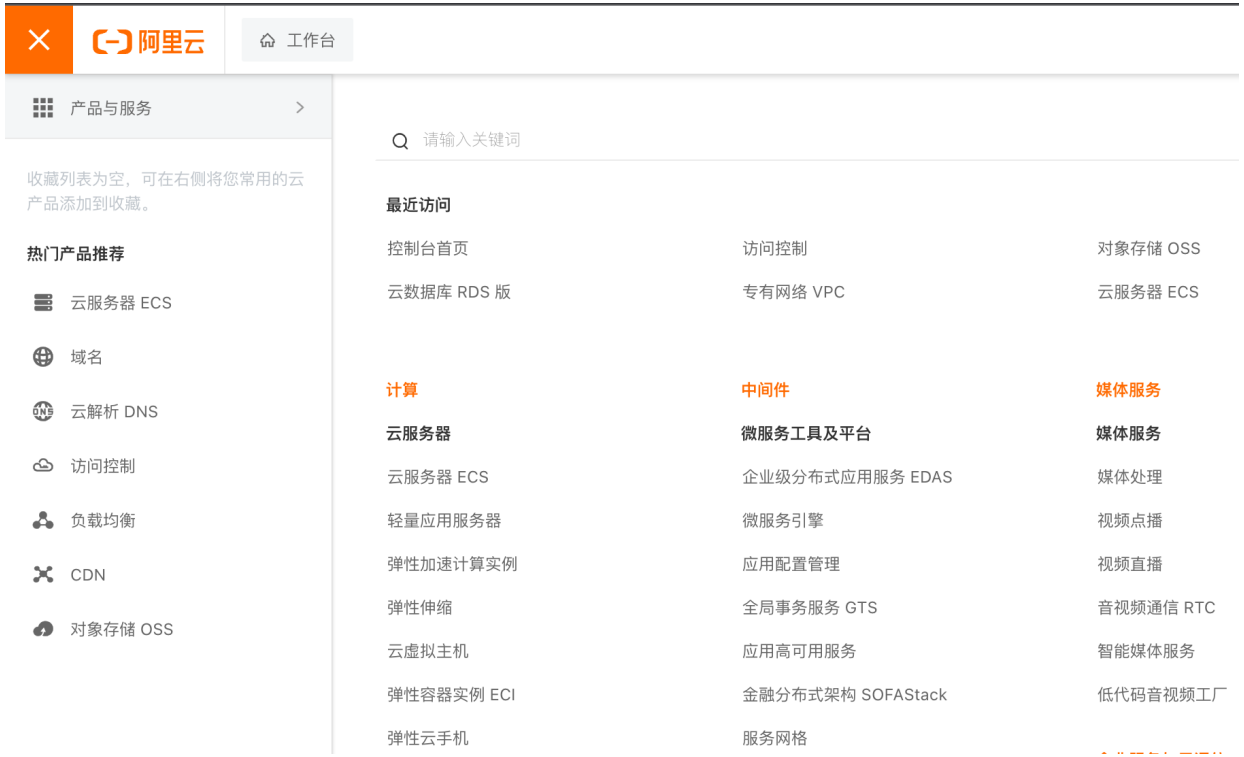
KusionStack + KCL
打造云原生时代开发者的自服务平台

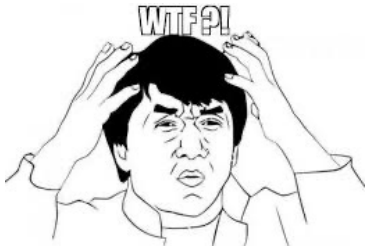
李大元、宗喆
蚂蚁集团

Content

目录

云原生时代的运维挑战

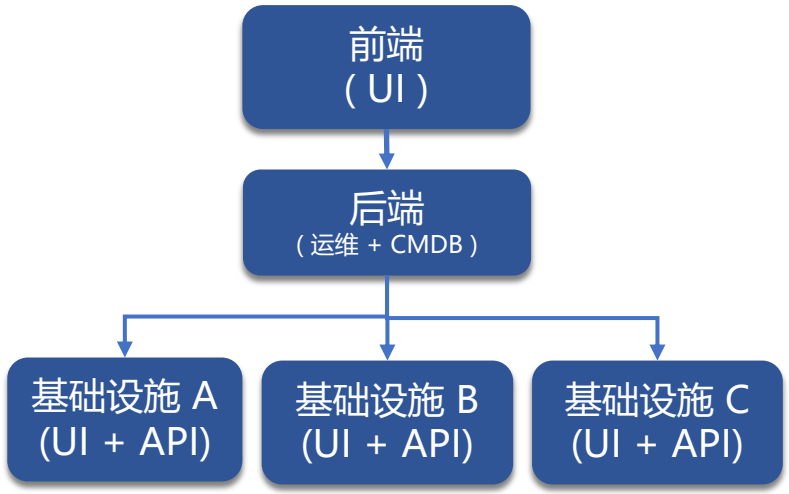


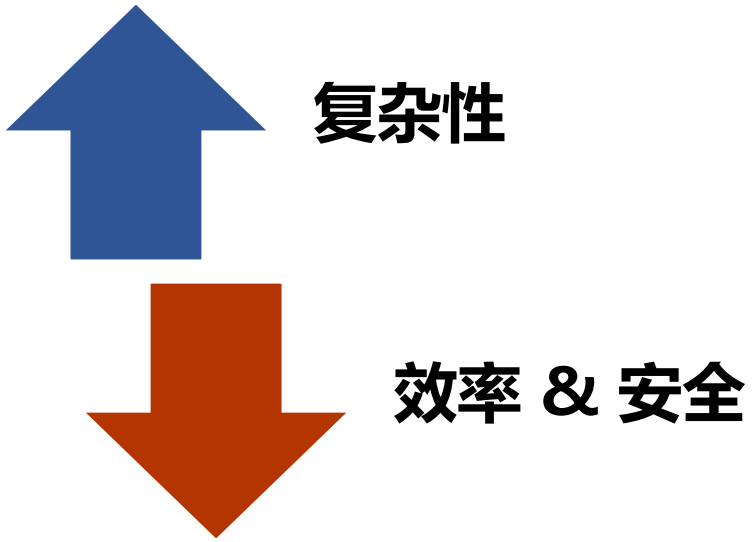


Source: David Bell

一次发布需要学习几个系统？找几个人？

传统 PaaS 的困境





Developer

- 认知负担高
- 发布运维时间变长

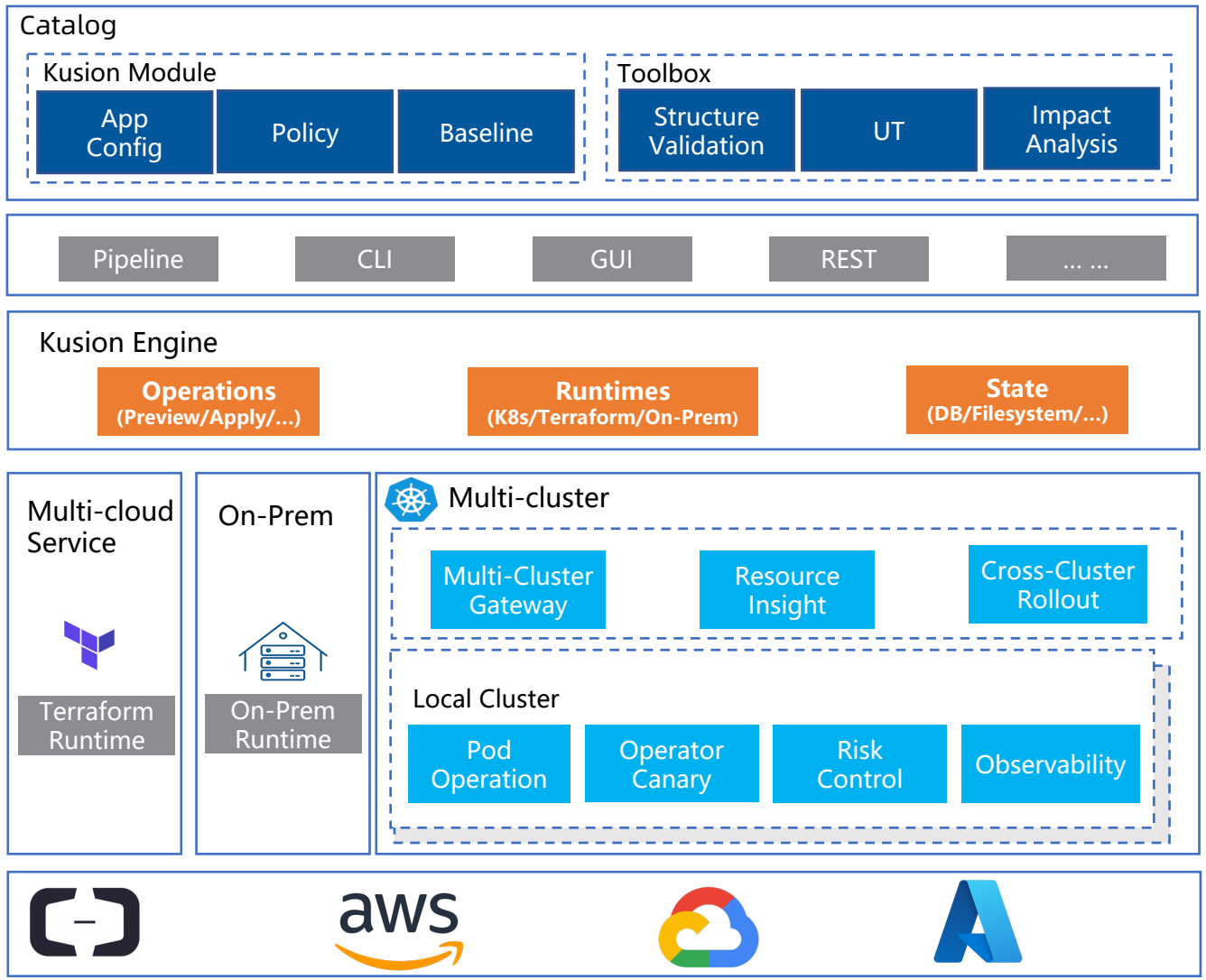
Platform

- 需求大幅增加，成为企业效能瓶颈
- 跨团队沟通成本大幅增加

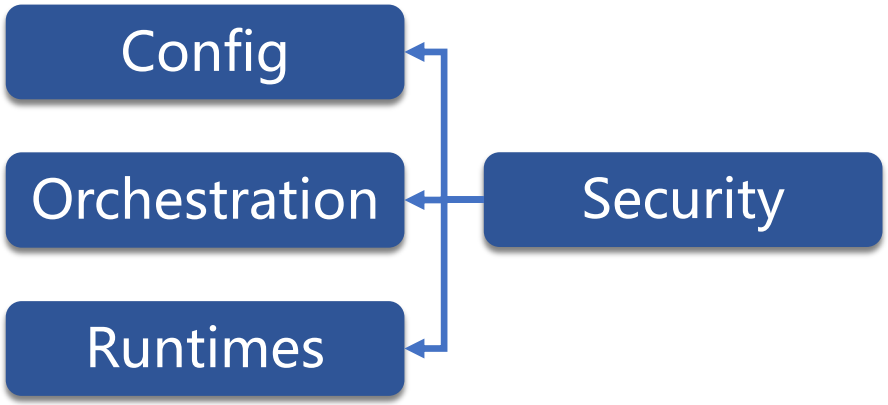
Security

- 多个独立基础设施平台，风险敞口大
- 声明式运维很强大也很危险

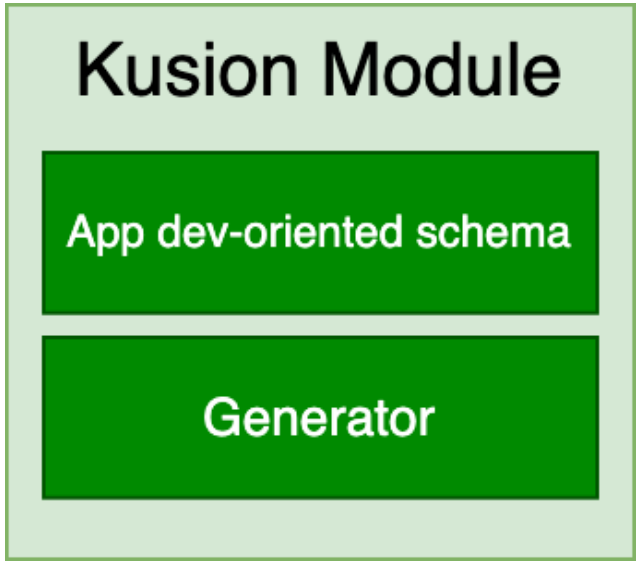
我们的实践— KusionStack 架构



助你更快、更安全的构建**自己的** Developer Platform



KusionStack——Module & Workspace




```yaml
modules:
  mysql:
    default:
      provider: aws
      size: 20
      instanceClass: db.t3.micro
      securityIPs:
      - 10.0.0.0/18
    smallClass:
      size: 50
      instanceClass: db.t3.small
      projectSelector:
      - foo
      - bar
    largeClass:
      instanceClass: db.t3.large
      projectSelector:
      - baz
```

Module 名称

默认配置

指定默认的 Module 配置

Override 配置

指定适用的 Project

workspace.yaml

Platform 配置

- 能力模块(资源、监控、中间件...)，实现**标准化**
- 差异化留给平台，实现**一次编写，多处发布**
- 一个环境一份配置，按需选择，实现**动态环境配置**

KusionStack——AppConfiguration

App Dev 配置

- 使用可理解、易组合的 Module 实现**自服务**
- 一份配置涵盖应用交付**全生命周期**

```
helloworld: ac.AppConfiguration {
    workload: wl.Service {
        replicas: 2
        containers: {
            "nginx": c.Container {
                image: "nginx:v1"
                command: ["/bin/sh", "-c", "echo hi"]
                env: {
                    "key": "value"
                }
                workingDir: "/tmp"
                resources: {
                    "cpu": "2"
                    "memory": "4Gi"
                }
                readinessProbe: p.Probe {
                    probeHandler: p.Http {
                        url: "http://localhost:80"
                    }
                }
            }
        }
        ports: [
            n.Port {
                port: 80
                targetPort: 8080
                exposeInternet: True
            }
        ]
    }
    pipeline: {
        "deploy": Deploy {
            manualApprove: true
            rollbackIfFailed: true
        }
    }
    dependency: {
        dependedApps: ["api-server"]
    }
}
```

Components

描述完整交付应用所需的组件，包括工作负载 (workload) 和网络端口、数据库等配件 (accessory)

nginx *container*

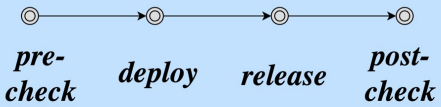
prots *database* *accessory*

Depedency

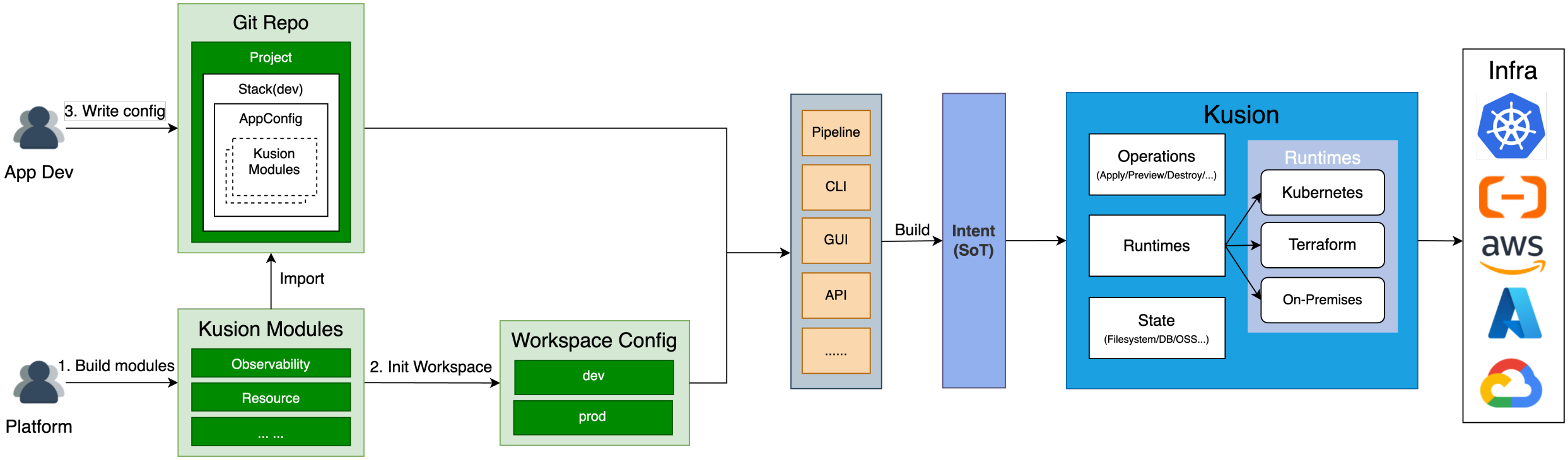
描述跨应用依赖关系

Pipeline

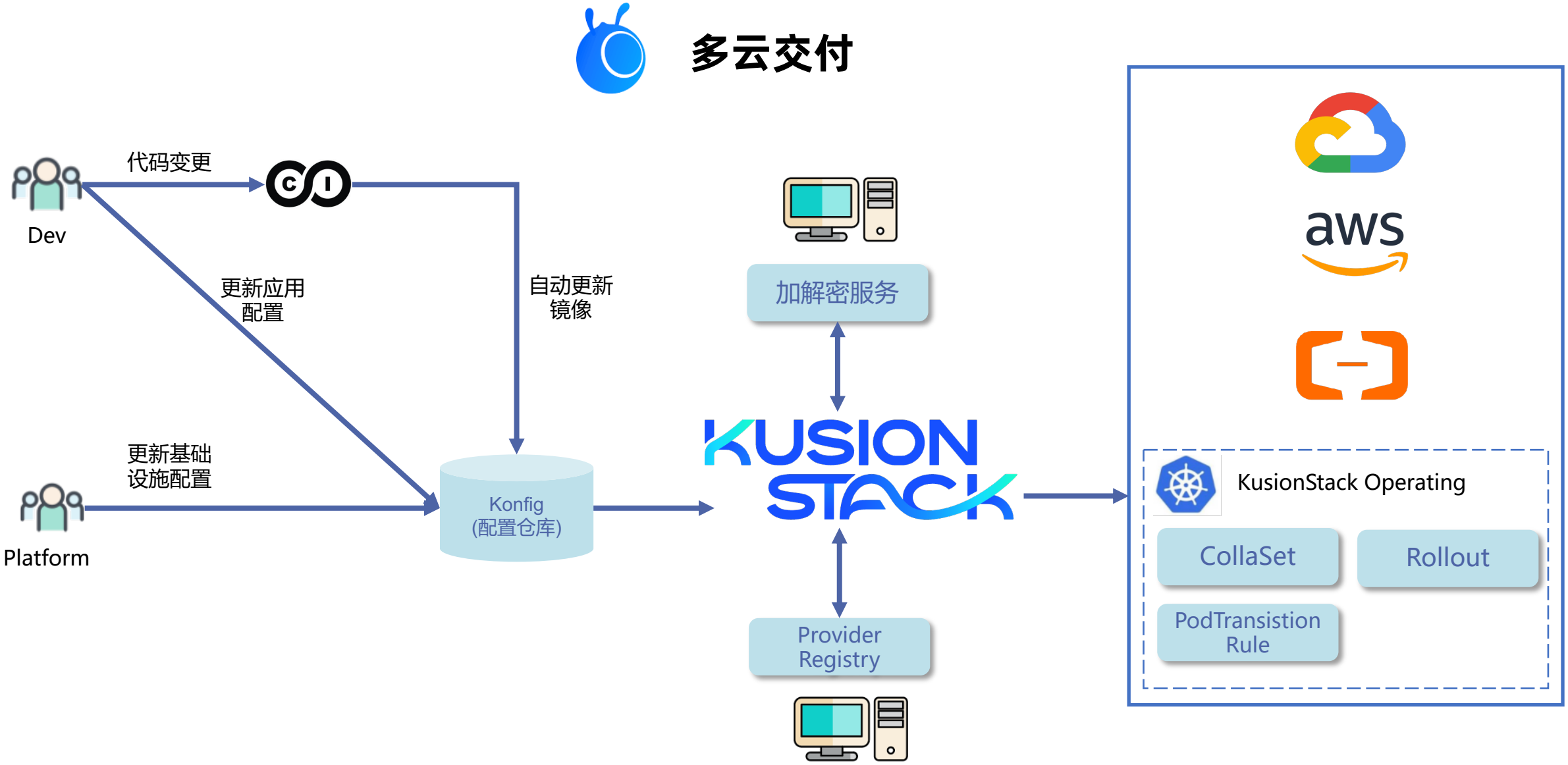
描述应用交付的流程，包括前置检查、审批、部署、后置校验等步骤



KusionStack——Workflow

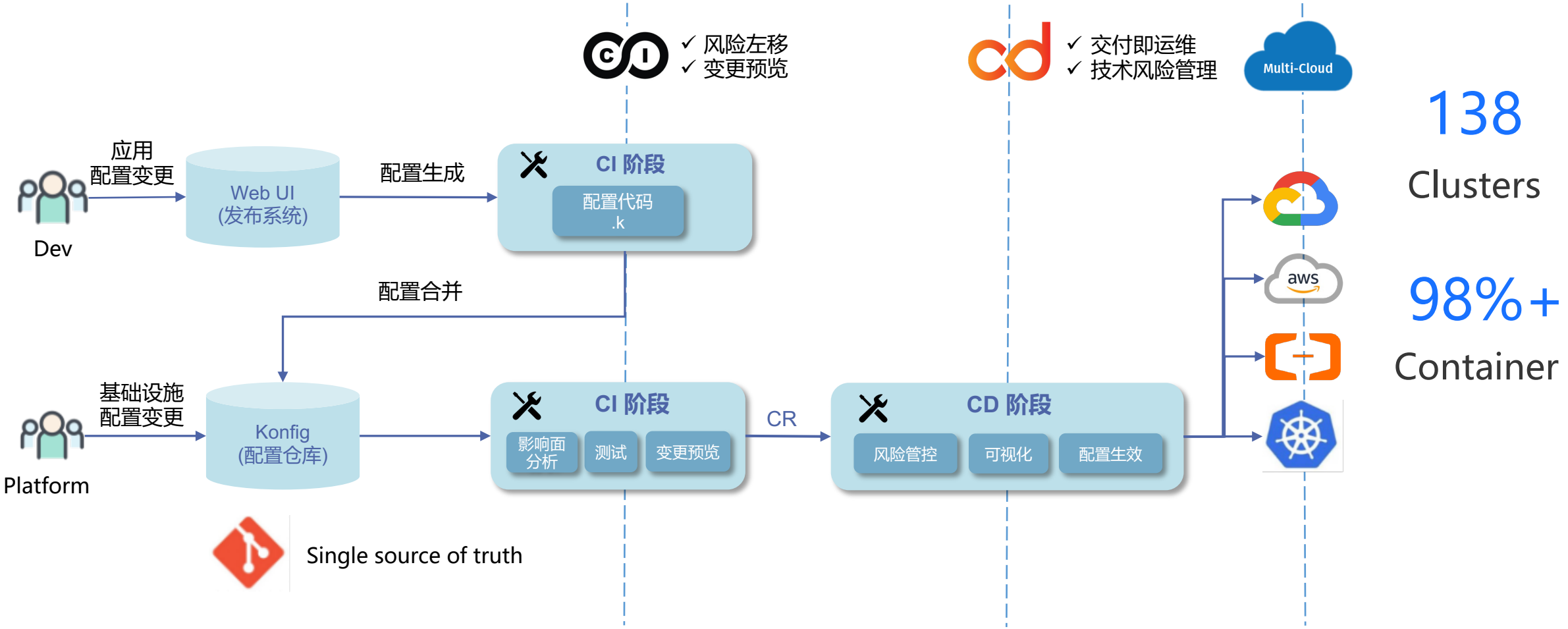


在蚂蚁和其他公司的实践



在蚂蚁和其他公司的实践

有赞



138

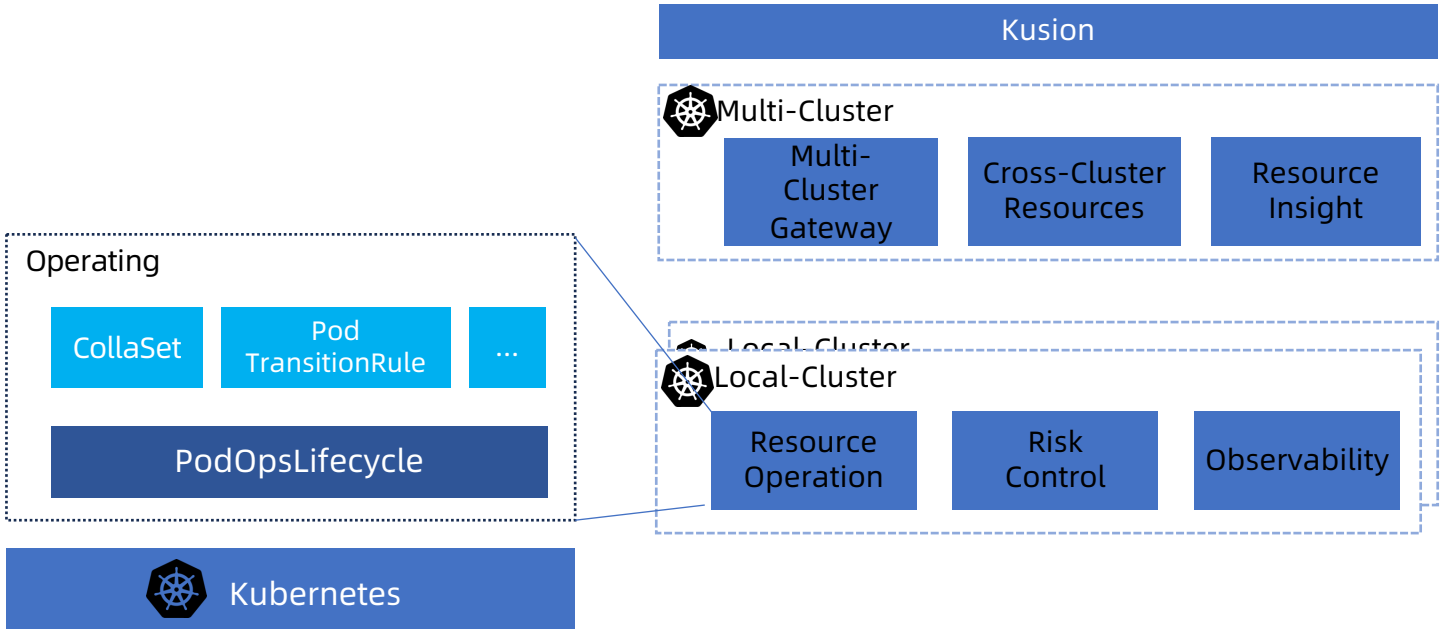
Clusters

98%+

Container

云原生资源交付与运维

https://github.com/KusionStack/operating

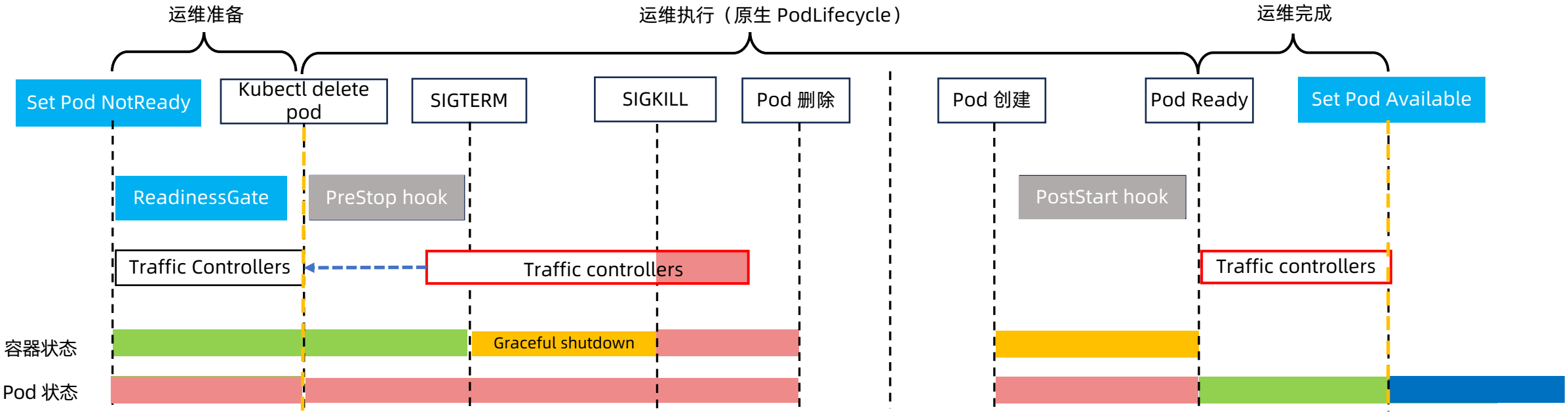



```
helloworld: ac.AppConfiguration {
    workload: wl.Service {
        replicas: 2
        containers: {
            "nginx": c.Container {
                image: "nginx:v1"
                command: ["/bin/sh", "-c", "echo hi"]
                env: {
                    "key": "value"
                }
                workingDir: "/tmp"
                resources: {
                    "cpu": "2"
                    "memory": "4Gi"
                }
                readinessProbe: p.Probe {
                    probeHandler: p.Http {
                        url: "http://localhost:80"
                    }
                }
            }
        }
        ports: [
            n.Port {
                port: 80
                targetPort: 8080
                exposeInternet: True
            }
        ]
    }
    pipeline: {
        "deploy": Deploy {
            manualApprove: true
            rollbackIfFailed: true
        }
    }
    dependency: {
        dependedApps: ["api-server"]
    }
}
```

Pod 运维生命周期

原生 Pod 变更过程 K8s 提供的交互和管控能力有限

- Pod 粒度
- 可回滚
- 可扩展



Operator 稳定性管理

https://github.com/KusionStack/controller-mesh

单点问题：

高负荷

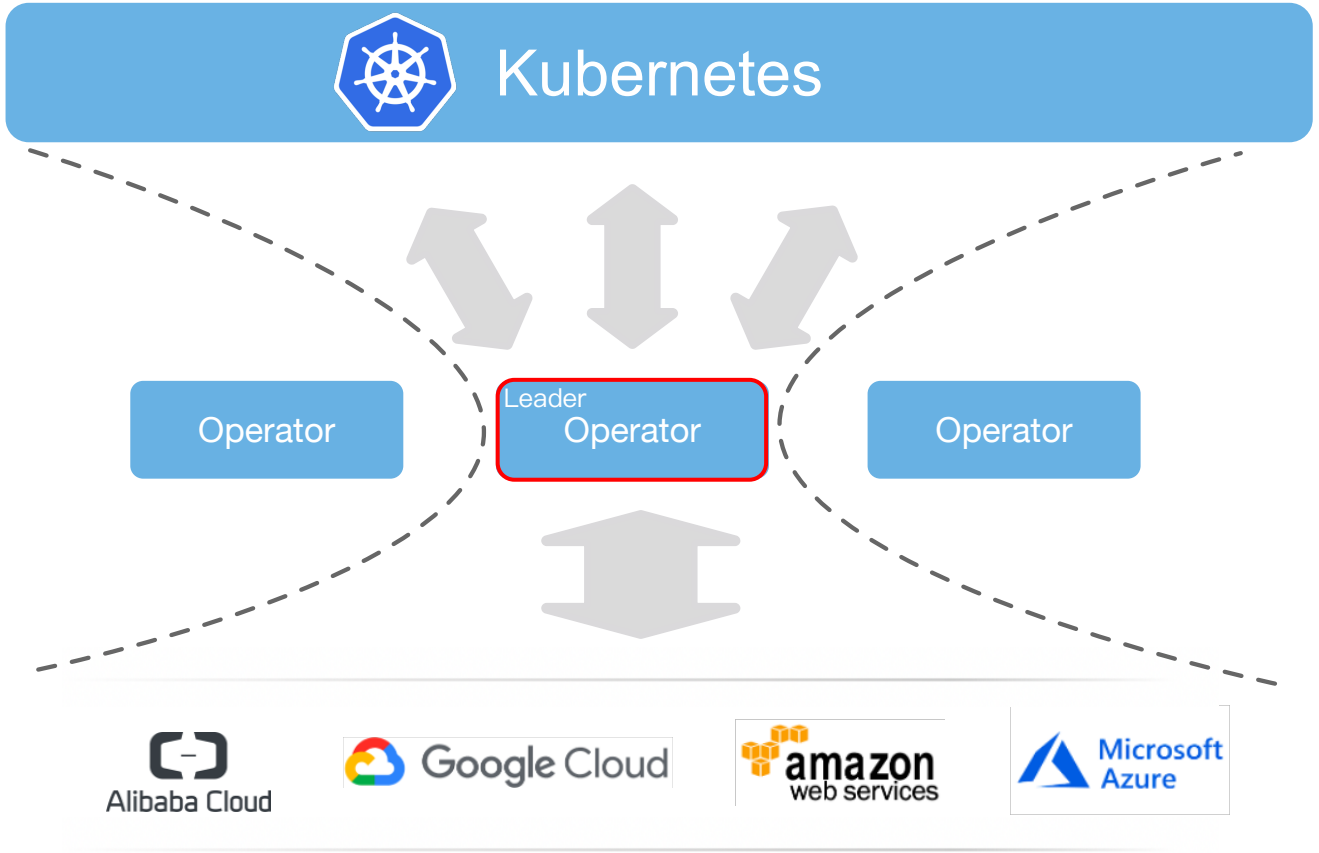
更多的 CPU & Memory
无法水平扩展
调和效率低
启动时间长

*List 700k pods -> 10min

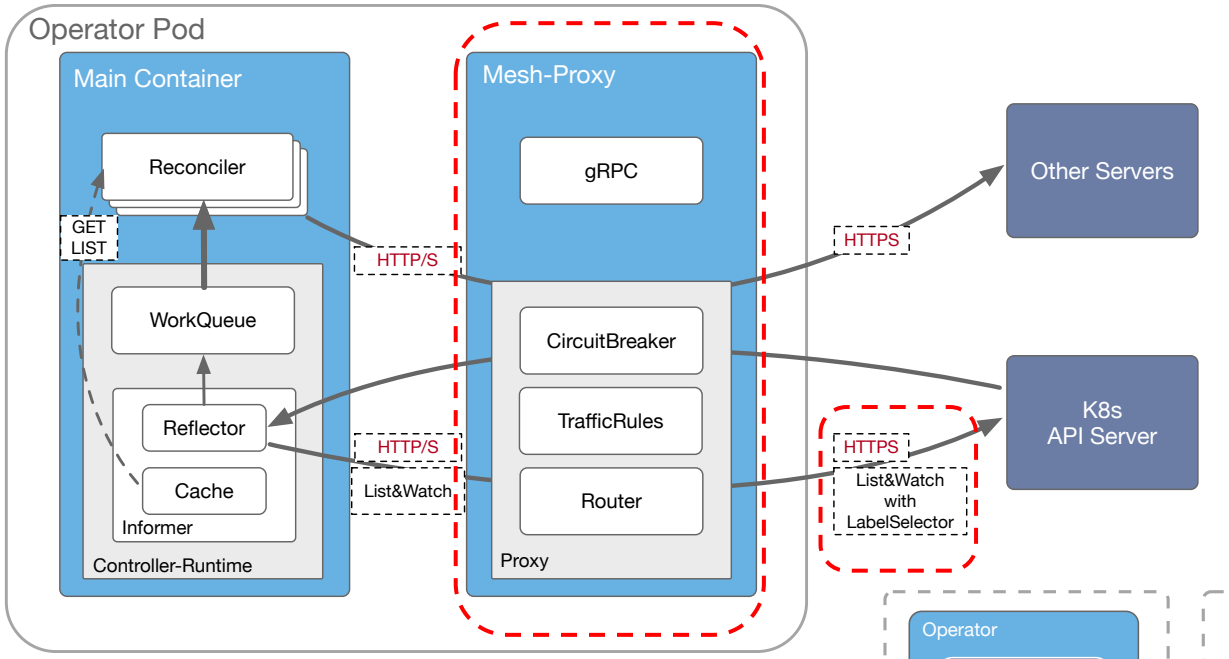
*Too old resource version

高风险

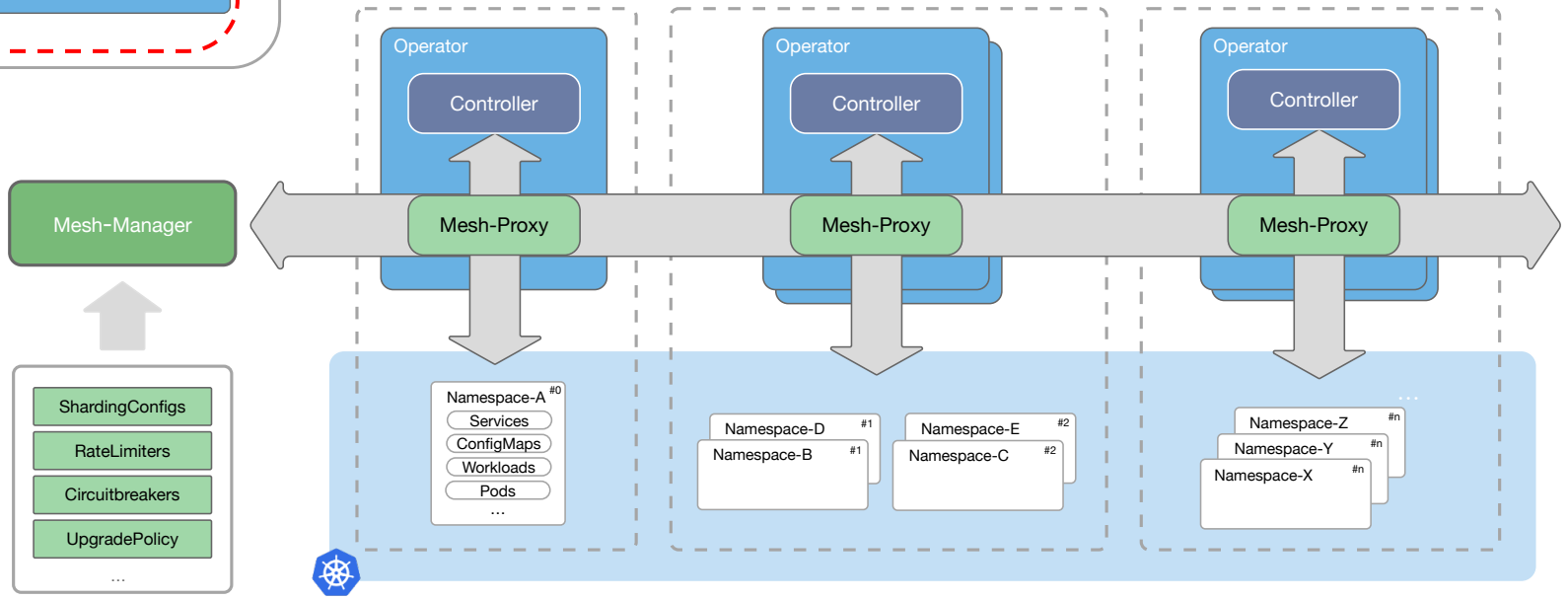
无法灰度
无法控制爆炸半径



Operator 稳定性管理 -- ControllerMesh



- 集群分片
- 水平扩展
- 隔离的灰度环境
- 灰度发布
- …



Takeaways

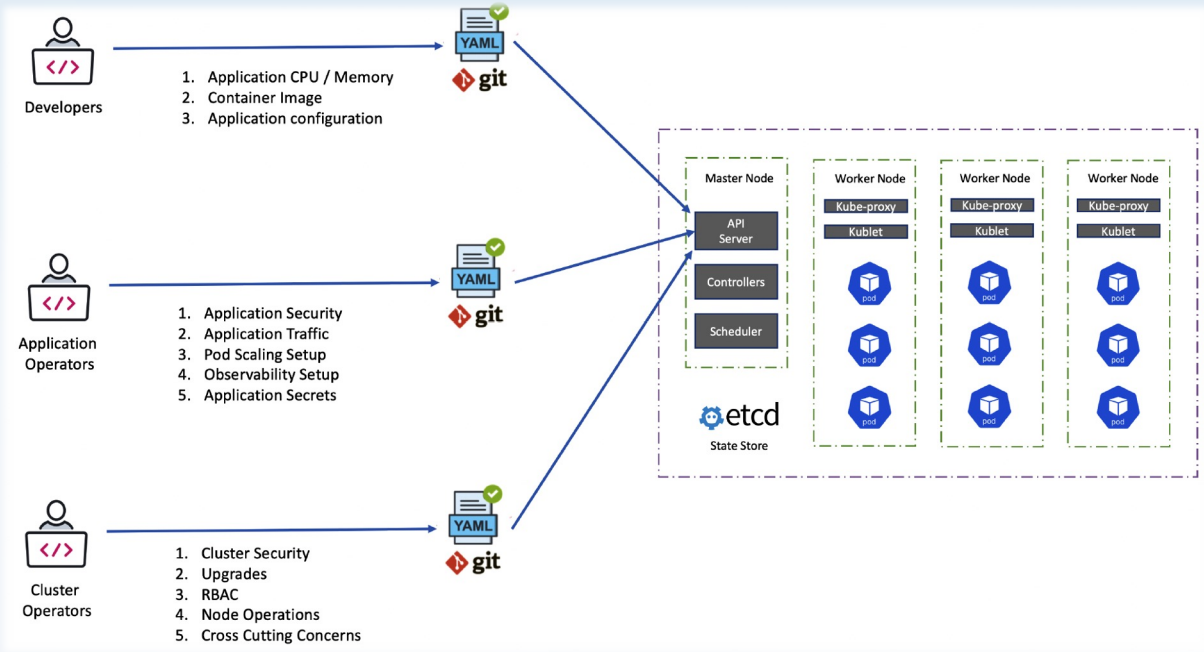
1. 应用研发**认知负担**太高，生产力下降，阻碍企业创新
2. 平台成为企业效率的瓶颈，需要**新架构**释放生产力
3. 平台构建可复用 Module，研发**自服务**是可行的技术方案
4. 基于开源产品构建 IDP 的 ROI 更高

Part 02

KCL 云原生策略配置语言

云原生时代，基础设施即代码（IaC）是开发者体验的核心

基础设施代码化（IaC）已成为自动化和管理云资源的关键



需要一种减轻开发者认知负担和开发成本、提供高效动态配置管理，并且通过标准的配置测试与验证手段来保证可靠性的配置管理方式

认知负担

- 应用开发人员需要面对复杂的基础设施和平台概念
- 不像云基础设施配置有 Terraform 等 IaC 工具，针对 Kubernetes 平台缺乏轻量的配置组合和校验工具

静态配置

- YAML 膨胀，维度爆炸
- 跨团队配置协作负担和配置漂移

效率、可靠性低

- 缺乏标准的测试验证手段，大多是胶水代码或者脚本的拼盘
- 缺乏高效配置协同的工具，大多通过人肉拉群解决

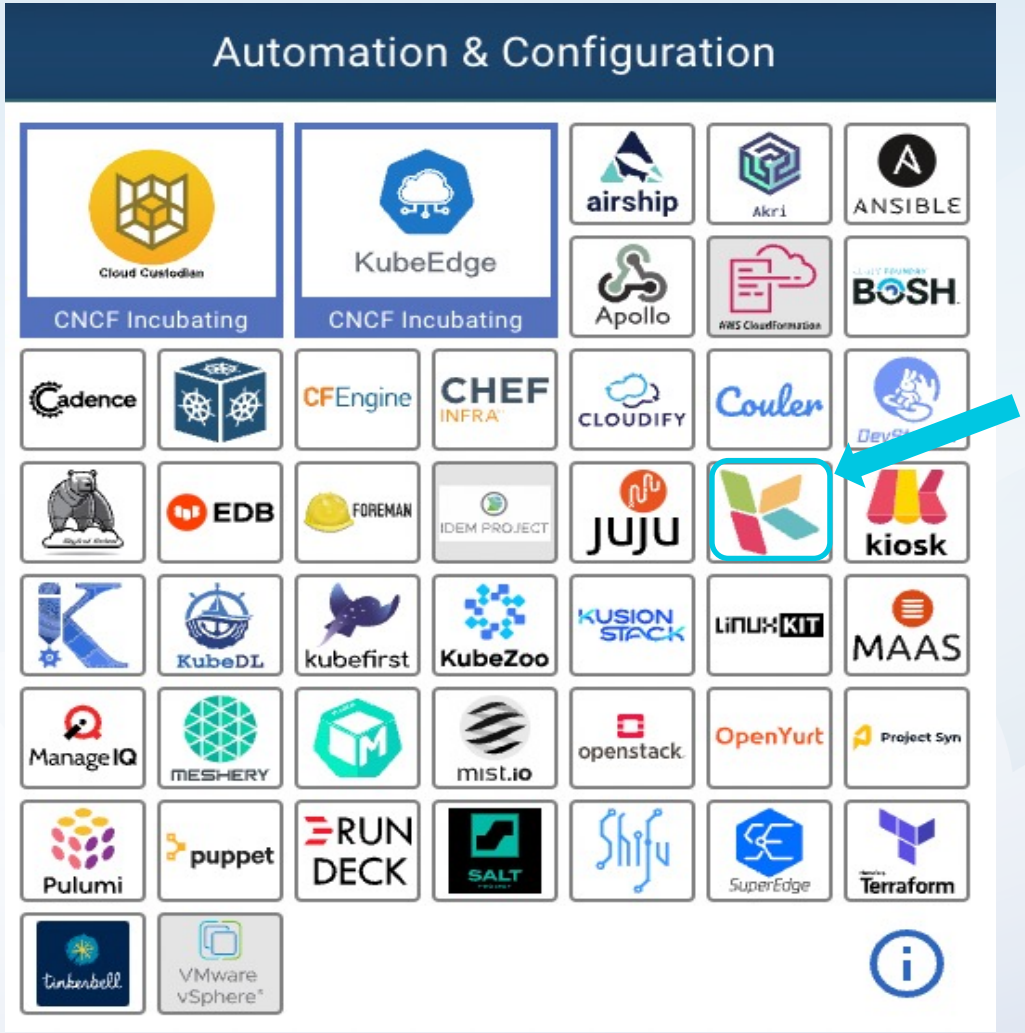
KCL 云原生策略配置语言

Mutation, Validation, Abstraction
Production-Ready

KCL is an open-source constraint-based record & functional language mainly used in configuration and policy scenarios.

- ✓ **支持流程控制：**KCL 支持流程控制/lambda 表达式等基于现代编程语言元素提供的流程控制，提供了动态配置管理的能力。
- ✓ **支持配置测试与验证：**KCL 使用强类型系统，支持 assert, check, rule 等语言特性支持编写配置验证策略。
- ✓ **降低认知负担和开发成本：**通过 Schema 抽象数据结构，提供丰富的三方库，以应用为中心的模型屏蔽复杂的基础设施和平台概念。

面向云原生领域的专用配置策略语言
(2022.6 开源，2023.9 成为 CNCF 基金会托管的 Sandbox 项目)



KCL 语言特性

Mutation

流程控制/lambda 表达式

```
dataLoop7 = [i for i, _ in data]
```

```
x = lambda {
    _e = 1
    if True:
        _e = 2
    {
        e: _e
    }
}
```

Validation

类型系统/约束定义/策略编写

```
x = "length"
assert len(x) == 6 # True
```

```
schema Sample:
    foo: str
    bar: int
    fooList: [str]

    check:
        bar > 0  #
        bar < 100
```

Abstraction

Schema 结构体定义/三方库导入

```
schema Sample:
    foo: str
    bar: int    variable is defined here,
    fooList: [str]

sample_inst = Sample{
    foo: "foo",
    bar: "1",    expected int, got str(1)
    fooList: ["foo", "bar"]
}
```

```
import k8s.api.core.v1 as k8core

k8core.Pod {
    metadata.name = "web-app"
    spec.containers = [{
        name = "main-container"
        image = "nginx"
        ports = [{containerPort = 80}]
    }]
}
```

KCL & KRM & 动态配置管理

- **Mutation**

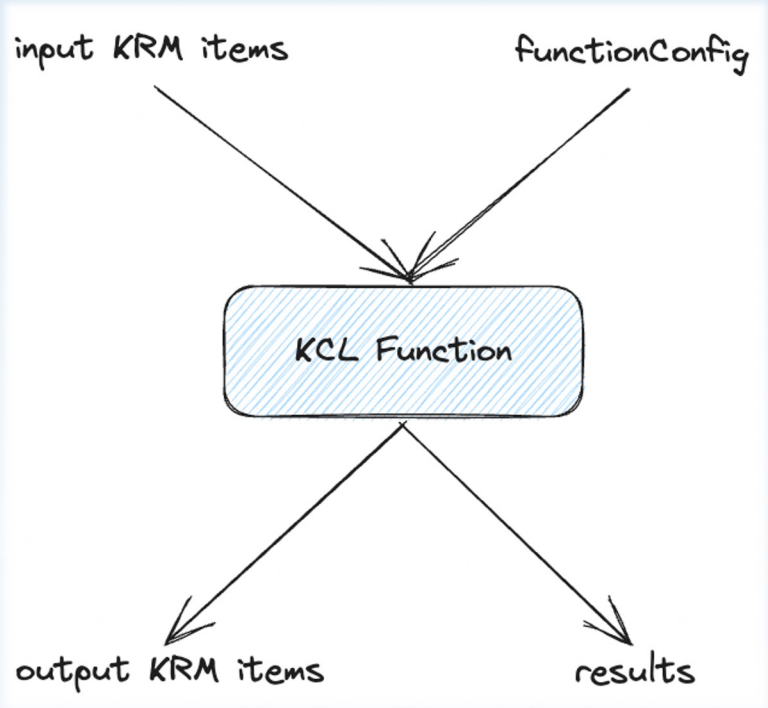
```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: set-annotations
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: mutation
      documentation: >-
        Add or change annotations
spec:
  params:
    toAdd: addValue
  source: oci://ghcr.io/kcl-lang/set-annotation
```

- **Abstraction**

```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: web-service
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: abstraction
      documentation: >-
        Web service application abstraction
spec:
  params:
    name: app
    containers:
      ngnix:
        image: ngnix
        ports:
          containerPort: 80
    labels:
      name: app
  source: oci://ghcr.io/kcl-lang/web-service
```

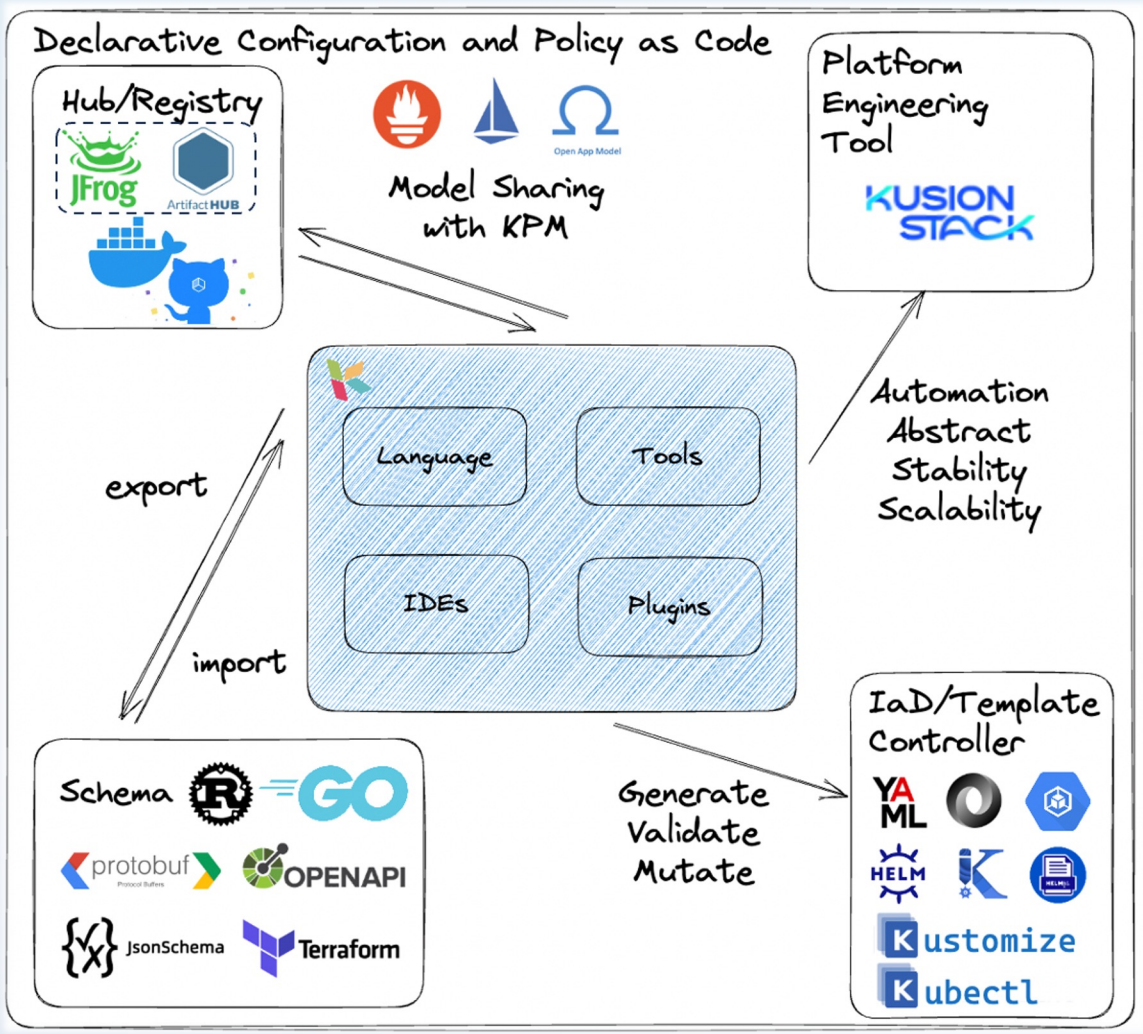
- **Validation**

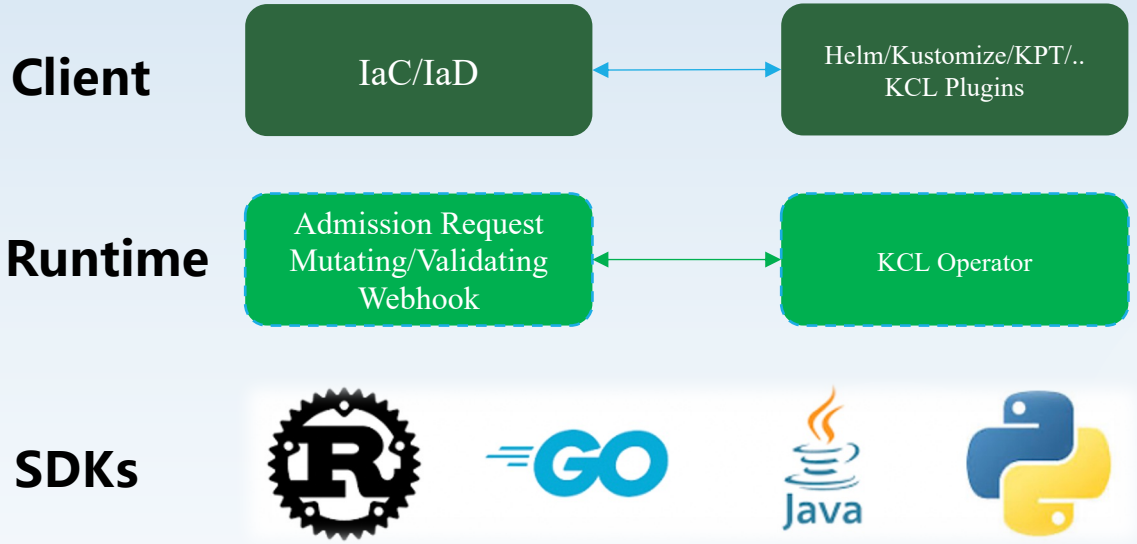
```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: https-only
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: validation
      documentation: >-
        Requires Ingress resources to be HTTPS only.  Ingress resources must
        include the `kubernetes.io/ingress.allow-http` annotation, set to `false`.
        By default a valid TLS {} configuration is required, this can be made
        optional by setting the `tlsOptional` parameter to `true`.
        More info: https://kubernetes.io/docs/concepts/services-networking/ingress/#tls
spec:
  source: oci://ghcr.io/kcl-lang/https-only
```



- **遵循统一的 KRM Function 规范**
- **多种代码源支持:** OCI, Git, Https, Filesystem...
- **可编程可扩展: 使用 KCL 语言简单编写模型**

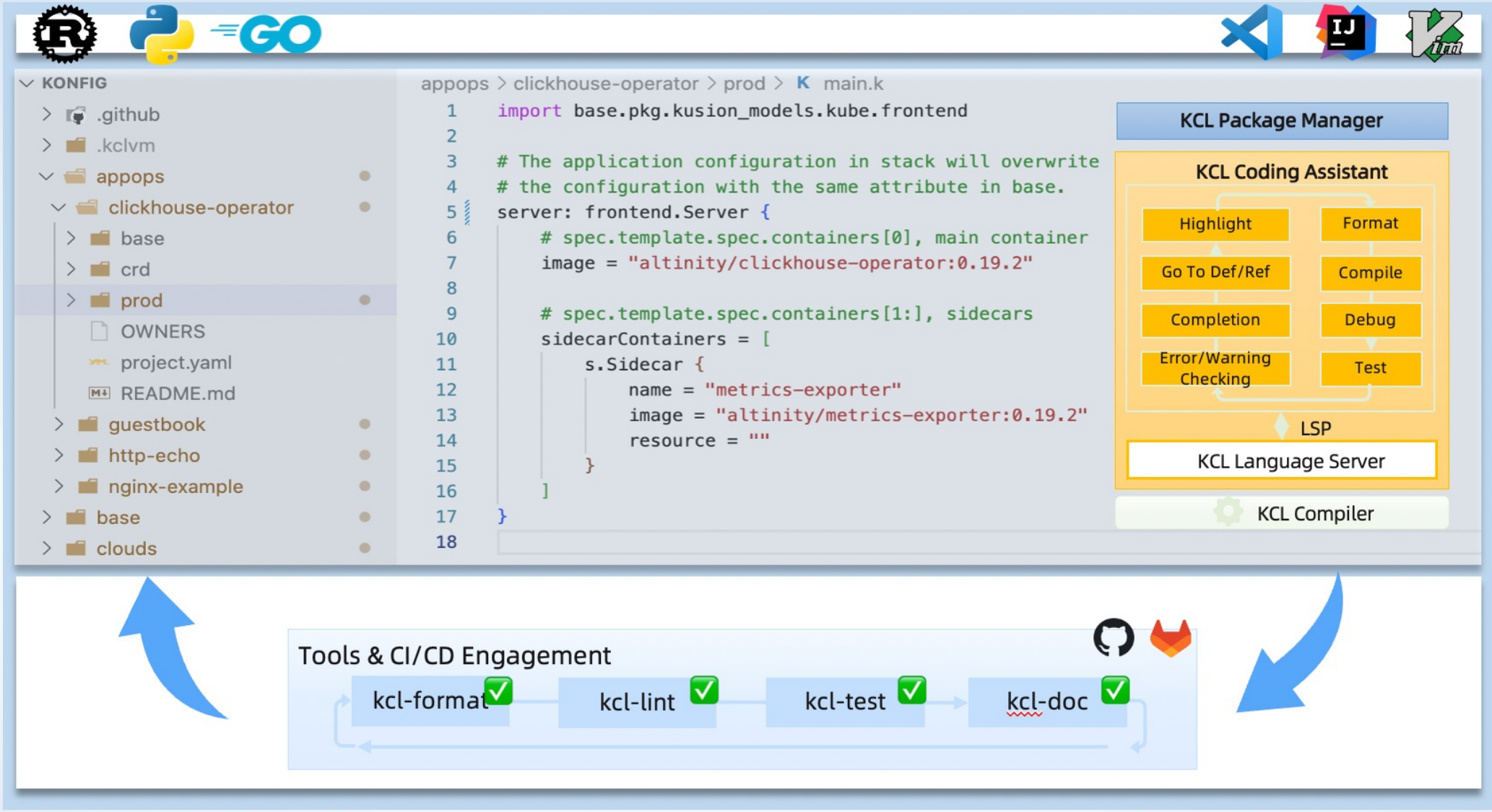
KCL 生态集成





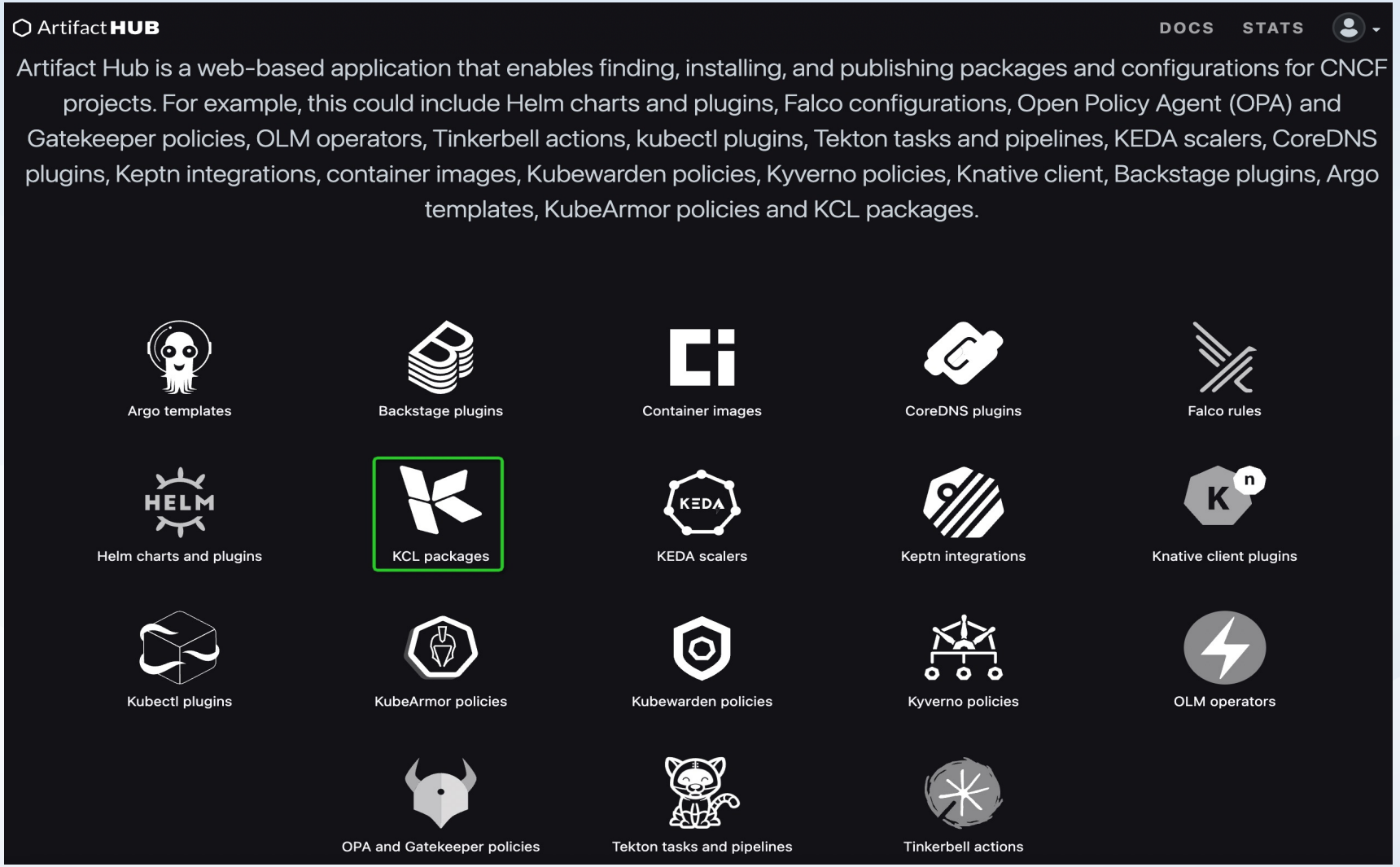
- **多语言 SDK:** Rust, Go, Python, Java SDK
- **包管理支持:** KPM 工具和多种 Registry /Hub支持
- **数据和 Schema 集成**：KCL Import/Export 工具
- **运行时集成:** 使用 KCL Operator 而不是重复开发 Kubernetes Admission Webhook
- **KRM 支持:** 统一的规范和插件支持 e.g., kubectl-kcl plugin, helm-kcl plugin, kustomize-kcl plugin, kpt-kcl-plugin ...
- **平台工程工具支持:** 作为 DCM 语言配合不同引擎/编排器进行应用交付

KCL 周边工具


KONFIG
.github
.kclvm
appops
clickhouse-operator
base
crd
prod
OWNERS
project.yaml
README.md
guestbook
http-echo
nginx-example
base
clouds
appops > clickhouse-operator > prod > main.k
1 import base.pkg.kusion_models.kube.frontend
2
3 # The application configuration in stack will overwrite
4 # the configuration with the same attribute in base.
5 server: frontend.Server {
6 # spec.template.spec.containers[0], main container
7 image = "altinity/clickhouse-operator:0.19.2"
8
9 # spec.template.spec.containers[1:], sidecars
10 sidecarContainers = [
11 s.Sidecar {
12 name = "metrics-exporter"
13 image = "altinity/metrics-exporter:0.19.2"
14 resource = ""
15 }
16]
17 }
18
KCL Package Manager
KCL Coding Assistant
Highlight
Format
Go To Def/Ref
Compile
Completion
Debug
Error/Warning Checking
Test
LSP
KCL Language Server
KCL Compiler
Tools & CI/CD Engagement
kcl-format
kcl-lint
kcl-test
kcl-doc

Artifacthub & KCL

- **开箱即用:** 一行命令添加依赖 e.g., kcl mod add k8s，现阶段官网模型 **200+** (欢迎共建)
- **多种场景支持**：配置编辑、校验、模型抽象，Kubernetes 生态模型，Terraform 生态模型，应用配置 ...
- **多种 Registry /Hub支持**: Docker Hub, ghcr.io, ...



实践: Kubernetes 动态配置管理

- 少数几行 KCL 代码即可完成对应配置编辑功能 (客户端和运行时代码可以复用)
- 无需开发额外的 Kubernetes Webhook 编辑和验证配置

- Step 1. 在集群当中安装 KCL Operator
- Step 2. Apply KCL and K8s manifests

```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: set-annotation
spec:
  params:
    annotations:
      managed-by: kcl-operator
  # Resource modification can be achieved with just one line of KCL code
  source: |
    items = [item | {metadata.annotations: option("params").annotations} for item in option("items")]
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  annotations:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

- Step 3. 获得资源 Mutation 结果

```
kubectl get po nginx -o yaml | grep kcl-operator

    managed-by: kcl-operator
```

保存在 OCI Registry 中支持复用

```yaml
# Reference the annotation modification model on OCI
source: oci://ghcr.io/kcl-lang/set-annotation
```

实践: Terraform 配置验证

main.tf

```
provider "aws" {
    region = "us-west-1"
}
resource "aws_instance" "web" {
  instance_type = "t2.micro"
  ami = "ami-09b4b74c"
}
resource "aws_autoscaling_group" "my_asg" {
  availability_zones        = ["us-west-1a"]
  name                      = "my_asg"
  max_size                  = 5
  min_size                  = 1
  health_check_grace_period = 300
  health_check_type         = "ELB"
  desired_capacity          = 4
  force_delete              = true
  launch_configuration      = "my_web_config"
}
resource "aws_launch_configuration" "my_web_config" {
    name = "my_web_config"
    image_id = "ami-09b4b74c"
    instance_type = "t2.micro"
}
```

terraform plan

tfplan.json

```
{
  "format_version": "0.1",
  "terraform_version": "0.12.6",
  "planned_values": {
    "root_module": {
      "resources": [
        {
          "address": "aws_autoscaling_group.my_asg",
          "mode": "managed",
          "type": "aws_autoscaling_group",
          "name": "my_asg",
          "provider_name": "aws",
          "schema_version": 0,
          "values": {
            "availability_zones": [
              "us-west-1a"
            ],
            "desired_capacity": 4,
            "enabled_metrics": null,
            "force_delete": true,
            "health_check_grace_period": 300,
            "health_check_type": "ELB",
            "initial_lifecycle_hook": [],
```

validate

generate

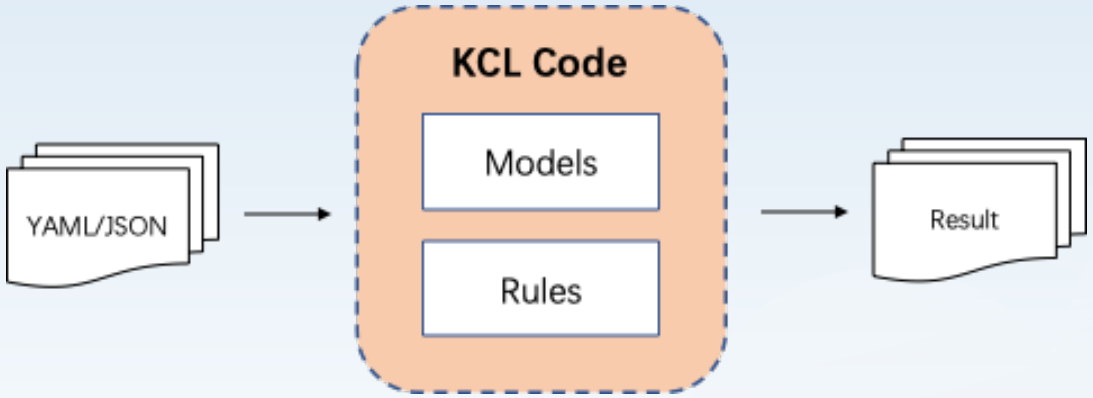
main.k

```
schema TFPlan:
    # Omit other attributes
    [...str]: any
    resource_changes?: [AcceptableChange]

schema AcceptableChange:
    # Omit other attributes
    [...str]: any
    check:
        # Reject AWS autoscaling group Resource delete action
        all action in change.actions {
            action not in ["delete"]
        } if type == "aws_autoscaling_group", "Disable AWS autoscaling group resourc
```

kcl-vet tfplan.json main.k

Validation Process

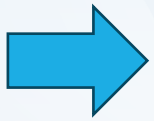


- ✓ **多种数据支持**：JSON/YAML 等数据支持
- ✓ **结构定义**：Schema 结构化定义及自定义错误支持
- ✓ **生态集成**：OpenAPI/Terraform Provider Schema 转换 KCL Schema 支持
- ✓ **开箱即用**：丰富的配置策略模型库和代码示例 (欢迎共建)

https://kcl-lang.io/docs/user_docs/guides/working-with-terraform/validation

实践: Kubernetes 配置生成

UI/CLI/API



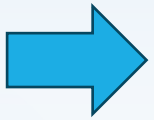
Standalone KCL Form

```
import .app

app.App {
    name = "app"
    containers.ngnix = {
        image = "ngnix"
        ports = [{containerPort = 80}]
    }
    service.ports = [{ port = 80 }]
}
```

KRM KCL Form

```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: web-service
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: abstraction
      documentation: >-
        Web service application abstraction
spec:
  params:
    name: app
    containers:
      ngnix:
        image: ngnix
        ports:
          containerPort: 80
    labels:
      name: app
  source: oci://ghcr.io/kcl-lang/web-service
```



Kubernetes Manifests

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-e
  namespace: ng
spec:
  ports:
  - nodePort: 30201
    port: 80
    targetPort: 80
  selector:
    app.kubernetes.io/name: nginx-example
    app.kubernetes.io/env: dev
    app.kubernetes.io/instance: nginx-example-dev
    app.kubernetes.io/component: nginx-exampledev
  type: NodePort
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: nginx-example
```

在蚂蚁和其他公司的实践

1K/day	10K+/day	1 : 9	100K+
Pipelines	KCL Compilations	Plat : Dev	Commits
600+	5.7K+	100K+	10K+
Contributors	Projects	Pods	Workloads

Adopted by

HUAWEI KYLIGENCE® 有赞 SafetyCulture

欢迎加入我们

- Web Site
  - https://kusionstack.io/
  - https://kcl-lang.io/
- Github
  - https://github.com/KusionStack/
  - https://github.com/kcl-lang
- Slack
  - KusionStack
  - CNCF KCL Slack Channel

微信群小助手

钉钉群

Thanks.