KUBERNETES
COMMUNITY DAYS HANGZHOU

KUBERNETES

基于云原生供应链的配置策略管理新范式

Kubernetes Resource Model (KRM) KCL Specification

演讲人：徐鹏飞

COMMUNITY DAYS
HANGZHOU 2023

KUBERNETES

KUBERNETES
COMMUNITY DAYS HANGZHOU

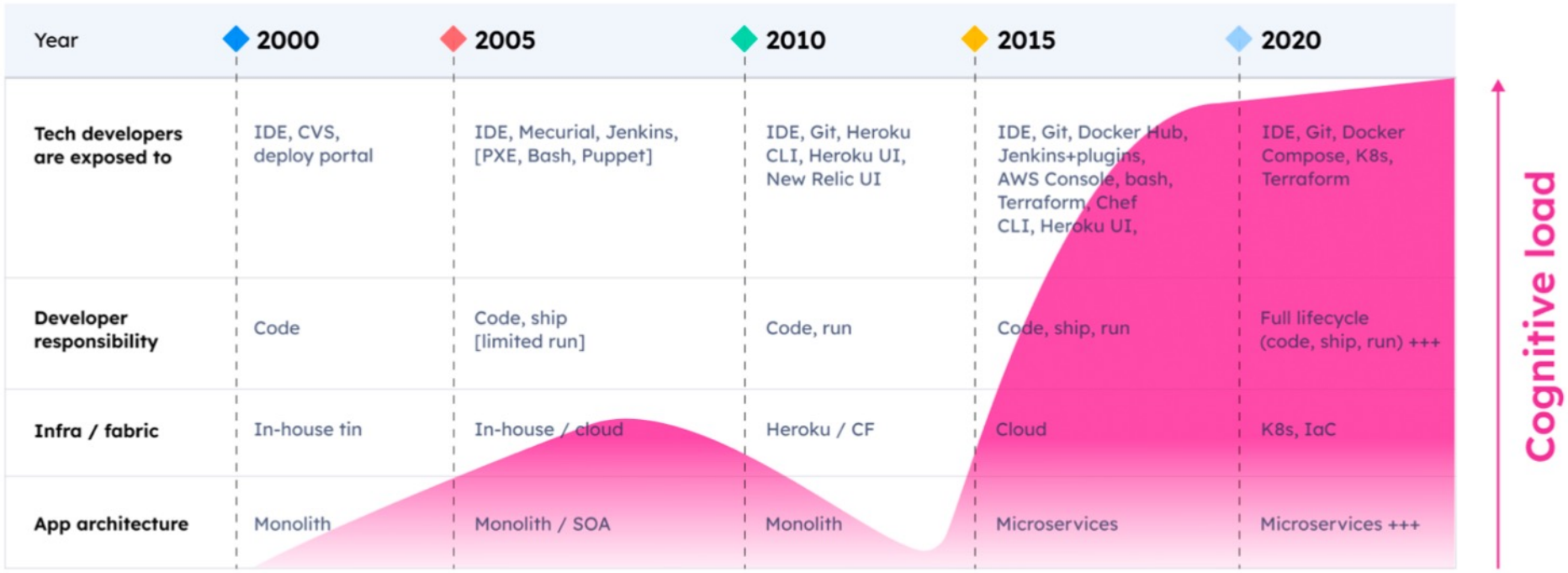
目录

CONTENTS

01. 背景

为什么我们需要一个新范式

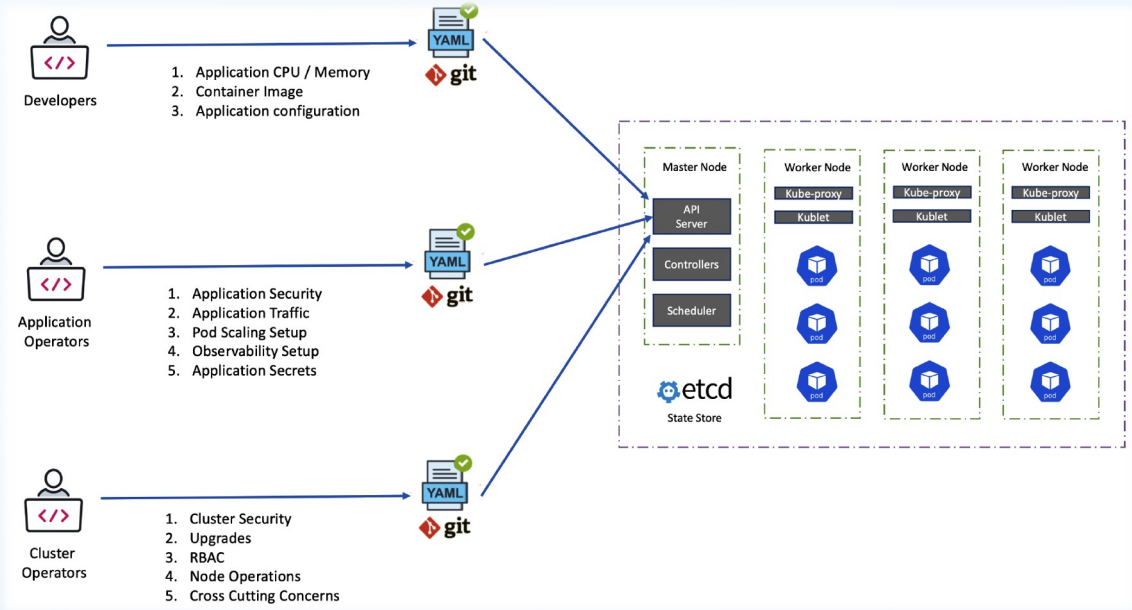
背景



Inspired by Daniel Bryant at PlatformCon 2022

随着云原生技术的发展，Kubernetes 底座和 Terraform 等 IaC 工具已成为越来越流行的管理和部署基于 (云) API 的应用程序工具

问题



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，Kubernetes 是平台的平台，特别是在客户端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

减轻基础设施对开发人员的认知负担，提高配置管理效率

Kubernetes 中的声明式应用管理: https://docs.google.com/document/d/1cLPGweVEYrVqQvBLJg6sxV-TrE5Rm2MNOBA_cxZP2WU/edit#
CNCF 平台工程白皮书: https://tag-app-delivery.cncf.io/whitepapers/platforms/
Google SRE 工作手册: https://sre.google/workbook/configuration-specifics/

KUBERNETES

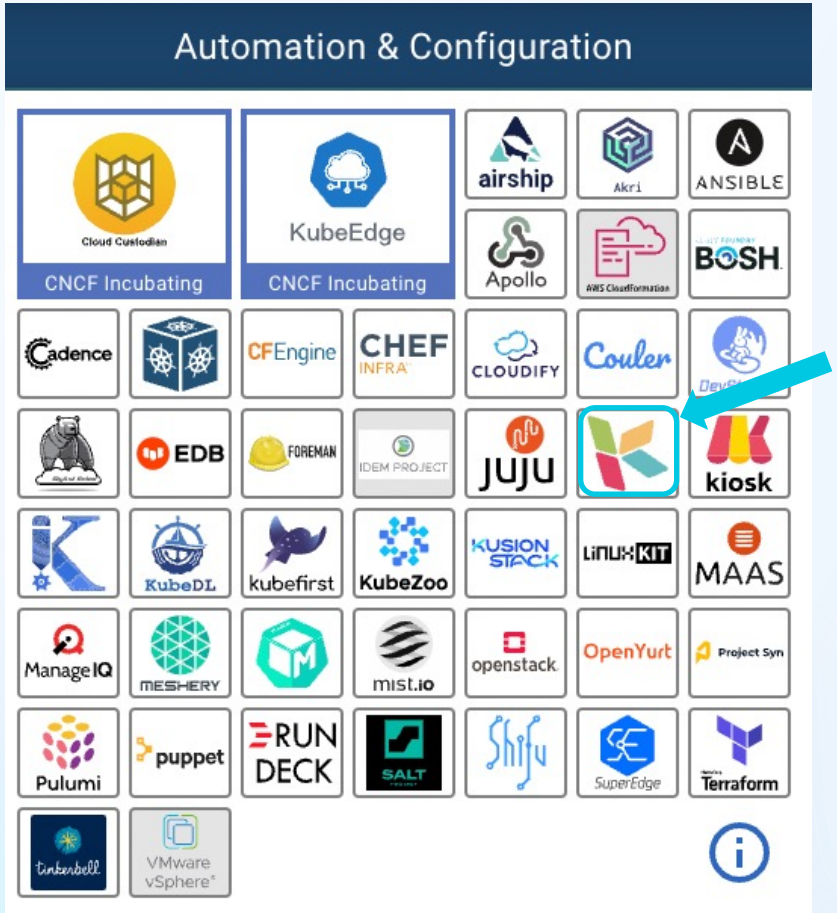
KUBERNETES
COMMUNITY DAYS HANGZHOU

02. 概念

KCL、KRM KCL 规范、语言核心元素

KCL

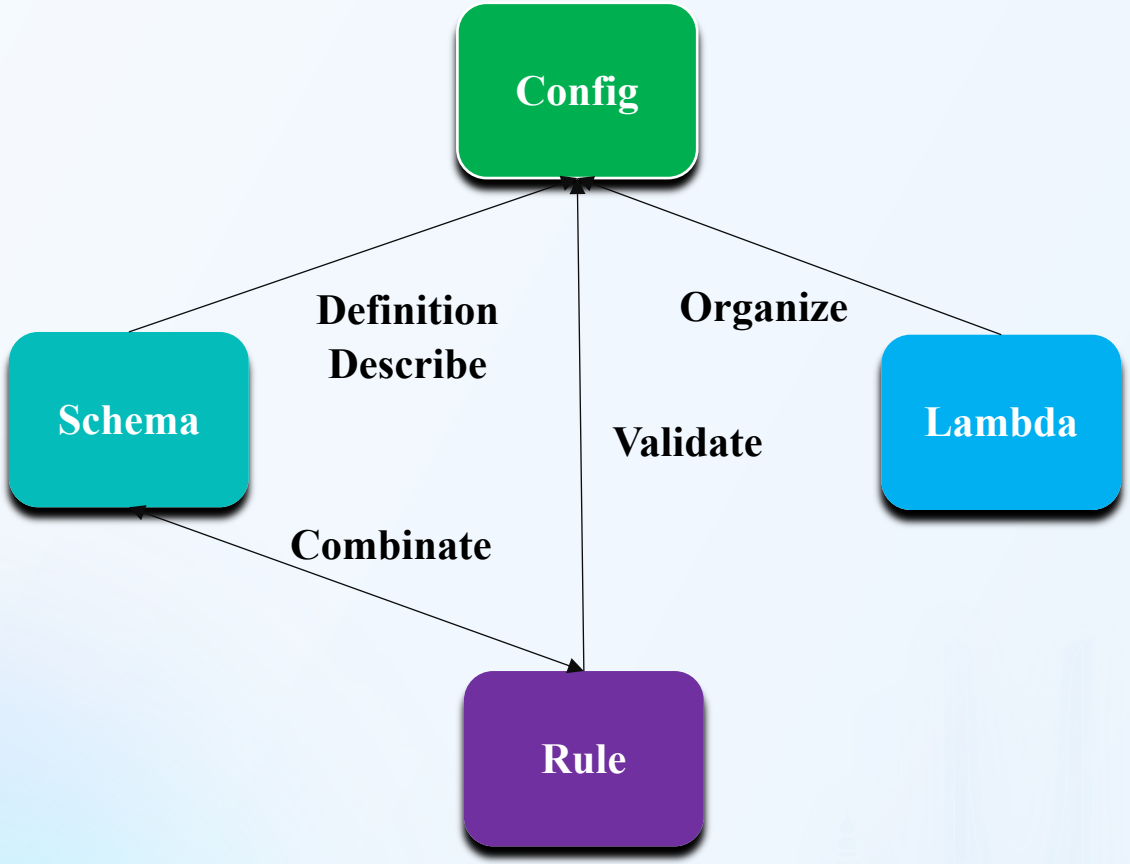
面向云原生领域的专用配置策略语言 (2022.6 开源，2023.9 成为 CNCF 基金会托管的 Sandbox 项目)



- ✓ **领域特定**：以收敛的语言和工具集合解决领域问题近乎无限的变化和复杂性
- ✓ **以应用为中心**：开发者可以理解的声明式应用配置模型
- ✓ **关注点分离**：应用/平台 Dev/SRE
- ✓ **动态配置管理**: 多租户多环境、可编程可扩展
- ✓ **风险左移**：实时的配置错误提示
- ✓ **可复用扩展**: OCI 等标准**软件供应链**集成和包管理工具支持，官方 Registry 提供 60+ 模型包
- ✓ **引擎解耦**：建立在一个完全开放的云原生世界当中，几乎不与任何编排/引擎工具或者 Kubernetes 控制器绑定，可同时为**客户端和运行时场景**提供 **API 抽象、组合和校验的能力**

Standalone KCL

Config + Schema + Rule + Lambda Pattern: $k = (T)v$




```
import k8s.core.v1
# Create a Kubernetes Deployment resource.
v1.Deployement {
    metadata.name = "nginx"
    metadata.labels.app = metadata.name
    spec = {
        replicas = 3
        selector.matchLabels.app = metadata.name
        template = {
            metadata.labels.app = metadata.name
            spec.containers = [{
                name = metadata.name
                image = "nginx"
                ports = [{ containerPort = 80 }]
            }]
        }
    }
}
```

可复用可扩展、抽象和组合能力、稳定性、高性能

KRM KCL

- **Mutation**

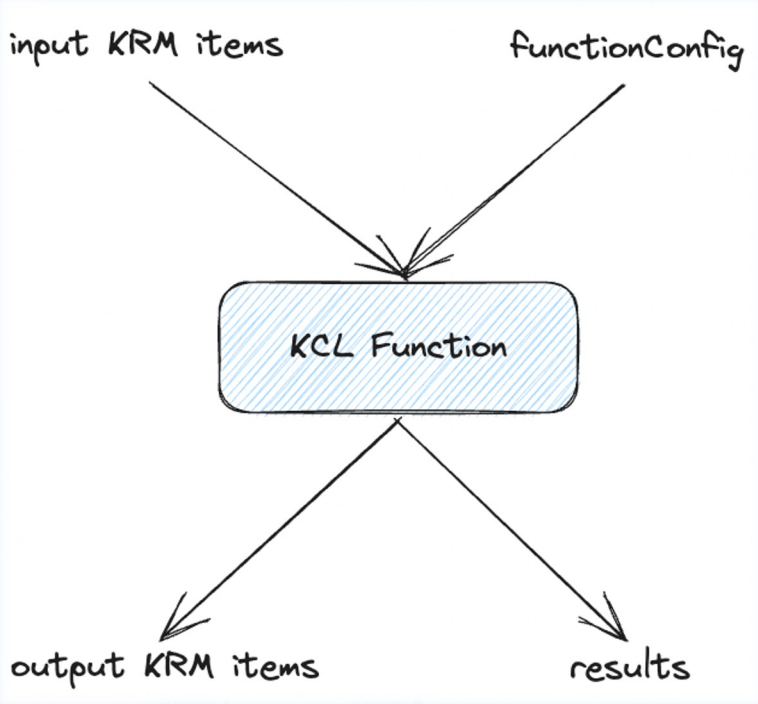
```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: set-annotations
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: mutation
      documentation: >-
        Add or change annotations
spec:
  params:
    toAdd: addValue
  source: oci://ghcr.io/kcl-lang/set-annotation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

- **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**

KUBERNETES

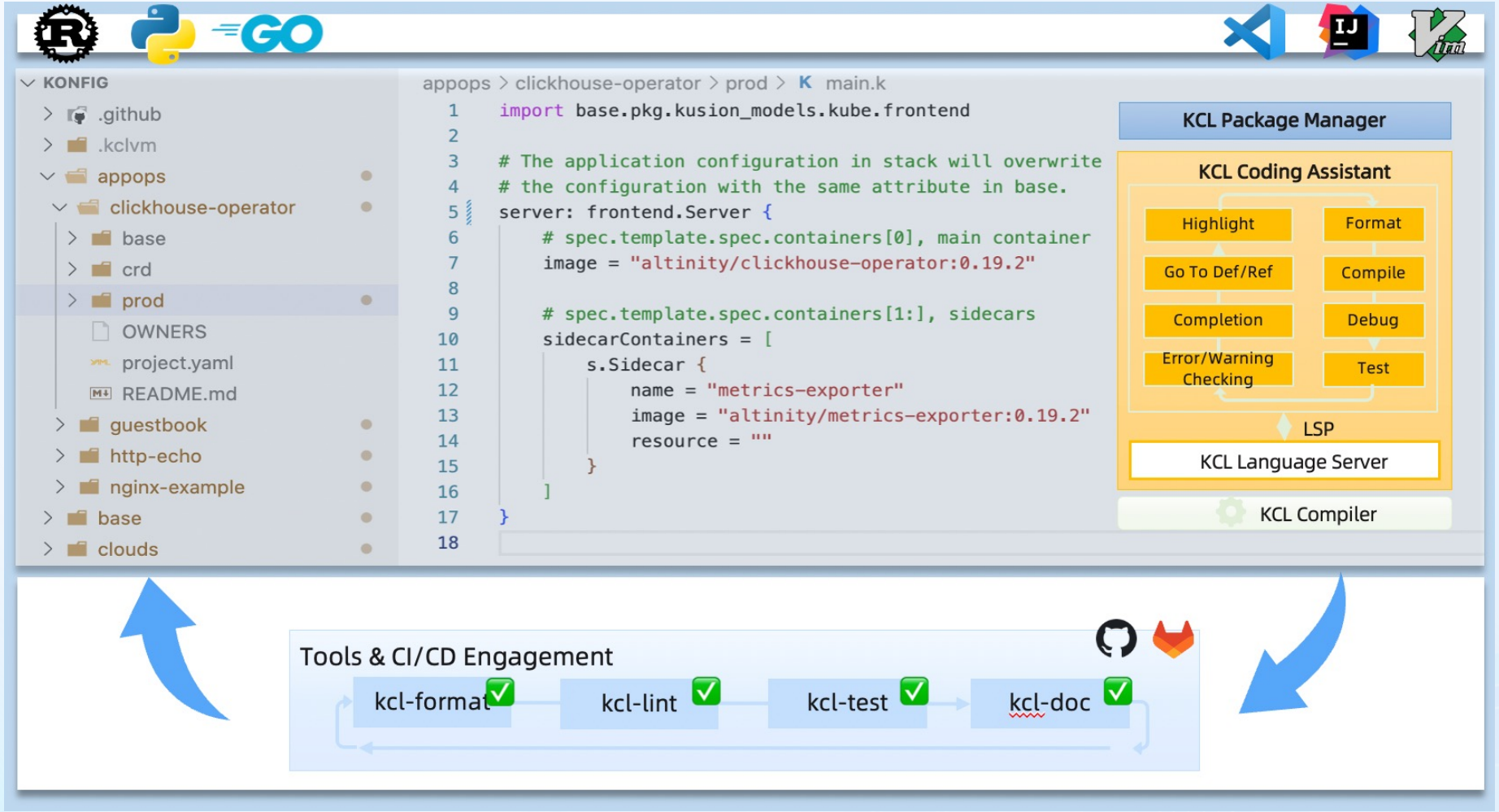
KUBERNETES
COMMUNITY DAYS HANGZHOU

03. 体验

语言、工具、云原生集成

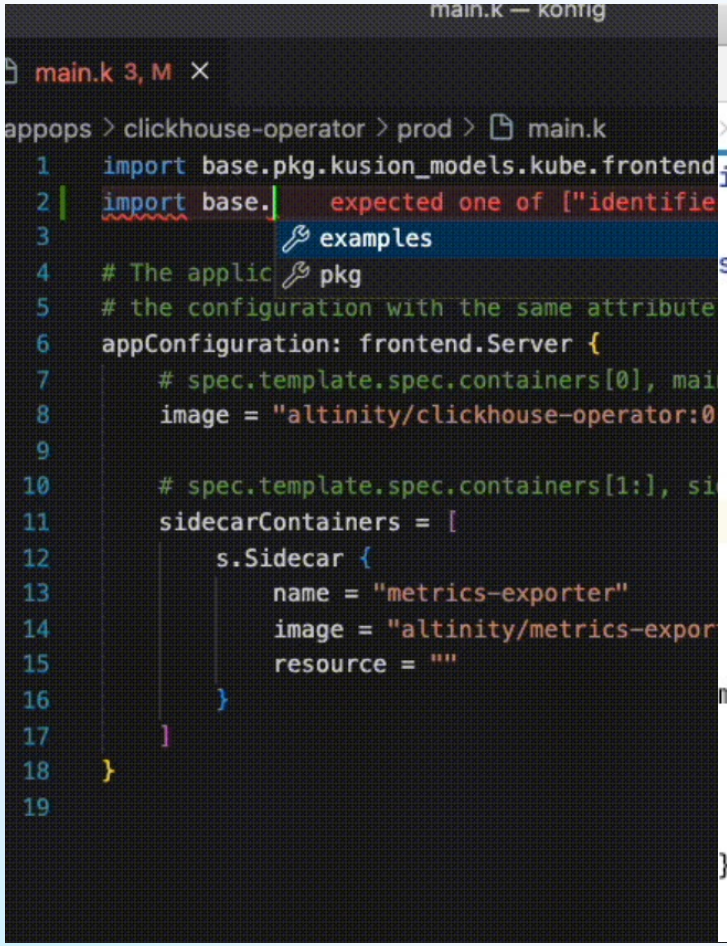
Workspace

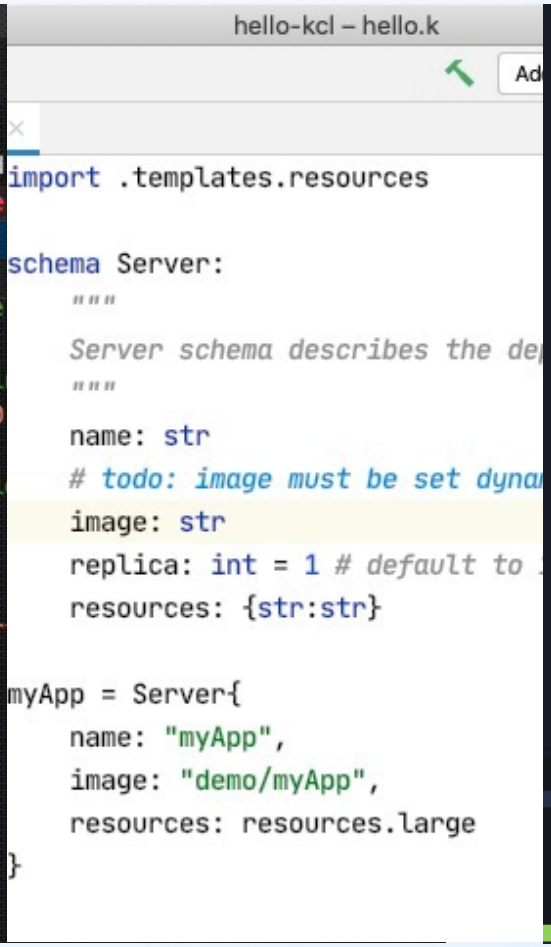
Language + Tools + IDEs + SDKs + Plugins

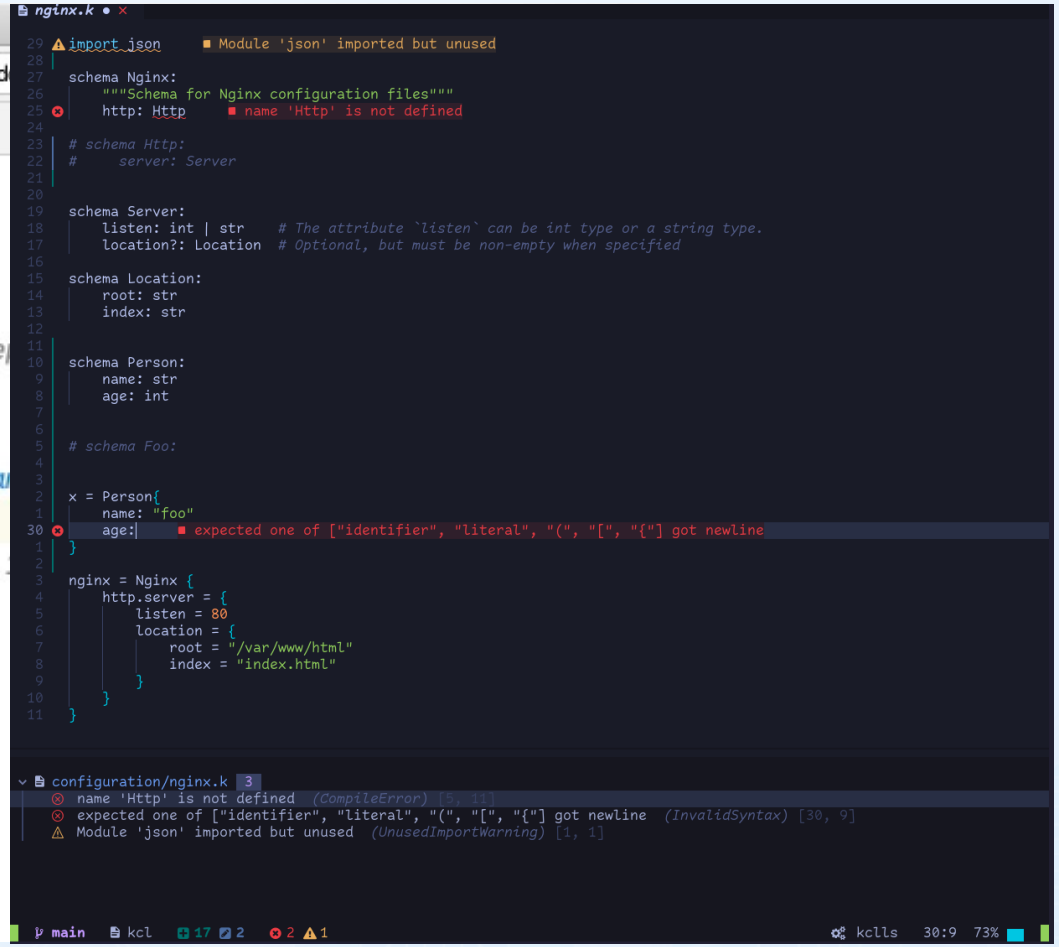


IDE Extension

- VS Code
- Idea
- NeoVim

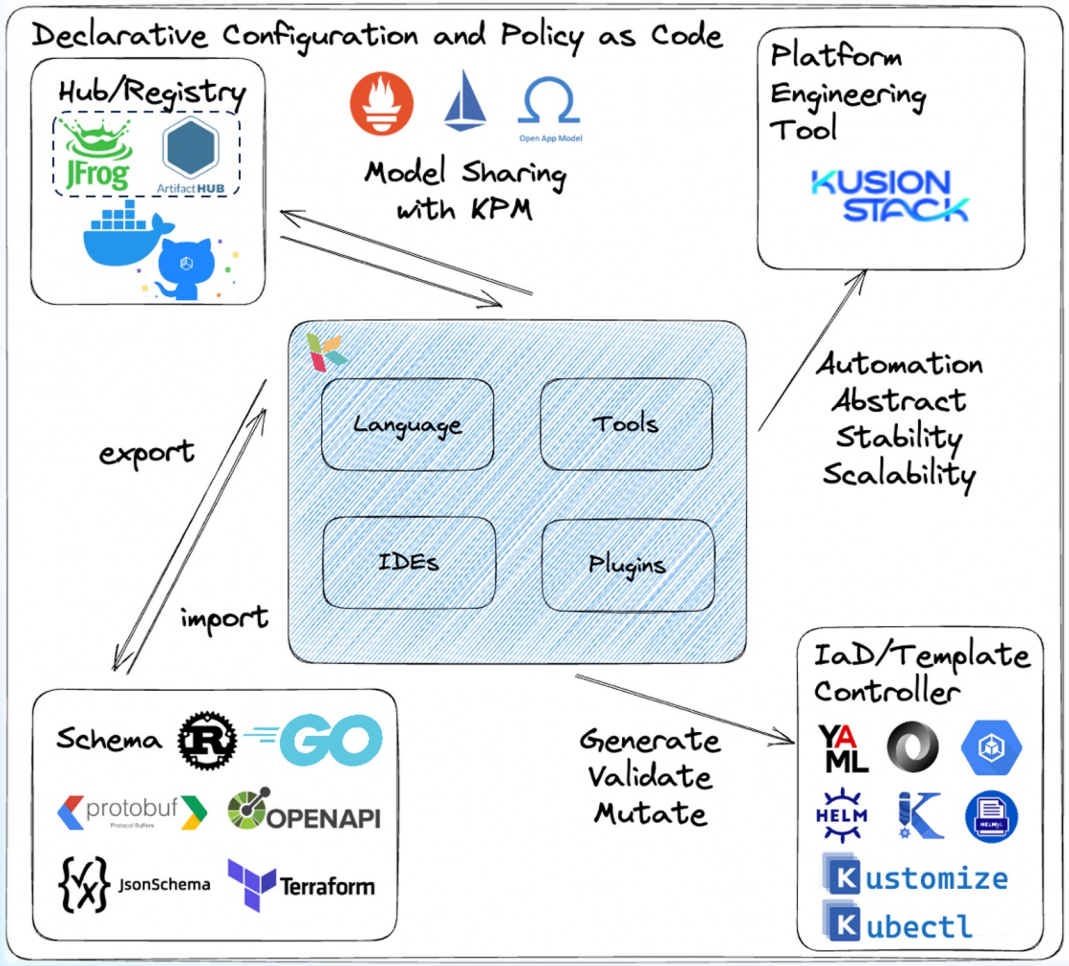






https://kcl-lang.io/docs/tools/Ide/

Integration



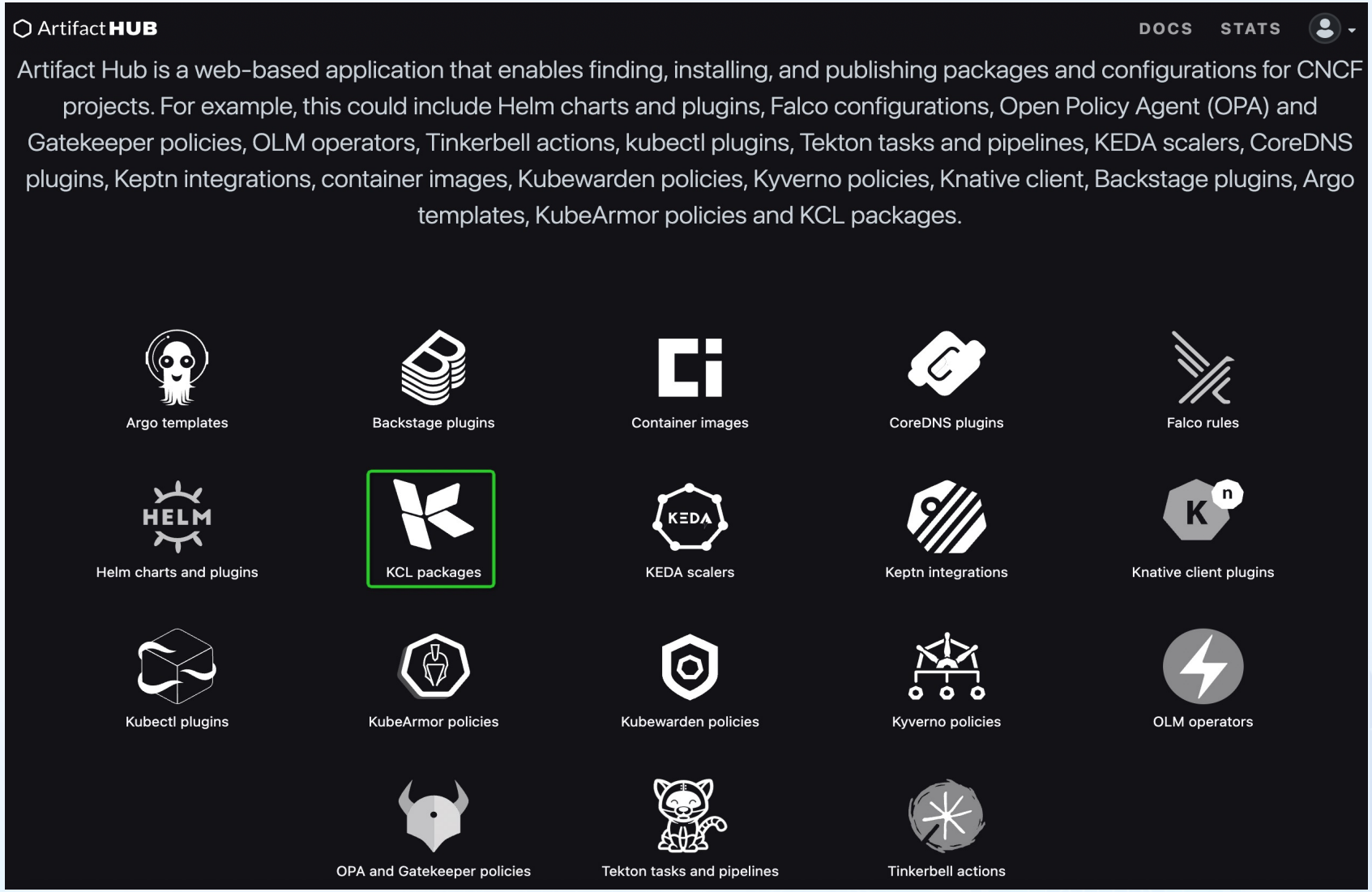
Client IaC/IaD ↔ Helm/Kustomize/KPT/.. KCL Plugins

Runtime Admission Request Mutating/Validating Webhook ↔ KCL Operator

SDKs

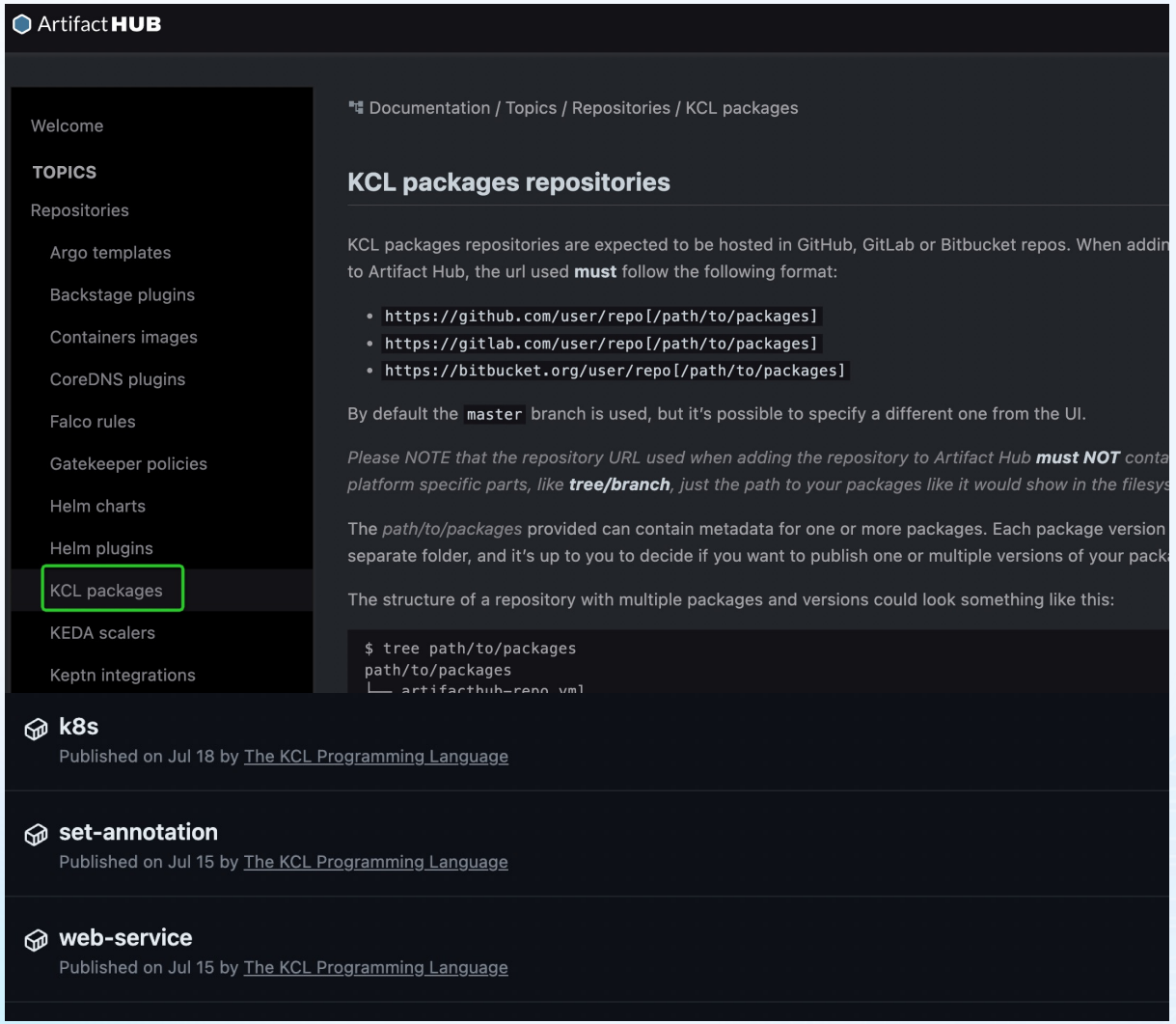
- **多语言 SDK:** Rust, Go, Python, Java SDK
- **包管理支持:** KPM 工具和多种 Registry /Hub支持
- **数据和 Schema 集成**：KCL Import/Export 工具
- **运行时集成:** 使用 KCL Operator 而不是重复开发 Kubernetes Admission Webhook
- **KRM 支持:** 统一的规范和插件支持 e.g., kubectl-kcl plugin, helm-kcl plugin, helmfile-kcl plugin, kustomize-kcl plugin, kpt-kcl-plugin ...
- **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

Artifact Hub KCL Integration (Staging)



ArtifactHub KCL Package 预览版：https://staging.artifacthub.io/

Registry & Modules



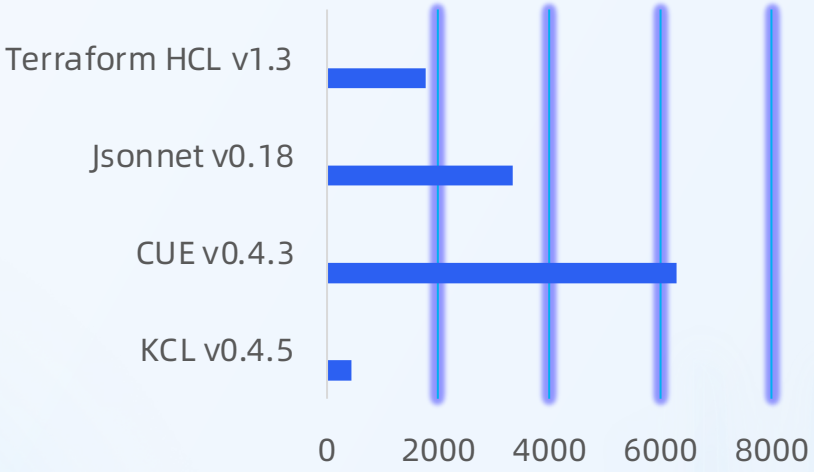
- **开箱即用:** 一行命令添加依赖 e.g., kpm add k8s，现阶段官网模型 **60+** (欢迎共建)
- **多种场景支持**：配置编辑、校验、模型抽象，Kubernetes 生态模型，Terraform 生态模型，应用配置，...
- **多种 Registry /Hub支持**: Docker Hub, ghcr.io, Harbor, ...
- **Artifact Hub KCL Integration (WIP):** 支持浏览器查询、身份认证等

ArtifactHub KCL Module 预览版：https://staging.artifacthub.io/packages/search?kind=20&sort=relevance&page=1

Performance

Loop and Function

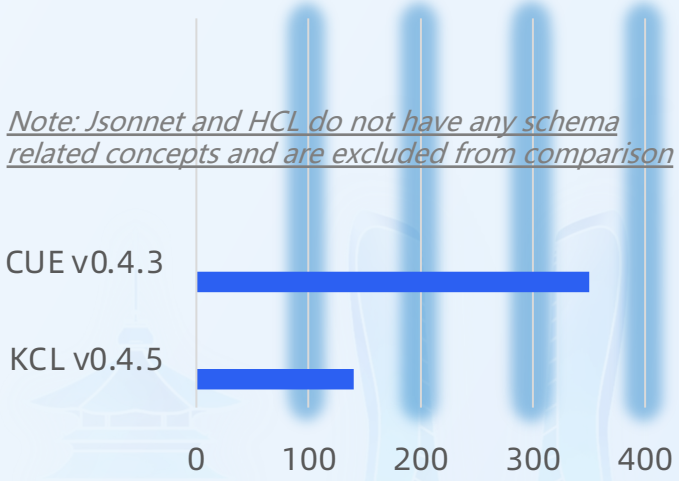
```
a = lambda x: int, y: int -> int {
    max([x, y])
}
temp = {"a${i}": a(1, 2) for i in range(10000)}
```



Kubernetes Configuration

```
import kubernetes.api.apps.v1

deployment = v1.Deployment {}
```

Note: Jsonnet and HCL do not have any schema related concepts and are excluded from comparison



Test environment: single core macOS 10.15.7 CPU: i7-8850H 2.6GHz 32GB 2400Mhz DDR4 No NUMA, e2e run time (ms)

KUBERNETES

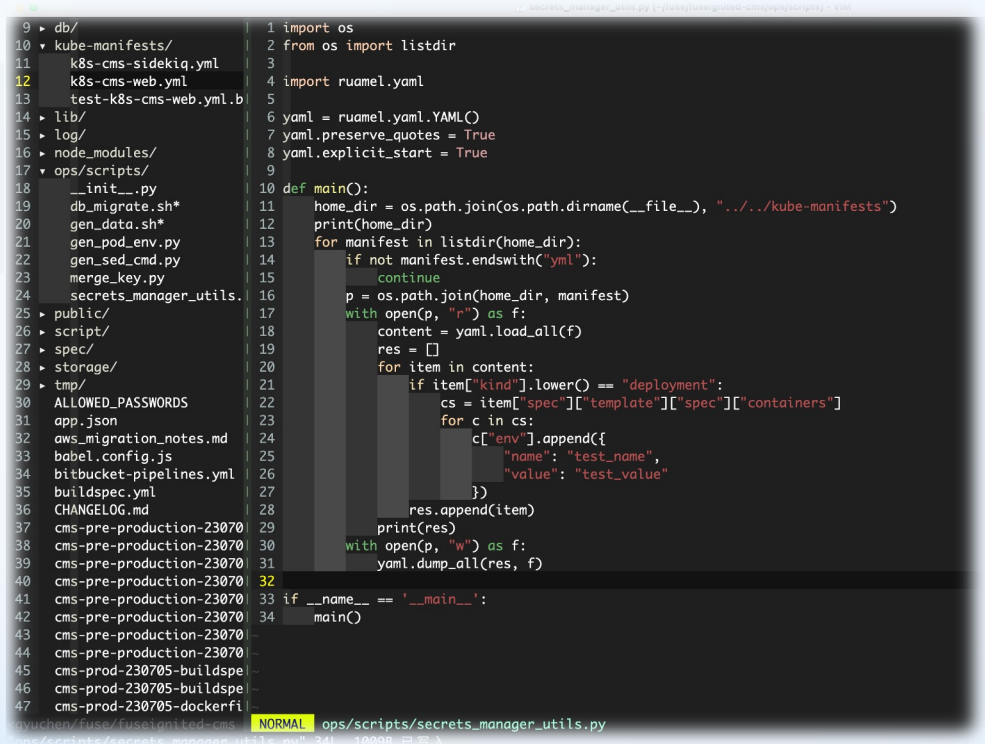
KUBERNETES
COMMUNITY DAYS HANGZHOU

04. 实践

Mutation、Validation、Abstraction

Kubernetes Mutation

Before



After

```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
spec:
  params:
    env:
    - name: test_name
      value: test_value
  source: oci://ghcr.io/kcl-lang/append-env
```

OCI Registry 上存放的 KCL 代码包

```
items = [item | {
    if item.kind == "Deployment":
        spec.template.spec.containers: [{
            env += option("params").env
        } for container in item.spec.template.spec.containers]
} for item in option("items") or []]
```

- 避免脚本、胶水代码拼盘
- 可复用，可测试

https://github.com/kcl-lang/krm-kcl

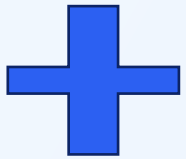
Kubernetes Validation

Disallow Service Load Balancer Module

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
  type: LoadBalancer # 错误地设置了 LoadBalancer
```

Allow Https Only Module

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  # Required the annotation kubernetes.io/ingress.allow-http: "false"
  name: tls-example-ingress
spec:
  tls:
  - hosts:
      - https-example.foo.com
    secretName: testsecret-tls
  rules:
  - host: https-example.foo.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 80
```



```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: disallow-svc-lb
spec:
  source: oci://ghcr.io/kcl-lang/disallow-svc-lb
```

将 K8s Manifests 和 KCL 策略同时使用，检查 K8s Manifests 配置是否合规

```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: https-only
spec:
  source: oci://ghcr.io/kcl-lang/https-only
```

- 统一界面：KCL 可以同时编写 Schema 和约束条件，无需配合 OpenAPI Schema/JSON Schema 使用

https://github.com/kcl-lang/krm-kcl

Kubernetes Abstraction

UI/CLI/API

Code

Standalone KCL Form

```
import .app

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

KRM KCL Form

```
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
```

Generate

Kubernetes Manifests

```
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
```

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

```
apiVersion: a
kind: Deploym
metadata:
  name: ngin
  namespace:
spec:
  replicas: 1
  selector:
    matchLabe
      app.kut
      app.kut
      app.kut
      app.kut
  template:
    metadata:
      labels:
        app.k
        app.k
        app.k
        app.kubernetes.io/component: nginx-exampledev
    spec:
      containers:
      - image: nginx:latest
        name: main
        ports:
        - containerPort: 80
          protocol: TCP
        resources:
          limits:
            cpu: 100m
            memory: 100Mi
            ephemeral-storage: 1Gi
          requests:
            cpu: 100m
            memory: 100Mi
            ephemeral-storage: 1Gi
```

- Kubernetes API 自由组合/抽象，关注点分离

https://github.com/kcl-lang/krm-kcl

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

```
kubectl apply -f- << EOF
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: set-annotation
spec:
  params:
    annotations:
      managed-by: kcl-operator
  source: oci://ghcr.io/kcl-lang/set-annotation
EOF
```

```
kubectl apply -f- << EOF
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  annotations:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
EOF
```

- Step 3. 获得资源 Mutation/Validation 结果

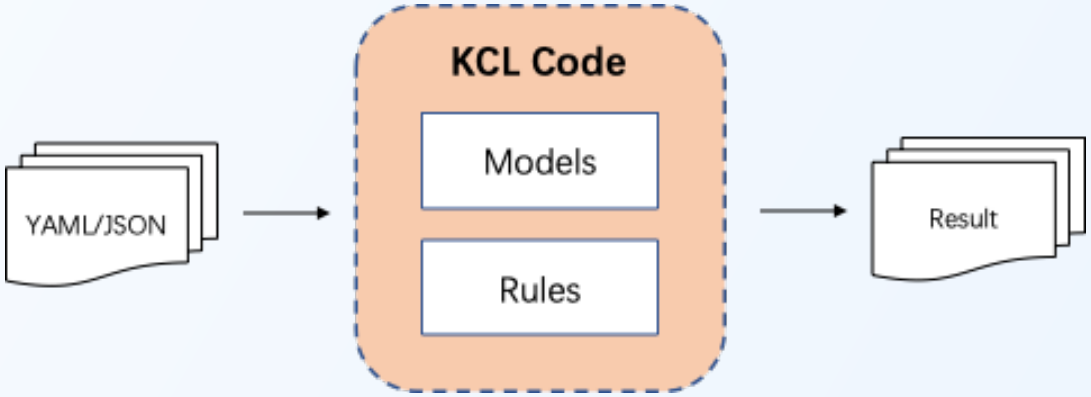
```
kubectl get po nginx -o yaml | grep kcl-operator
    managed-by: kcl-operator
```

- 少数几行 KCL 代码即可完成对应配置编辑/校验功能 (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
- 多种代码源支持 OCI, Git, Https, Filesystem, ...

https://github.com/kcl-lang/kcl-operator

Terraform Configuration Validation & Generation

Validation Process



- ✓ **多种数据支持**：JSON/YAML/...
- ✓ **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)

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

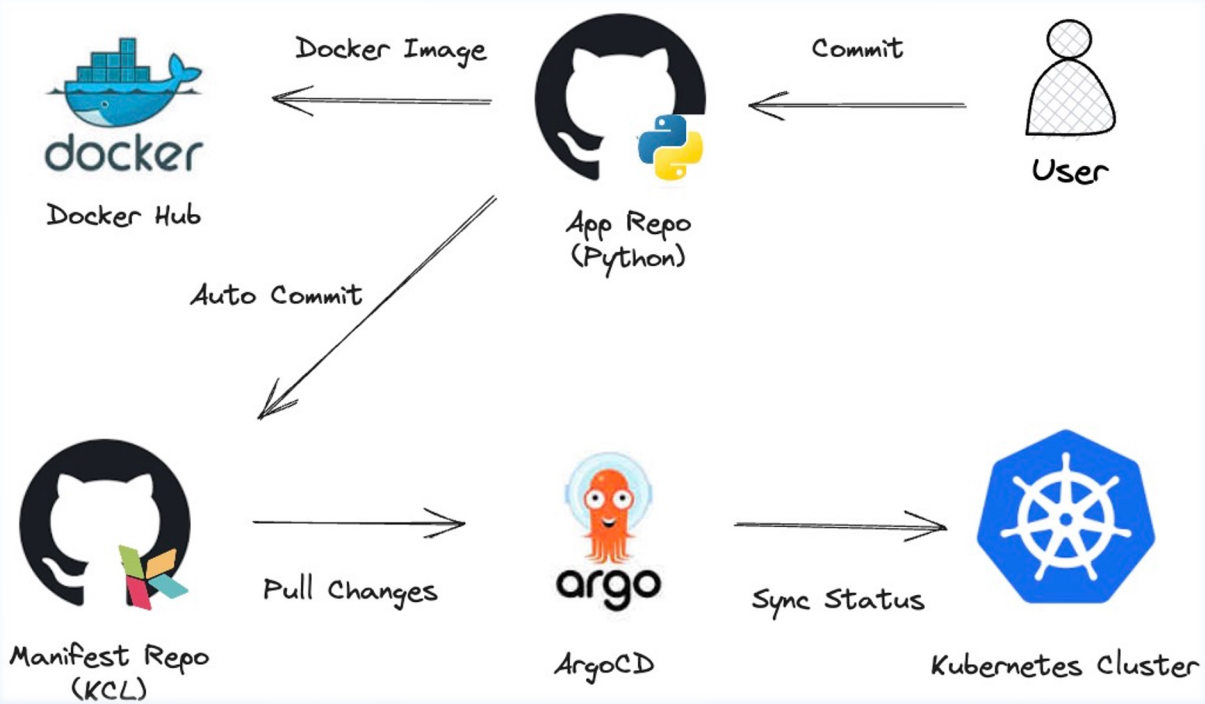
```
schema TFPlan:
    # Omit other attributes
    [...str]: any
    resource_changes?: [AcceptableChange]

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

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

IaC & GitOps

Better Configuration & Automation using KCL APIs



Commit

kcl code set image to kcllang/flask_demo:6428cff4309afc8c1c40ad180bb9...

...cfd82546be3e

main

github-actions[bot] committed 3 minutes ago

Showing 1 changed file with 1 addition and 1 deletion.

2 main.k

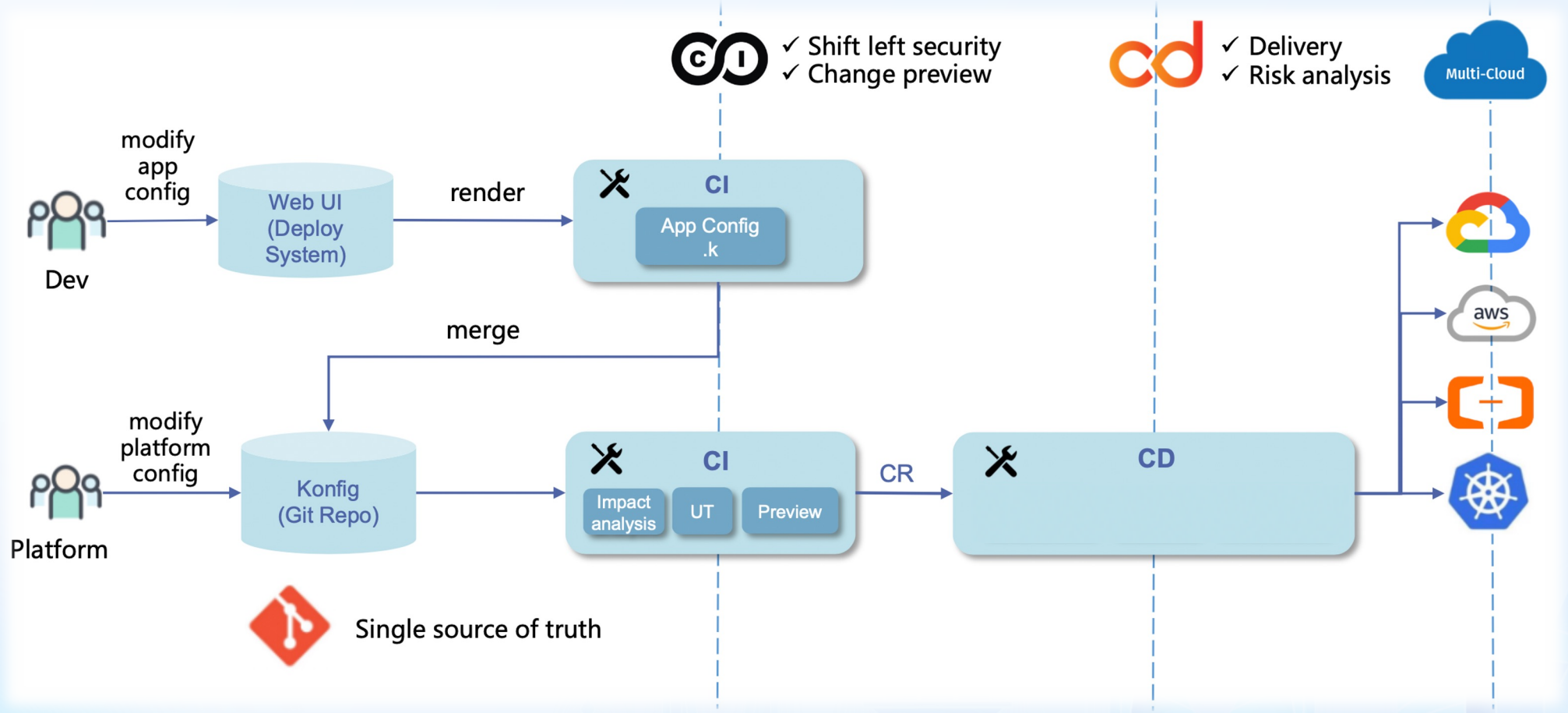
```
@@ -3,7 +3,7 @@ config = app.App {
3  3      name = "flask_demo"
4  4      containers: {
5  5          flask_demo = {
6     -            image = "kcllang/flask_demo:f1f2cbc0c4555d141e9f642fbd12edaf34d0b723"
   6  +            image = "kcllang/flask_demo:6428cff4309afc8c1c40ad180bb9cfd82546be3e"
7  7              ports = [{containerPort = 5000}]
8  8          }
9  9      }
```

配置驱动的工作流：同时支持 Standalone KCL 和 KRM KCL 配置格式
多种 CI/CD 和 GitOps 工具支持 e.g., ArgoCD

https://kcl-lang.io/docs/user_docs/guides/gitops/gitops-quick-start

App Delivery

For Kubernetes and Cloud Resources



多种支持应用交付引擎支持: 如 KusionStack, KubeVela

https://kusionstack.io
https://kubevela.net

App Delivery

Demo

Summary

Mutation, Validation, Abstraction
Production-Ready

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

- 从云原生配置策略及软件供应链层面解决***认知负担高、静态配置、效率和稳定性等问题***
- 通过定义合适的 API 抽象，配置编辑策略隐藏基础设施和平台细节，减轻开发人员的负担。
- 通过通过更现代的声明式配置策略语言和工具，KRM KCL 规范, OCI Registry 和 Artifact Hub 等，帮助不同团队/角色之间更轻松地共享、传播和交付配置和策略。（**欢迎共建配置和策略模型**👏）

More Resources

- **官方网站**
  - https://kcl-lang.io/
  - https://kusionstack.io/
- **GitHub**
  - https://github.com/kcl-lang
  - https://github.com/kusionStack
- **Twitter**
  - @kcl language
- **Slack**
  - CNCF KCL Slack Channel: https://cloud-native.slack.com/archives/C05TC96NWN8

钉钉(DingTalk ID 42753001)

微信公众号

KUBERNETES
COMMUNITY DAYS HANGZHOU

KUBERNETES

THANKS

感谢您的观看

COMMUNITY DAYS
HANGZHOU 2023

Appendix

Community Projects

Structured KV

Pros.

- Easy to write and read
- Rich multi-language API
- Various Path Tools

Cons.

- Redundant information
- Insufficient functionality e.g. abstraction, constraint, ...

Tech.

- JSON
- YAML

Product

- Kustomize
- ...

Templated KV

Pros.

- Simple config logic support
- Dynamic argument input

Cons.

- Increase of argument makes it difficult to maintain
- Insufficient functionality e.g. abstraction, constraint, ...

Tech.

- Velocity
- Go Template

Product

- Helm
- Helmfile
- ...

Programmable KV

Pros.

- Required programming features
- Code modularity
- Templates & Data abstraction

Cons.

- Insufficient type constraints
- Insufficient restraint ability
- Runtime error

Tech.

- GCL
- HCL
- Bicep
- Starlark
- Jsonnet
- CEL
- OPA/Rego
- ...

Product

- Terraform
- Tanka
- Radius
- ytt
- kpt

Typed KV

Pros.

- Rich config constraint syntax
- Unified type & value constraint
- Configuration conflict checking

Cons.

- Difficult to configuration override for multi-environment scenarios
- Runtime checks and limited performance

Tech.

- CUE
- Nickel
- ...

Product

- KubeVela
- ...

Modeled KV

Pros.

- Model-centric & constraint-centric
- Scalability on separated block writing with rich merge strategies
- Static type system & analysis
- High Performance

Cons.

- Expansion of different models requires investment in R&D

Tech.

- **KCL**
- ...

Product

- KusionStack
- KRM-KCL Tools and Operators