

# 使用 KCL 重塑配置管理界面

徐鹏飞 | KCL 项目 Maintainer

# 目录

01 背景

02 概念

03 架构

04 实践

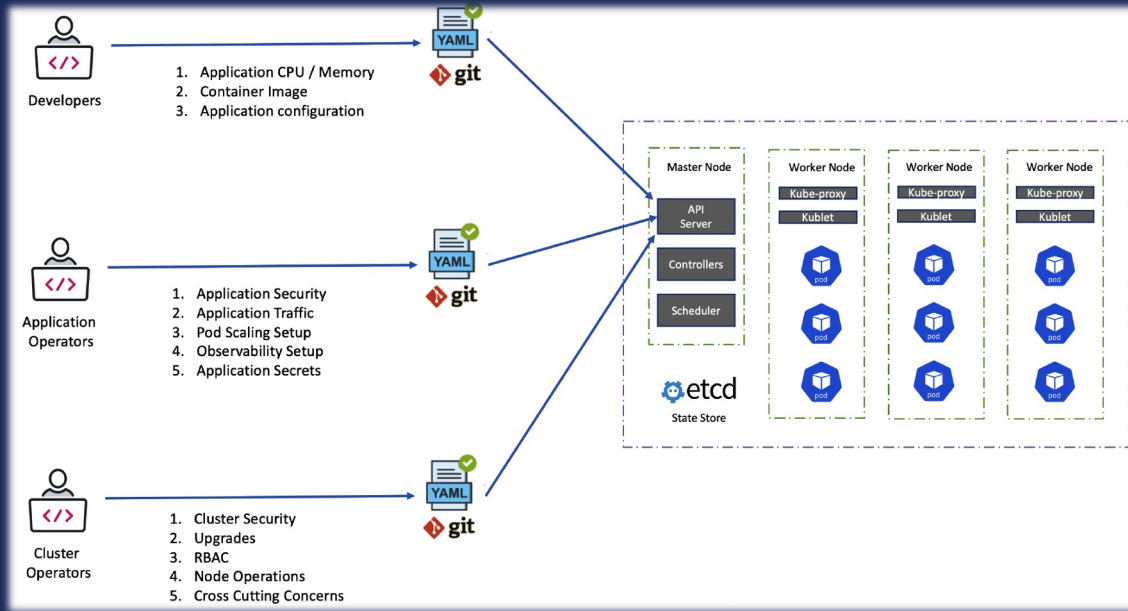
05 规划

# 背景

---

## 01

# 背景



## 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**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/1cLPGweVEYrVqQvBLq6sxV-TrE5Rm2MNOBA\\_cxZP2WU/edit#](https://docs.google.com/document/d/1cLPGweVEYrVqQvBLq6sxV-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/>



# 背景

- **屏蔽基础设施和平台细节，降低开发者认知负担**
  - 自定义抽象模型
  - 解决 YAML 或者模版配置维度爆炸问题
  - 可编程支持：逻辑、类型、函数、模块、...
- **无副作用地进行大规模配置管理**
  - 可扩展
  - 风险左移
  - 自动化
  - 高性能
  - 包管理支持
- **通过插件为已有的配置管理工具进行增强**
  - Helm
  - Kustomize
  - kpt
  - ...



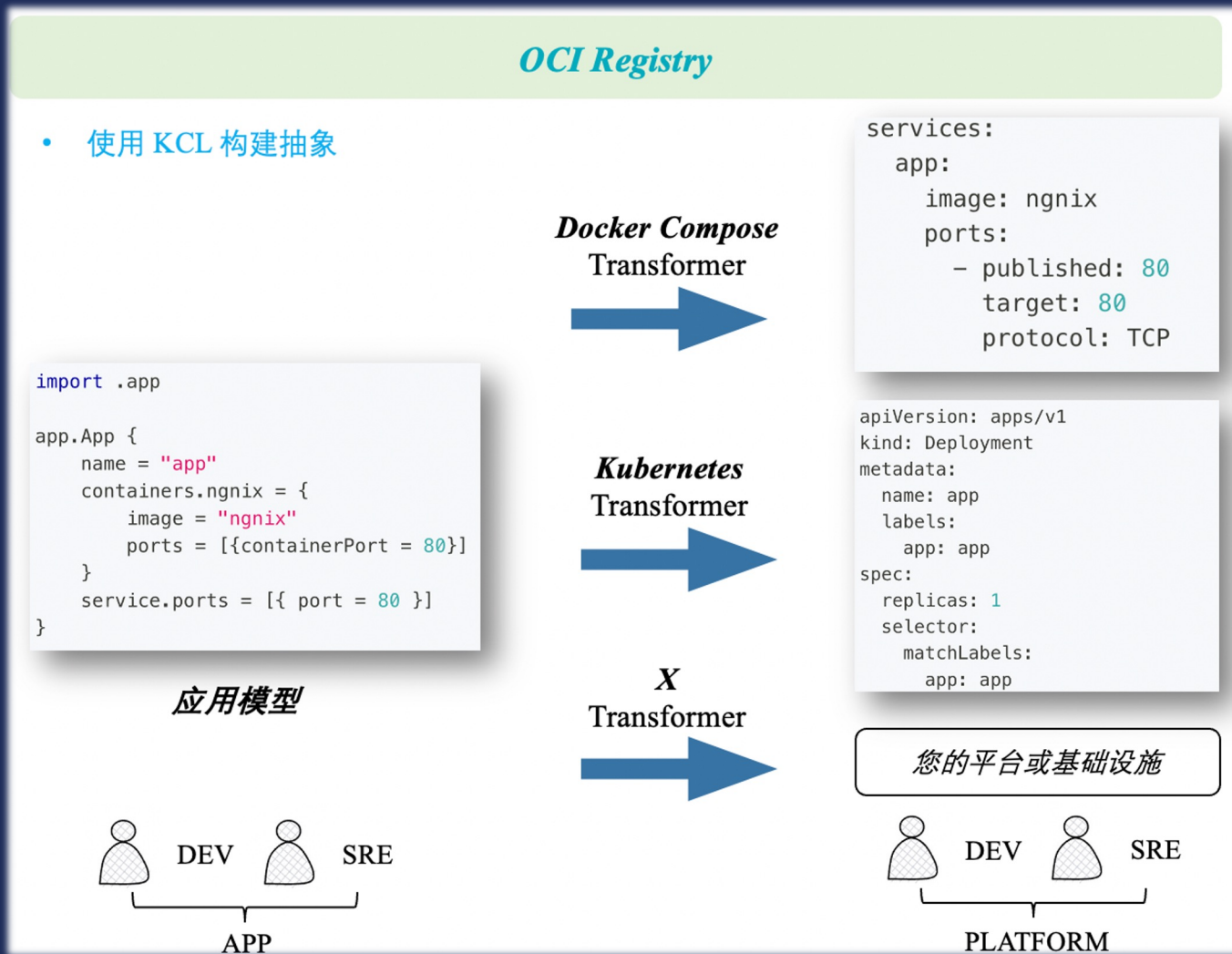
KCL 可以做什么

# 概念

---

## 02

## 配置 Standalone KCL 形式



- Validation: 通过 KCL Schema/Rule 等结构对配置进行约束校验
- Mutation: 通过 KCL 函数对资源进行变换或注入配置

- ✓ 开发者可以理解的声明式应用配置模型
- ✓ 动态配置管理 DCM: 可编程可扩展
- ✓ 关注点分离: 应用/平台 Dev/SRE
- ✓ 风险左移: 实时的配置错误提示
- ✓ 标准的 OCI 配置交付和包管理支持: KPM

# 配置 KRM KCL 形式

## • Mutation

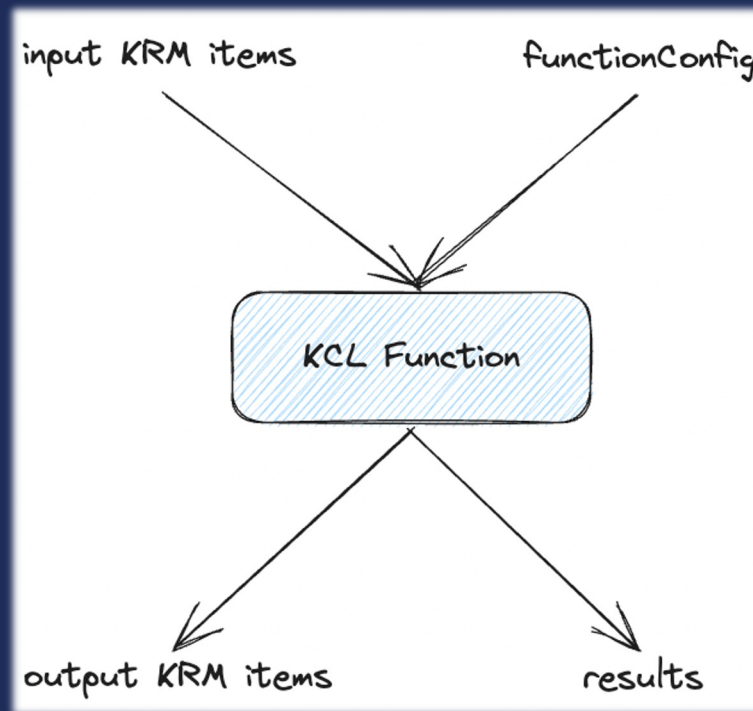
```
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: set-annotations
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: mutation
      documentation: >-
        Add or change annotations
spec:
  params:
    toAdd: addValue
    source: oci://ghcr.io/kcl-lang/set-annotation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

## • Validation

```
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
```

## • Abstraction

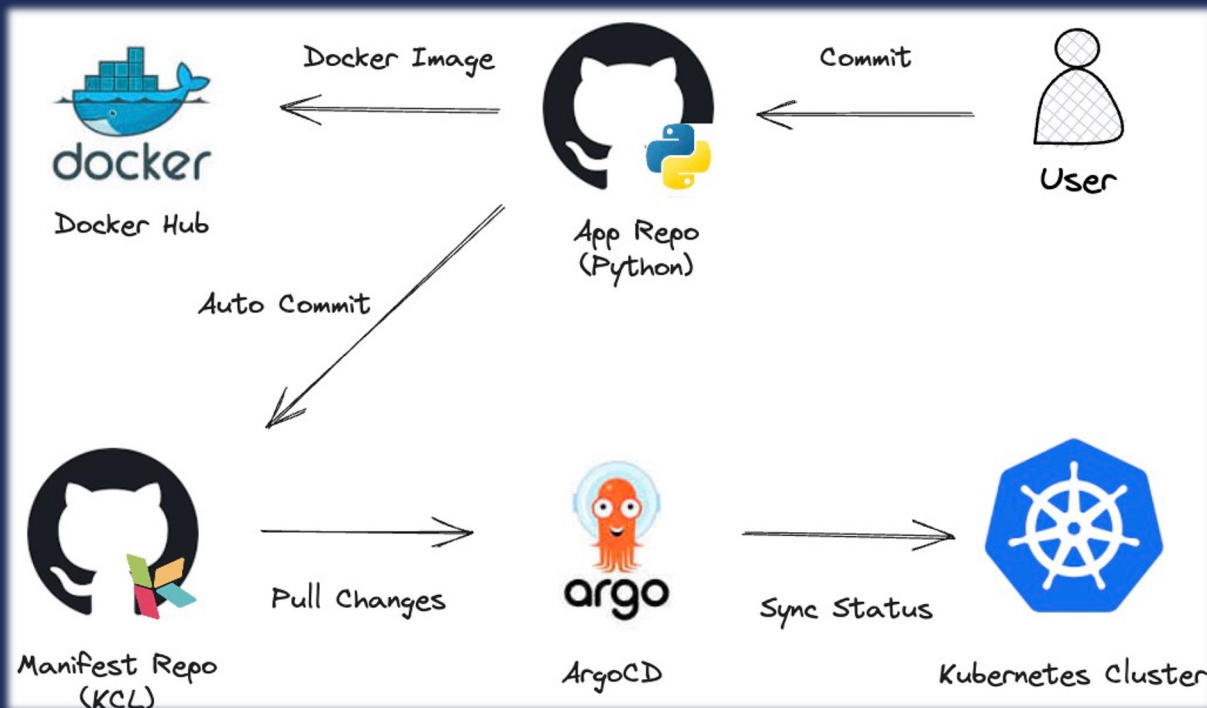
```
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: web-service
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: abstraction
      documentation: >-
        Web service application abstraction
spec:
  params:
    name: app
  containers:
    nginx: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



- 遵循统一的 KRM Function 规范
- 可编程可扩展
- 多种代码源支持: OCI, Git, Https, ...



# 自动化



```
Commit

kcl code set image to kcclang/flask_demo:6428cff4309afc8c1c40ad180bb9...
...cfd82546be3e

main

github-actions[bot] committed 3 minutes ago

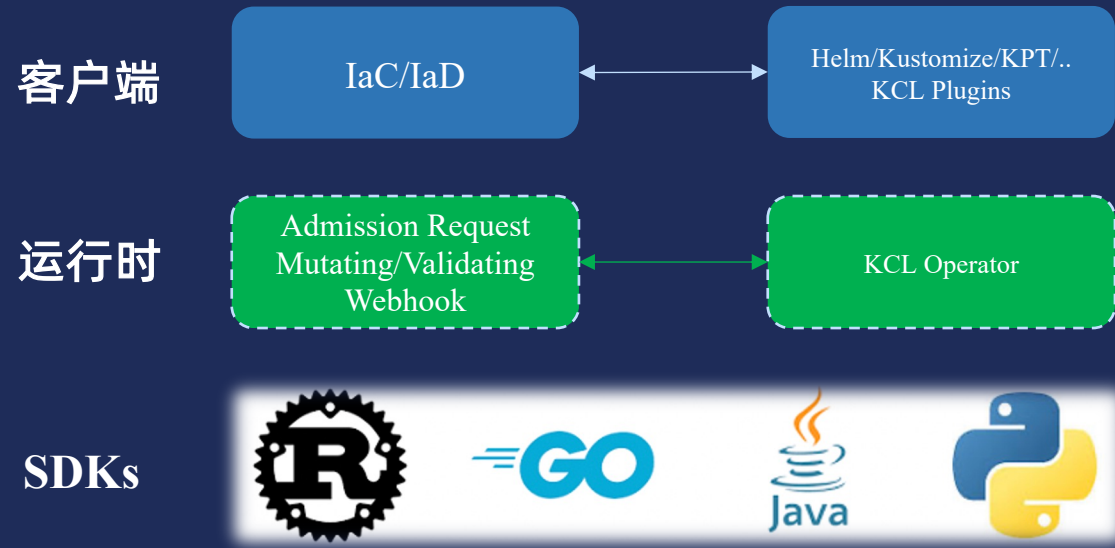
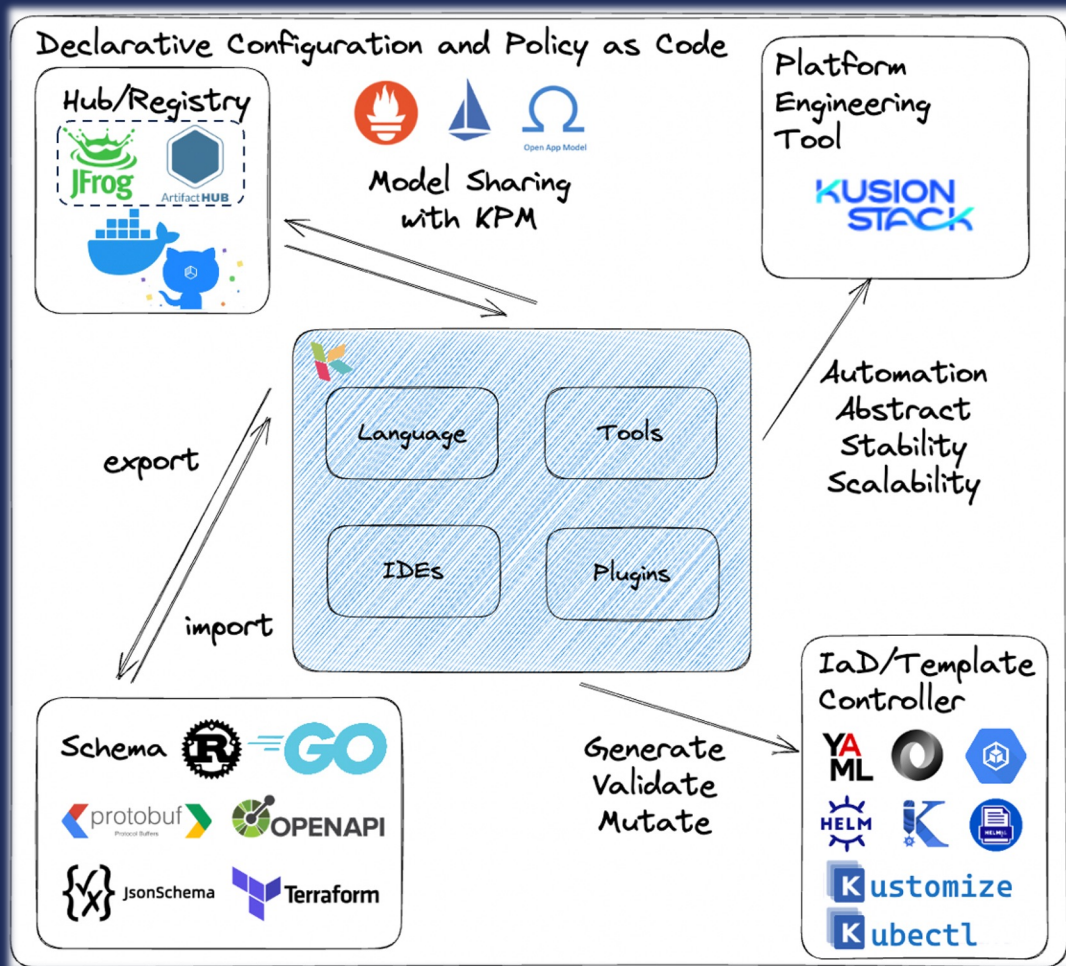
Showing 1 changed file with 1 addition and 1 deletion.

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

[https://kcl-lang.io/docs/user\\_docs/guides/gitops/gitops-quick-start](https://kcl-lang.io/docs/user_docs/guides/gitops/gitops-quick-start)

# 集成



- 多语言 SDK: Go, Rust, Python, Java
- 包管理支持: KPM 工具和多种 Registry 支持
- 数据和 Schema 集成: KCL Import 工具
- 运行时集成: 使用 KCL Operator 而不是重复开发 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

# 架构

---

03

# 工作空间

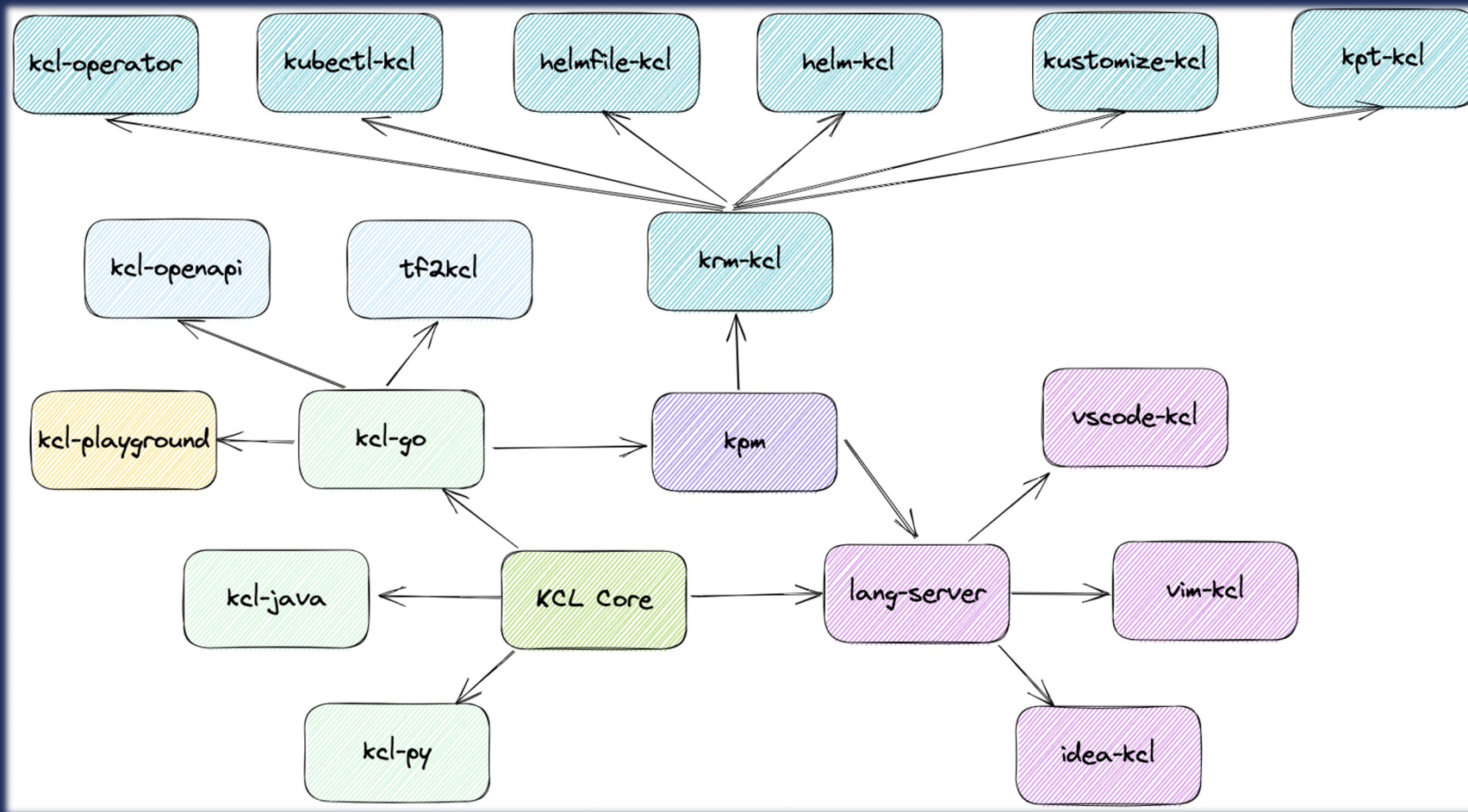
## Language + Tools + IDEs + SDKs + Plugins

The screenshot shows an IDE interface with a file explorer on the left, a code editor in the center, and a tool palette on the right. The code editor displays KCL code for a ClickHouse operator. The tool palette includes a KCL Package Manager, a KCL Coding Assistant with buttons for Highlight, Format, Go To Def/Ref, Compile, Completion, Debug, Error/Warning Checking, and Test, an LSP (Language Server Protocol) layer, a KCL Language Server, and a KCL Compiler. Below the IDE, a 'Tools & CI/CD Engagement' diagram shows a workflow: kcl-format (with a green checkmark) → kcl-lint (with a green checkmark) → kcl-test (with a green checkmark) → kcl-doc (with a green checkmark). Blue arrows point from the IDE to the workflow diagram.

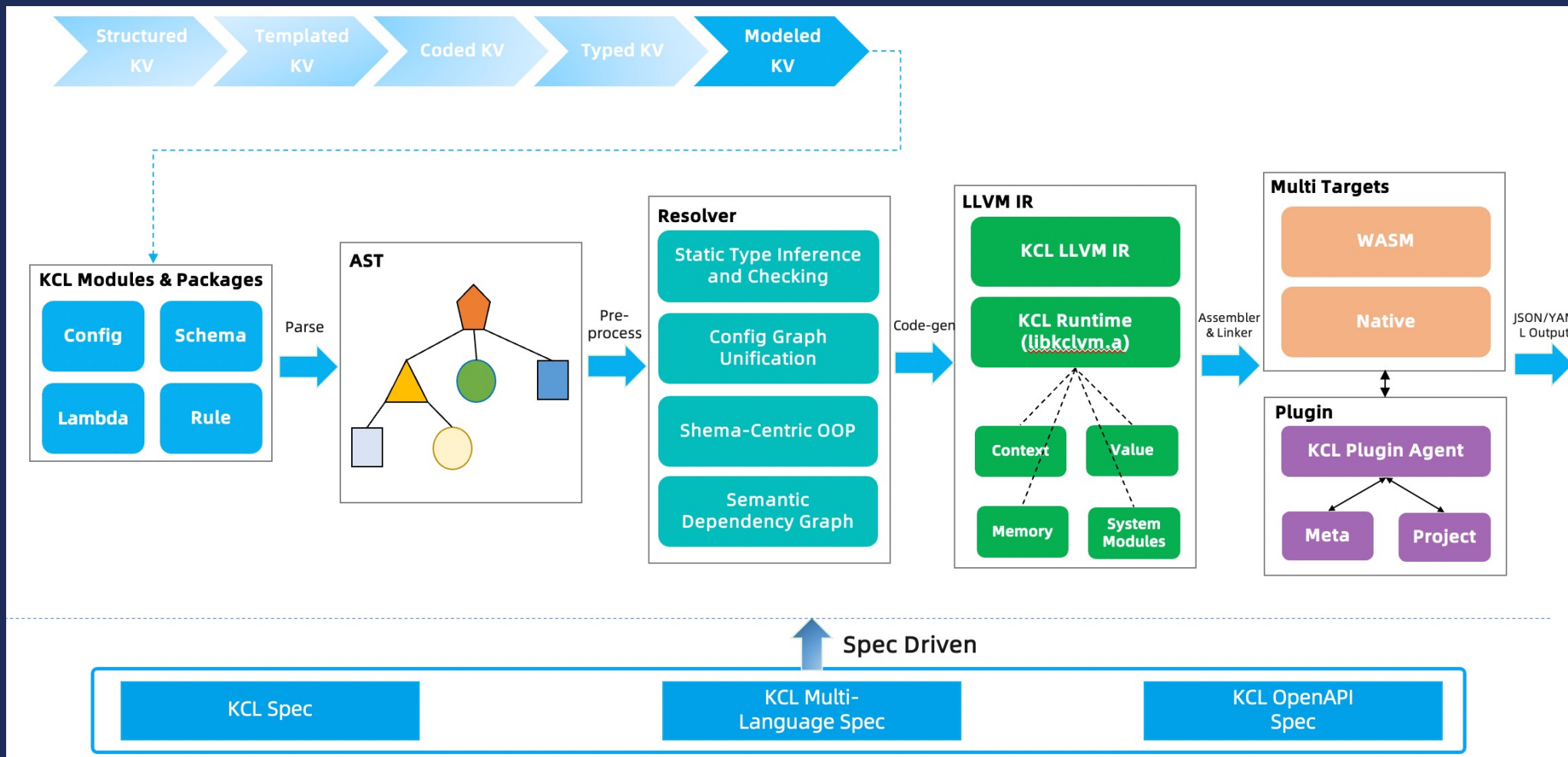
```
1 import base.pkg.kusion_models.kube.frontend
2
3 # The application configuration in stack will overwrite
4 # the configuration with the same attribute in base.
5 server: frontend.Server {
6   # spec.template.spec.containers[0], main container
7   image = "altinity/clickhouse-operator:0.19.2"
8
9   # 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
```



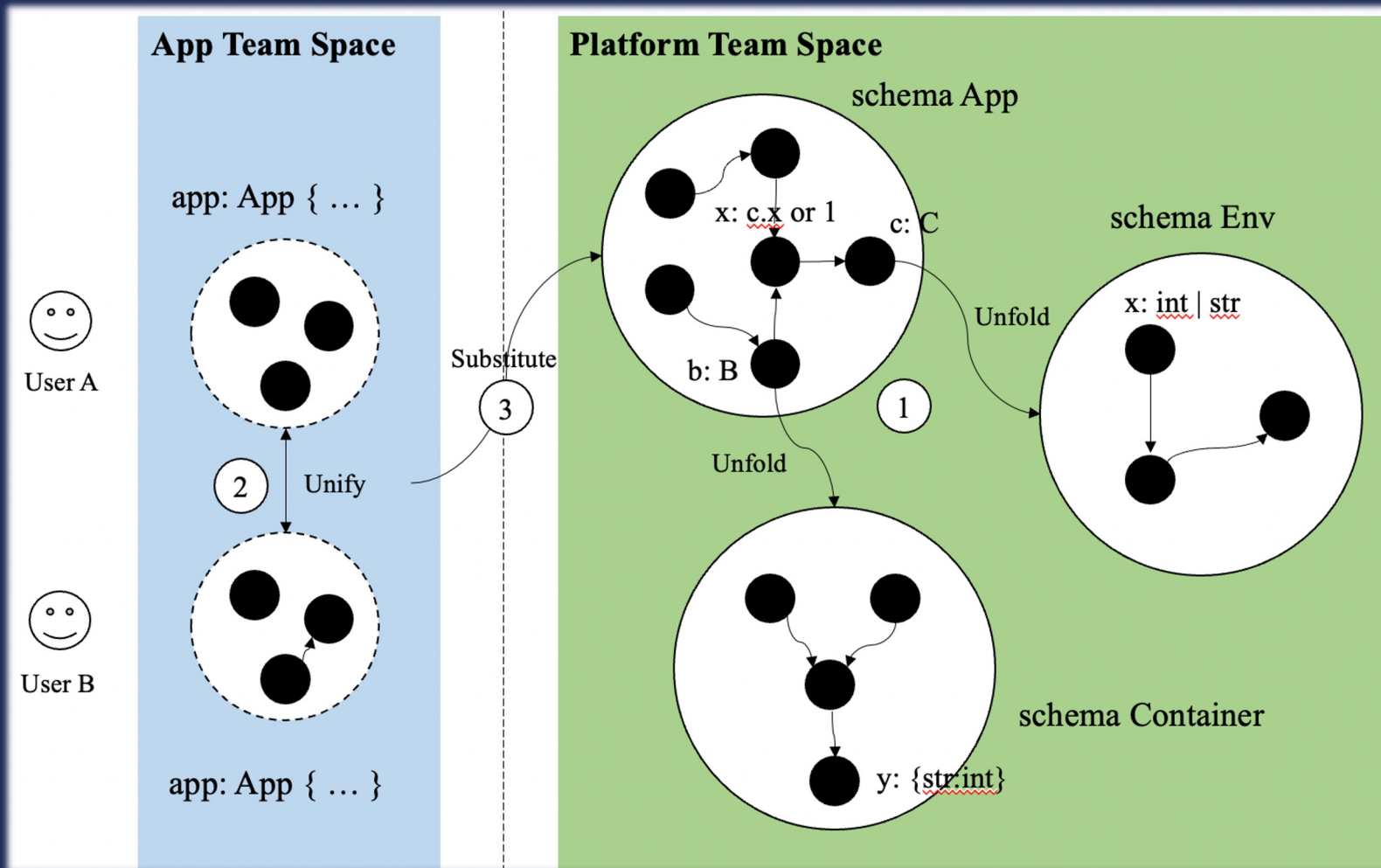
# 组件



# 技术架构

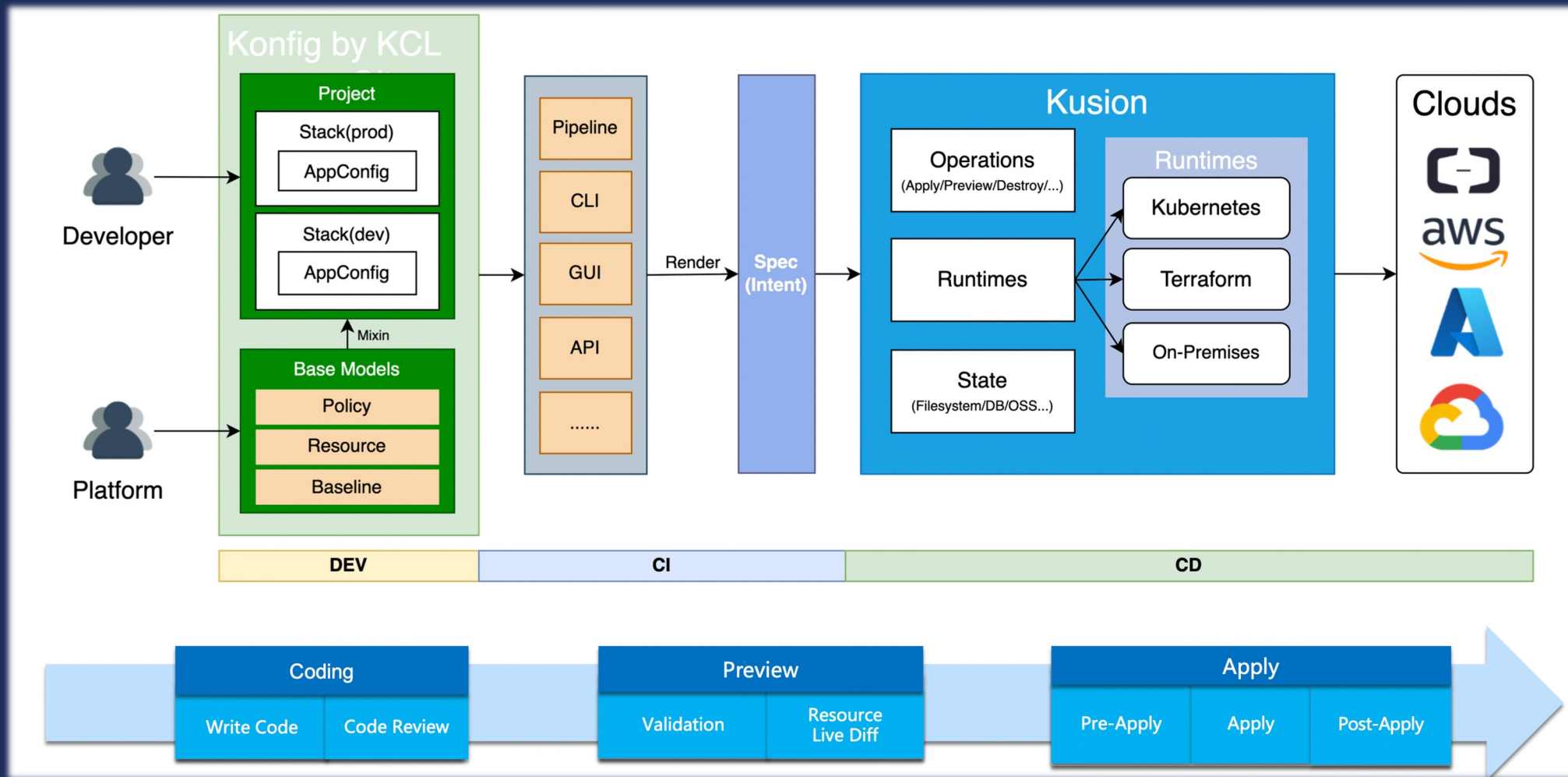


# 配置图模型



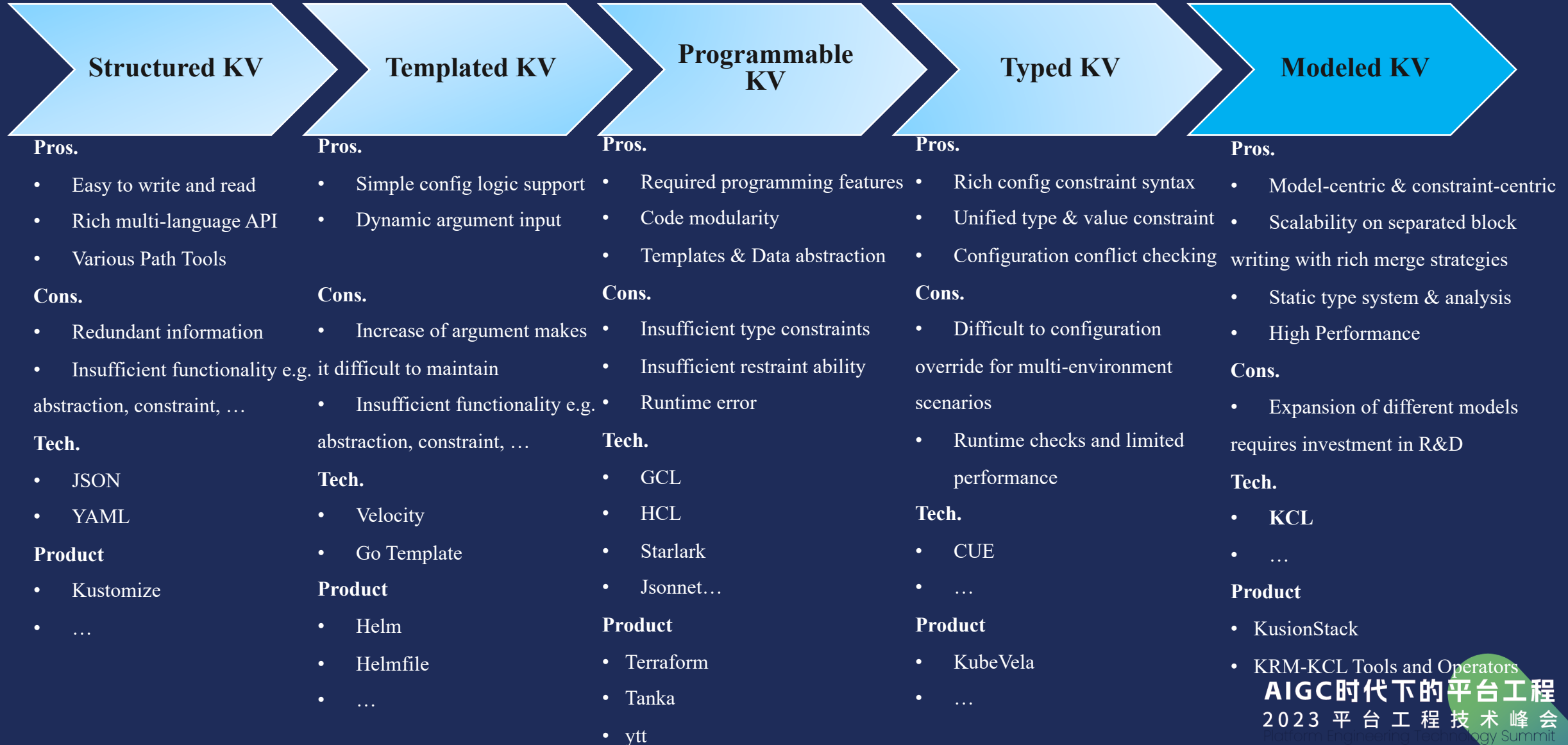
- ✓ **配置生成**
  - ✓ 自动合并
  - ✓ 多种合并策略支持
- ✓ **配置变换**
  - ✓ 数据/Schema 集成
  - ✓ CRUD API
- ✓ **配置校验**
  - ✓ 静态类型
  - ✓ 约束规则
  - ✓ 不可变性
- ✓ **配置抽象**
  - ✓ 关注点分离

# Kusion 引擎





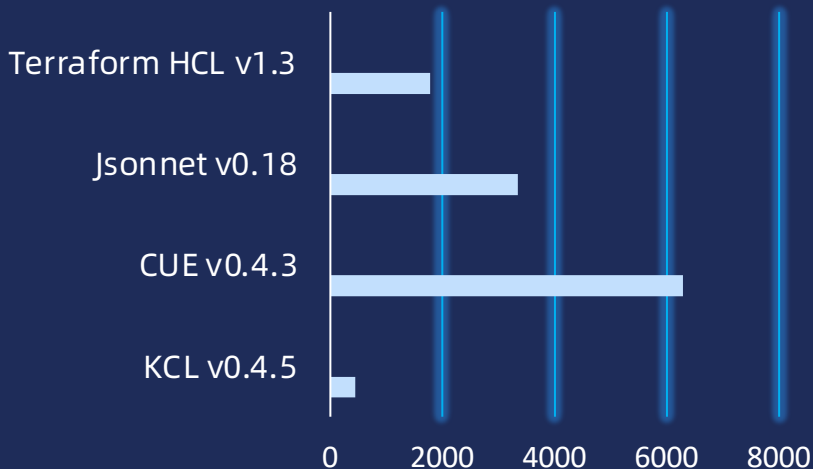
# 社区项目



# 性能对比

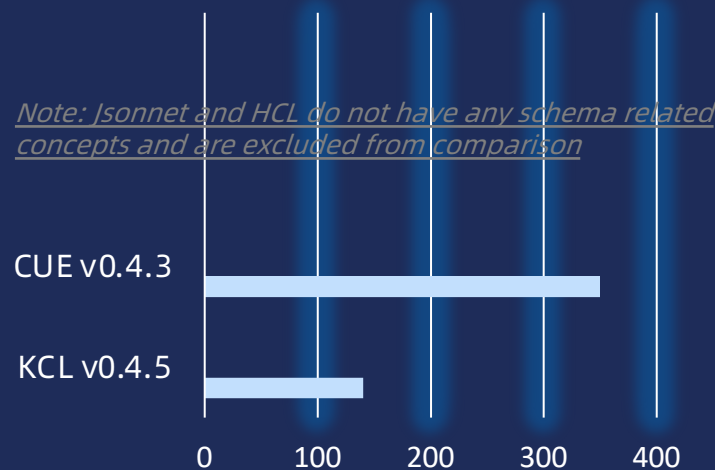
## Loop and Function

```
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 {}
```



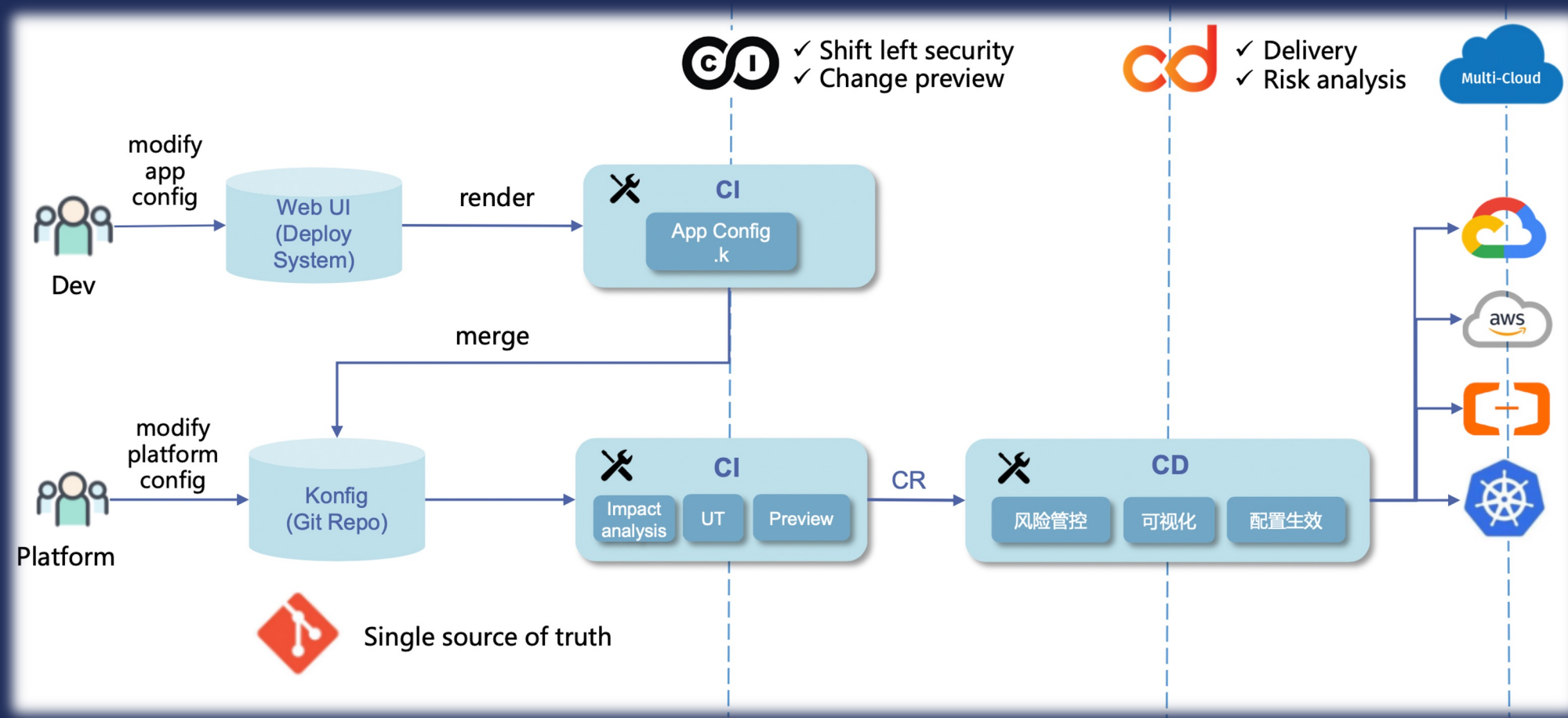
Test environment: single core macOS 10.15.7 CPU: i7-8850H 2.6GHz 32GB 2400Mhz DDR4 No NUMA, e2e run time (ms)

# 实践

---

## 04

# 实践





# 实践

```
import base.pkg.kusion_models.kube.frontend

appConfiguration: frontend.Server {
  image = "howieyuen/gocity:latest"
}
```



```
schema ServerBackend(inputConfig: server.Server):
  """ServerBackend converts the user-written front-end model 'Server' into a
  collection of kubernetes resources and places the resource collection into
  the 'kubernetes' attribute.
  """
  mixin [
    # Resource builder mixin
    mixins.NamespaceMixin,
    mixins.ConfigMapMixin,
    mixins.SecretMixin,
    mixins.ServiceMixin,
    mixins.IngressMixin,
    mixins.ServiceAccountMixin,

    # Monitor mixin
    mixins.MonitorMixin
  ]

  # Store the input config parameter, ensure it can be seen in protocol and
  config: server.Server = inputConfig

  # Workload name.
  workloadName: str = "{}".format(metadata.__META_APP_NAME, metadata.__META_APP_NAMESPACE)
  # App variable contains labels, selector and environments.
  app: utils.ApplicationBuilder = utils.ApplicationBuilder {}
  # Main containers and sidecar containers.
  mainContainer: (str)
  sidecarContainers?: [(str)]
  initContainers?: [(str)]

  if config.mainContainer:
    assert config.image, "config.image must be specified and can't be empty."
    # Construct input of converter using the volumes.
    mainContainer = utils.VolumePatch(config.volumes, [utils.ContainerFrontendBuilder {
      name = config.mainContainer
      if config.mainContainer.useBuiltInEnv:
        env = app.envs
        name = config.mainContainer.name or "main"
        image = config.image
        resource = config?.schedulingStrategy?.resource
    }])?@

  if config.sidecarContainers:
    sidecarContainers = utils.VolumePatch(config.volumes, [utils.ContainerFrontendBuilder {
      name = config.sidecarContainers.name
      image = config.image
      resource = config?.schedulingStrategy?.resource
    }])?@

  if config.initContainers:
    initContainers = utils.VolumePatch(config.volumes, [utils.ContainerFrontendBuilder {
      name = config.initContainers.name
      image = config.image
      resource = config?.schedulingStrategy?.resource
    }])?@

  # Construct workload attributes.
  workloadAttributes: (str) = {
    metadata = utils.MetadataBuilder(config) | {
      name = workloadName
    }
  }
  spec = {
    replicas = config.replicas
    if config.useBuiltInSelector:
      selector.matchLabels: app.selector | config.selector
    else:

```



- Spec
- Deployment
- Service
- ConfigMap
- Database
- Monitor
- ...

Front-end model ( Developer )

Back-end model ( Platform )

K8s/clouds/on-prem resources

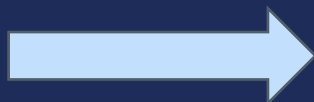
# 实践

```
→ bin git:(watch-ui) ./kusion apply -w /Users/yuanhao/opensource/KusionStack/konfig/base/examples/server/app_service/prod --watch --yes  
✓ Compiling in stack prod...
```

# 实践

```
9  > db/
10 > kube-manifests/
11   kube-manifests/
12   kube-manifests/
13   kube-manifests/
14 > lib/
15 > log/
16 > node_modules/
17 > ops/scripts/
18   __init__.py
19   db_migrate.sh*
20   gen_data.sh*
21   gen_pod_env.py
22   gen_sed_cmd.py
23   merge_key.py
24   secrets_manager_utils.py
25 > public/
26 > script/
27 > spec/
28 > storage/
29 > tmp/
30   ALLOWED_PASSWORDS
31   app.json
32   aws_migration_notes.md
33   babel.config.js
34   bitbucket-pipelines.yml
35   buildspec.yml
36   CHANGELOG.md
37   cms-pre-production-23070
38   cms-pre-production-23070
39   cms-pre-production-23070
40   cms-pre-production-23070
41   cms-pre-production-23070
42   cms-pre-production-23070
43   cms-pre-production-23070
44   cms-pre-production-23070
45   cms-prod-230705-buildspe
46   cms-prod-230705-buildspe
47   cms-prod-230705-dockerfi
48   gtyuchen/fuse/FuseIgnited-cms
49   ops/scripts/secrets_manager_utils.py

1 import os
2 from os import listdir
3
4 import ruamel.yaml
5
6 yaml = ruamel.yaml.YAML()
7 yaml.preserve_quotes = True
8 yaml.explicit_start = True
9
10 def main():
11     home_dir = os.path.join(os.path.dirname(__file__), "../kube-manifests")
12     print(home_dir)
13     for manifest in listdir(home_dir):
14         if not manifest.endswith(".yaml"):
15             continue
16         p = os.path.join(home_dir, manifest)
17         with open(p, "r") as f:
18             content = yaml.load_all(f)
19             res = []
20             for item in content:
21                 if item["kind"].lower() == "deployment":
22                     cs = item["spec"]["template"]["spec"]["containers"]
23                     for c in cs:
24                         c["env"].append({
25                             "name": "test_name",
26                             "value": "test_value"
27                         })
28                 res.append(item)
29             print(res)
30             with open(p, "w") as f:
31                 yaml.dump_all(res, f)
32
33 if __name__ == '__main__':
34     main()
```



- Code

```
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
spec:
  params:
    env:
      - name: test_name
        value: test_value
  source: |
    [r | {if resource.kind == "Deployment": spec.template.spec.containers: [{
      env += option("params").env
    }] * len(r.spec.template.spec.containers)} for r in option("items")]
```

- OCI

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

# 实践

```
OpenAPI spec schemas in KCL - https://kcl-lang.io/
openapi.k
1  """
2  Bindings for OpenAPI 3.0.0 - https://swagger.io/specification/#schema-object
3
4  JSON Schema validation for OpenAPI 3.0.0 must conform to this version:
5  https://datatracker.ietf.org/doc/html/draft-wright-json-schema-validation-00
6  """
7
8  schema Ref:
9    $ref: str
10
11 schema Contact:
12   """
13   https://swagger.io/specification/#contact-object
14   """
15   name?: str
16   url?: str
17   email?: str
18
19 schema Description:
20   description?: str
21
22 schema SchemaString(Description):
23   type: "string" = "string"
24   enum?: [str]
25   default?: str
26   format?: "byte" | "binary" | "date" | "date-time" | "password"
27
28 schema SchemaNumber(Description):
29   type: "number" = "number"
30   minimum?: float
31   maximum?: float
32   exclusiveMinimum?: bool
33   exclusiveMaximum?: bool
34
35 schema SchemaInteger(Description):
36   type: "integer" = "integer"
37   minimum?: int
```

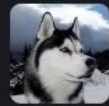


- ✓ JSON/YAML 支持
- ✓ 双向 Open API 转换支持
- ✓ 自定义错误信息

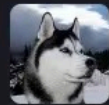
<https://gist.github.com/yawaramin/6f56f4b053d2b090353d7b7d40a722ce>



# 用户声音



**Prahalaad Ramji** 2 months ago  
amazing sounds good 👍



**Prahalaad Ramji** 2 months ago  
so yea, i'm in the process of doing a one2one migration of what i have in cue and seeing where i hit blocks.

## Support KCL as a language runtime? #13722

yawaramin started this conversation in Ideas



**yawaramin** last week

Hi, currently Pulumi supports several languages for writing configuration. Would it be possible to add **KCL**?

KCL is a statically typed language (implemented in Go) that allows defining configuration that follows schemas (type > 10 ), and rules/policies. It currently compiles directly to YAML, so it is possible to use it indirectly with Pulumi.

KCL is a full-fledged language which specifically targets configuration. It has a typechecker, linter, formatter, packager, complete toolchain. KCL packages can be uploaded to an OCI registry and common configuration (types) can be easily maintained in a central location. Typechecking is near-instant and the feedback loop is very tight. But by not being a general purpose language, KCL avoids many pitfalls especially in security.

My question is just if there would be interest from the Pulumi project to support it directly as a language runtime and not as a plugin. It would look like. (We know that it would work, because KCL can express anything YAML can.)



**magick93** 4 hours ago

Hello

I'm currently relying heavily on CUE and Hof (<https://github.com/hofstadter-io/hof>) for generating source code.

I've recently discovered kcl-lang, and am very impressed by the project for many reasons.

I'd like to suggest that, while there is obvious value in applying kcl to configuration, it can also be used to generate source code. Consider it like uml - it can describe every aspect of source code - and then, when combined with a text generator, be used to generate source code and artifacts.

For example, generating:

- database ddl
- svelte / react / vue web components
- finite state machines
- etc etc

To achieve this it would be good if there was ability to generate kcl models from existing models, for example, import json, jsonschema, yaml, and ideally cue.

Similarly, export structs in Rust, Go, Java etc, for use in existing templating.



**nkabir** 8 minutes ago **Author**

That will work. Thank you. KCL is fantastic.



Write a reply



**subbed\_** +1 · 1 hr. ago

I'm getting some HCL and Jsonnet vibes, I like it.



更多玩法, 欢迎加入 KCL 社区

# 社区



2022年5月开源

# 规划

---

05

# 技术规划



下一步是什么？

# AI Based Programming Language for the Cloud?

$AIGC \approx AI + DSL + Engine + CoT$

平台/云能力**抽象**封装成 DSL 统一 App 和 Infra

**引擎**消费 DSL 输入

将 DSL 的组合方式（链式思维的方式，CoT）提炼成 Prompt 喂给 AI 模型并返回结果



## 其他资源

### • 官方网站

- <https://kcl-lang.io/>
- <https://kusionstack.io/>

### • GitHub

- <https://github.com/kcl-lang>
- <https://github.com/KusionStack>

### • Twitter

- [@kcl language](https://twitter.com/kcl_language)
- [@kusionstack](https://twitter.com/kusionstack)

### • Slack

- <https://kcl-lang.slack.com>
- <https://kusionstack.slack.com>

[钉钉\(DingTalk ID 42753001\)](#)



微信搜一搜

规模化云原生运维



# Thank You!

**AIGC时代下的平台工程**  
2023 平台工程技术峰会