KCL: A constraint-based record & functional language mainly used in cloud-native configuration and policy scenarios.

Pengfei Xu (xpf6677@gmail.com)

July 2023

Agenda

01 Background

02 Design

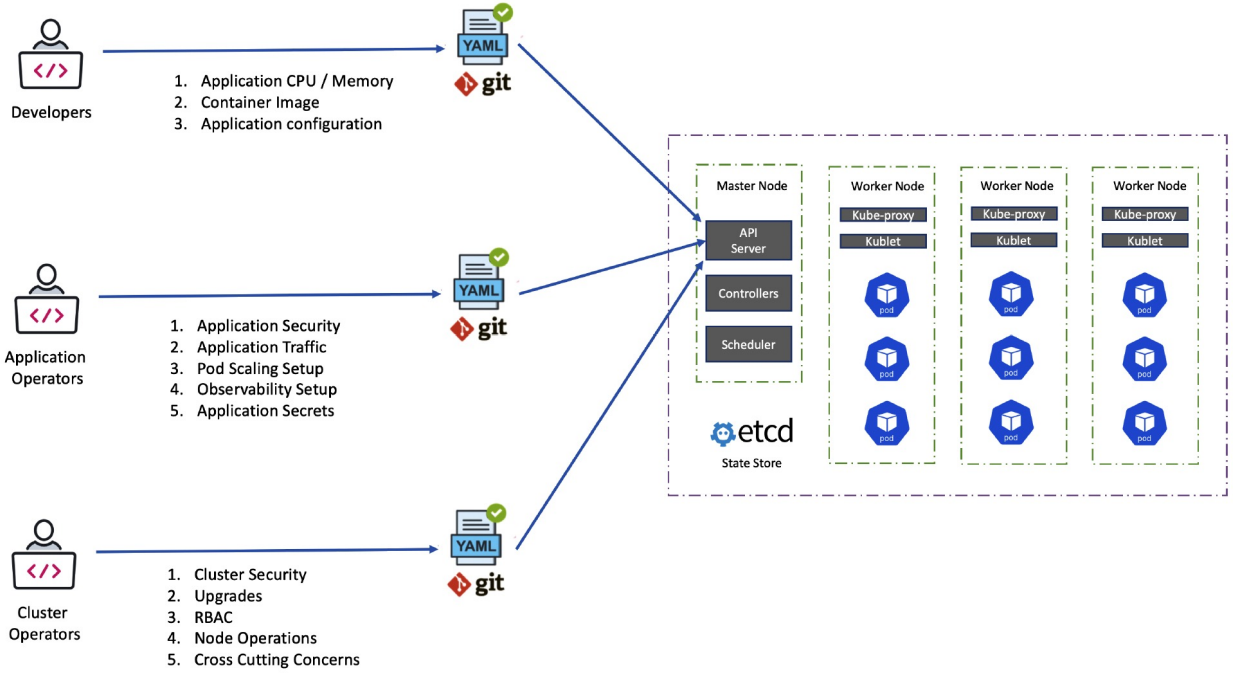
03 Scenarios

04 Evaluation

Background

01

Background



Cognitive Loading

- Complex infrastructure/platform concepts (500+ models, 2000+ fields)
- Complex application YAML configuration (10000+ lines)

Static Config

- Fragmented configuration dimension explosion (multi environment, multi tenant)
- Difficulty in customizing third-party application configurations

Low Efficiency/Reliability

- No unified configuration engineering technology stack, lack of standardized testing and validation methods,

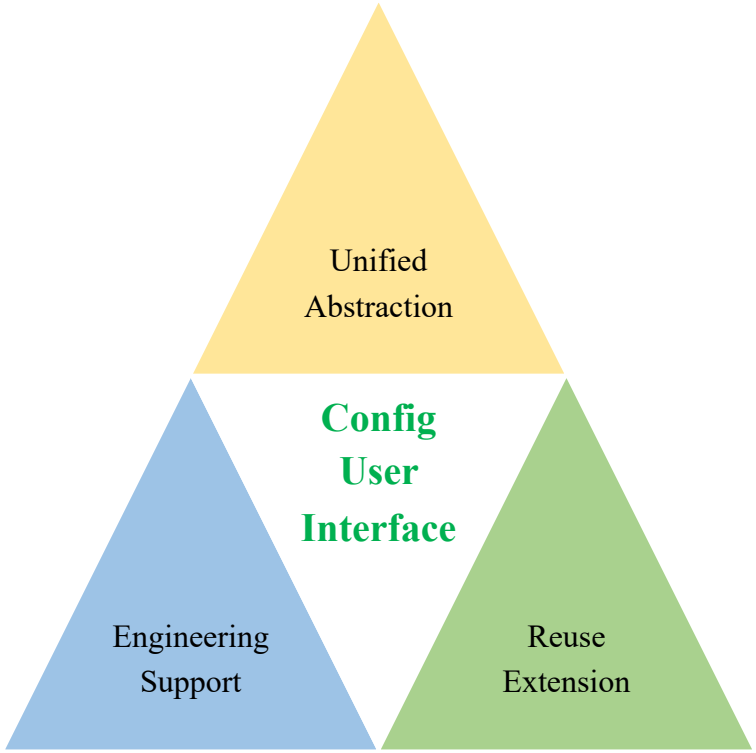
Reduce the burden of infrastructure for developers and improve the efficiency of configuration management

Declarative Application Management in Kubernetes: https://docs.google.com/document/d/1cLPGweVEYrVqQvBLJg6sxV-TrE5Rm2MNOBA_cxZP2WU/edit#
Google SRE Workbook: https://sre.google/workbook/configuration-specifics/

Thinking

Problem Abstraction: Large-scale Configuration Management

Problem Decomposition: The core of configuration management problems is **the problem of configuration language**, and the upper level automation systems and business processes revolve around language



Mask infrastructure and platform details through **reshaping the configuration interface and dynamic configuration management**

Declarative Application Management in Kubernetes: https://docs.google.com/document/d/1cLPGweVEYrVqQvBLJg6sxV-TrE5Rm2MNOBA_cxZP2WU/edit#
Google SRE Workbook: https://sre.google/workbook/configuration-specifics/

Why KCL

- **Hide infrastructure and platform details to reduce the burden of developers.**
  - Abstraction
  - Solve issues on YAML/Template bloat
  - Language enhancement: logic, type, function and package.

- **Large-scale configuration management without side effects cross teams.**
  - Stability
  - Scalability
  - Automation
  - High performance
  - Package distribute and sharing

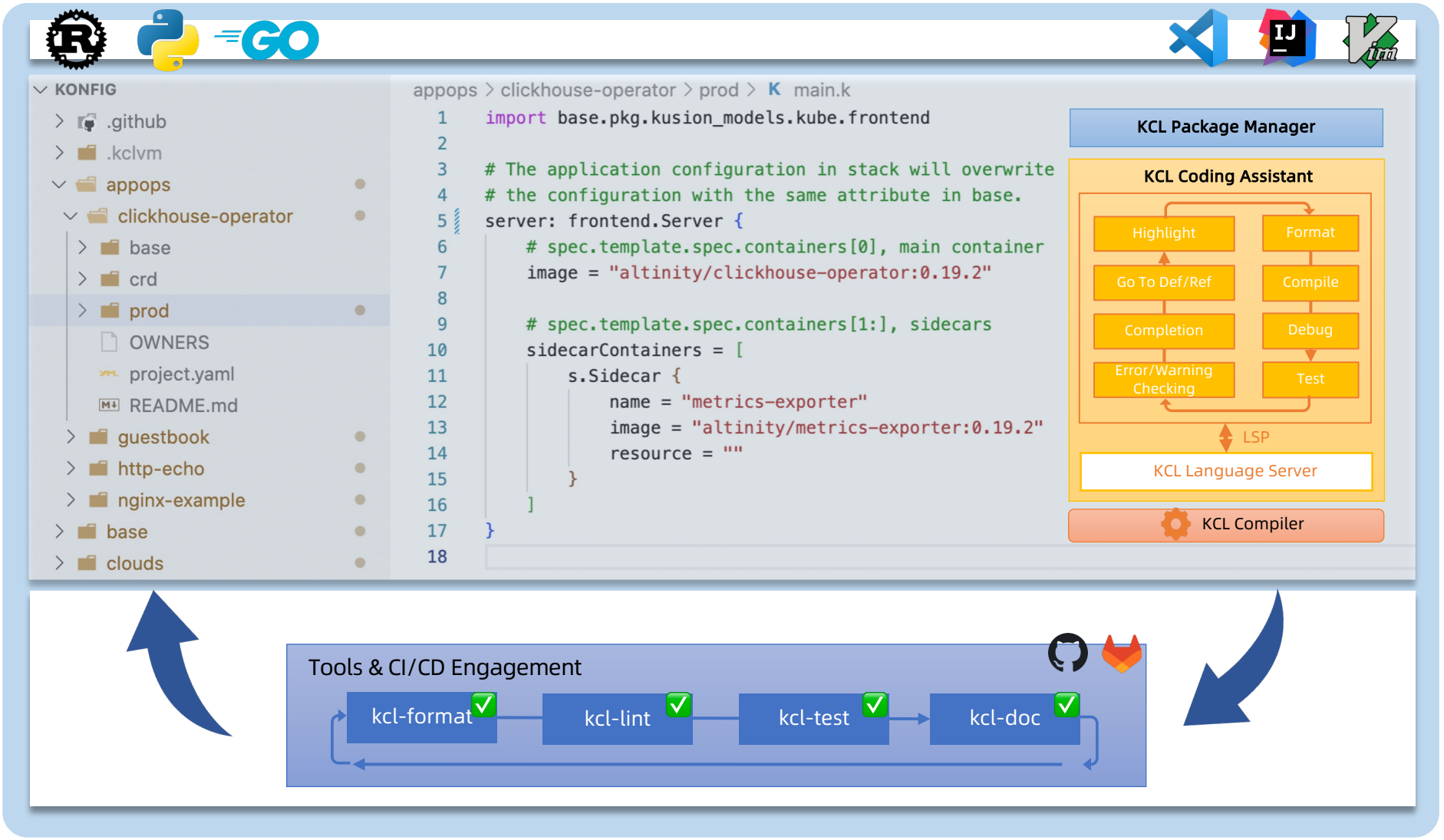
- **Enhancement for configuration tools e.g., Helm, Kustomize.**
  - Mutating
  - Validating

Design

02

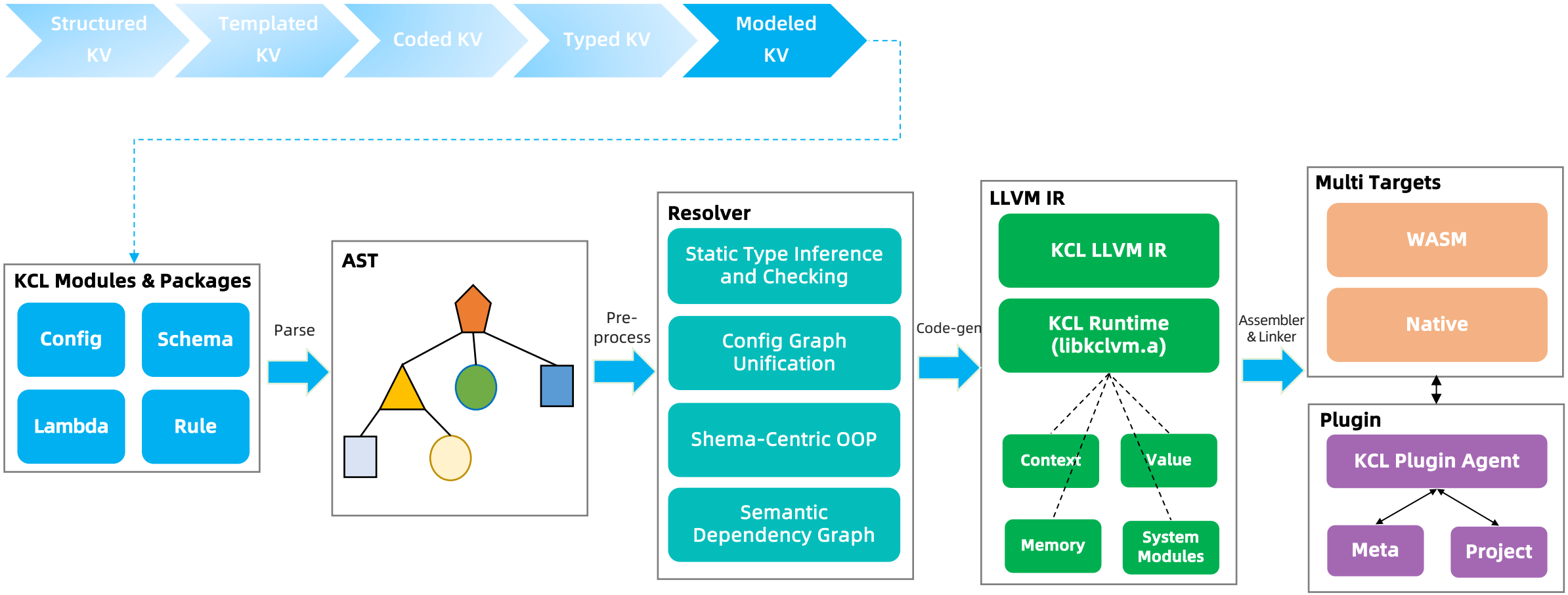
Overview

Language + Tools + IDEs + SDKs + Plugins



https://github.com/kcl-lang

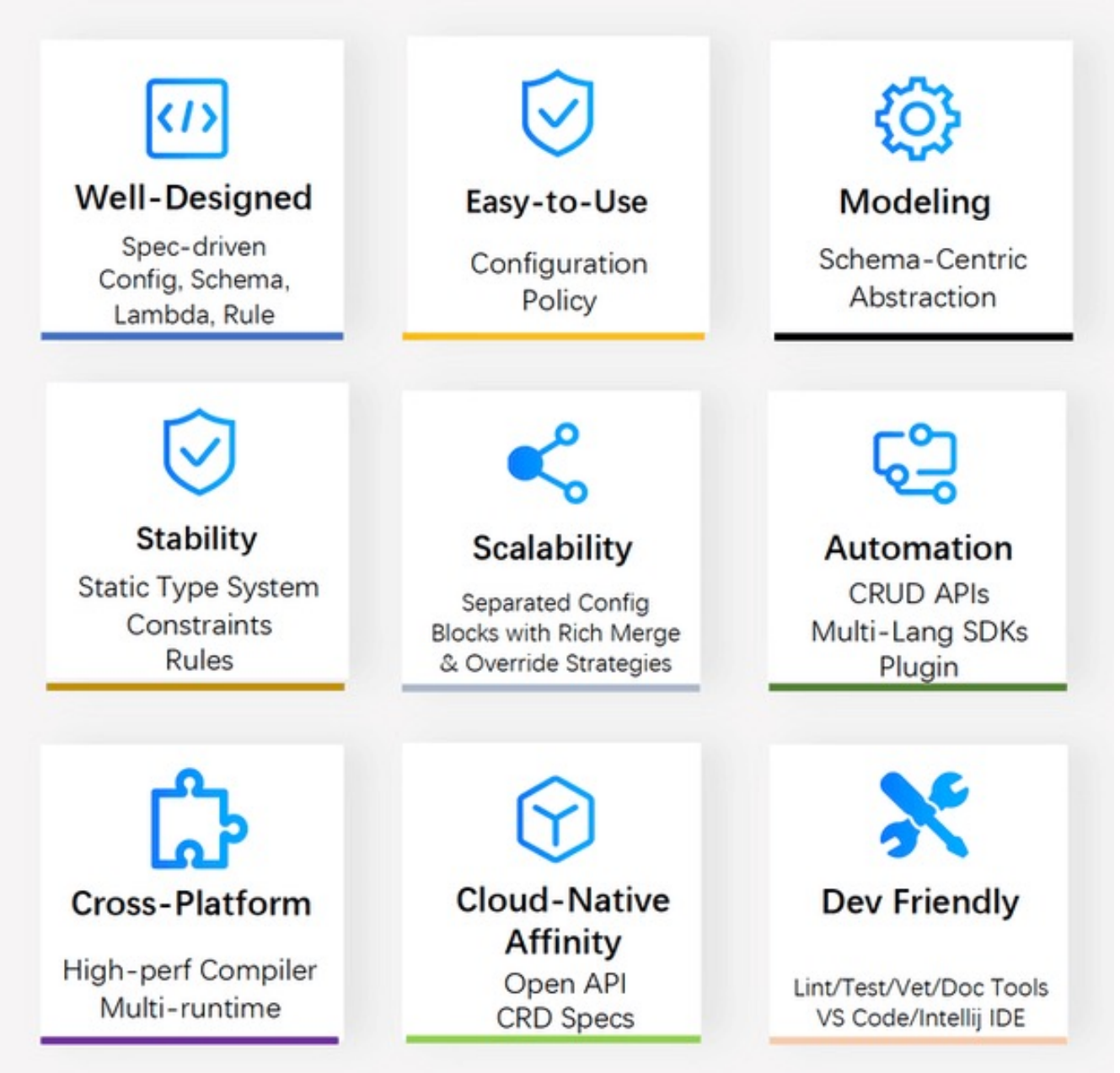
Tech



Spec Driven

KCL Spec | KCL Multi-Language Spec | KCL OpenAPI Spec

Features


Well-Designed
Spec-driven
Config, Schema,
Lambda, Rule
Easy-to-Use
Configuration
Policy
Modeling
Schema-Centric
Abstraction
Stability
Static Type System
Constraints
Rules
Scalability
Separated Config
Blocks with Rich Merge
& Override Strategies
Automation
CRUD APIs
Multi-Lang SDKs
Plugin
Cross-Platform
High-perf Compiler
Multi-runtime
Cloud-Native
Affinity
Open API
CRD Specs
Dev Friendly
Lint/Test/Vet/Doc Tools
VS Code/Intellij IDE

Pattern

The configuration of attributes in KCL usually meets the simple pattern:

$$k = (T)v$$

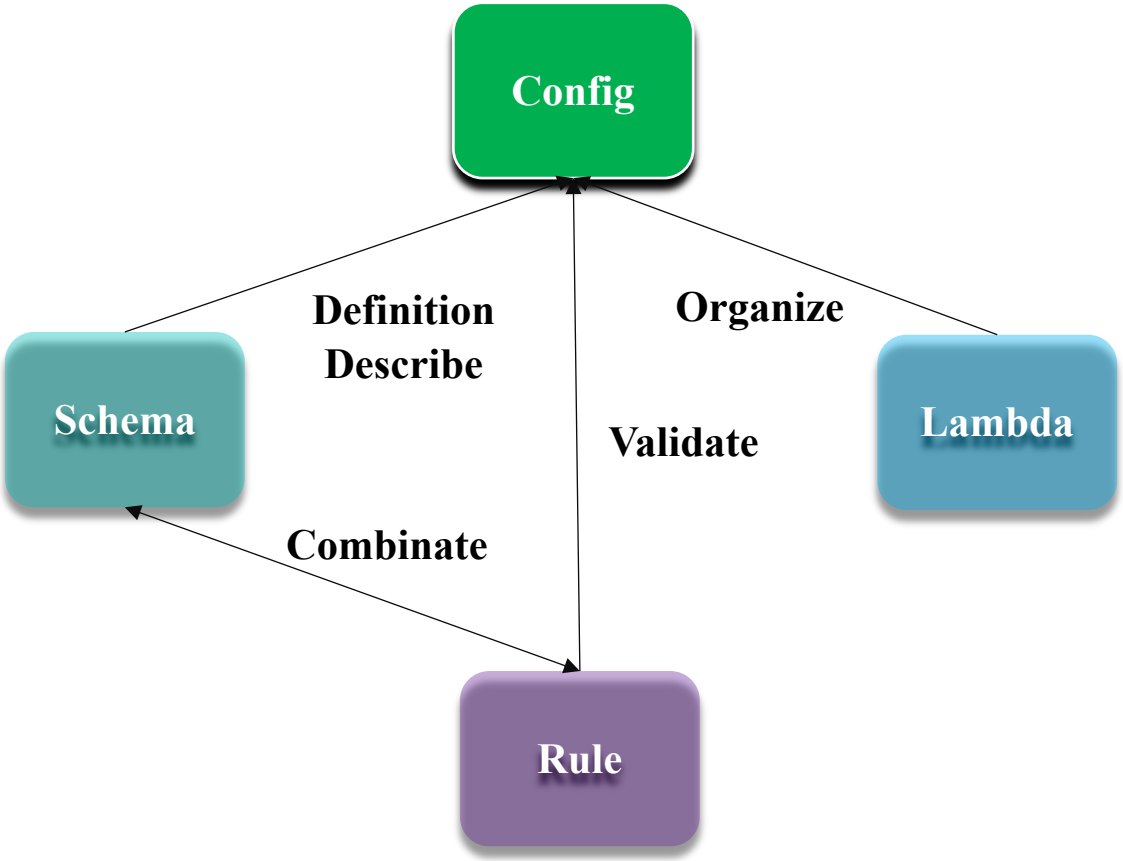
where k is the attribute name, v is the attributes value, and T is the type annotation. Since KCL has the ability of the type inference, T is usually omitted.

This is an example of generating kubernetes manifests.

```
apiVersion = "apps/v1"
kind = "Deployment"
metadata = {
    name = "nginx"
    labels.app = name
}
spec = {
    replicas = 3
    selector.matchLabels = metadata.labels
    template.metadata.labels = metadata.labels
    template.spec.containers = [
        {
            name = metadata.name
            image = "${metadata.name}:1.14.2"
            ports = [{ containerPort = 80 }]
        }
    ]
}
```

Concepts

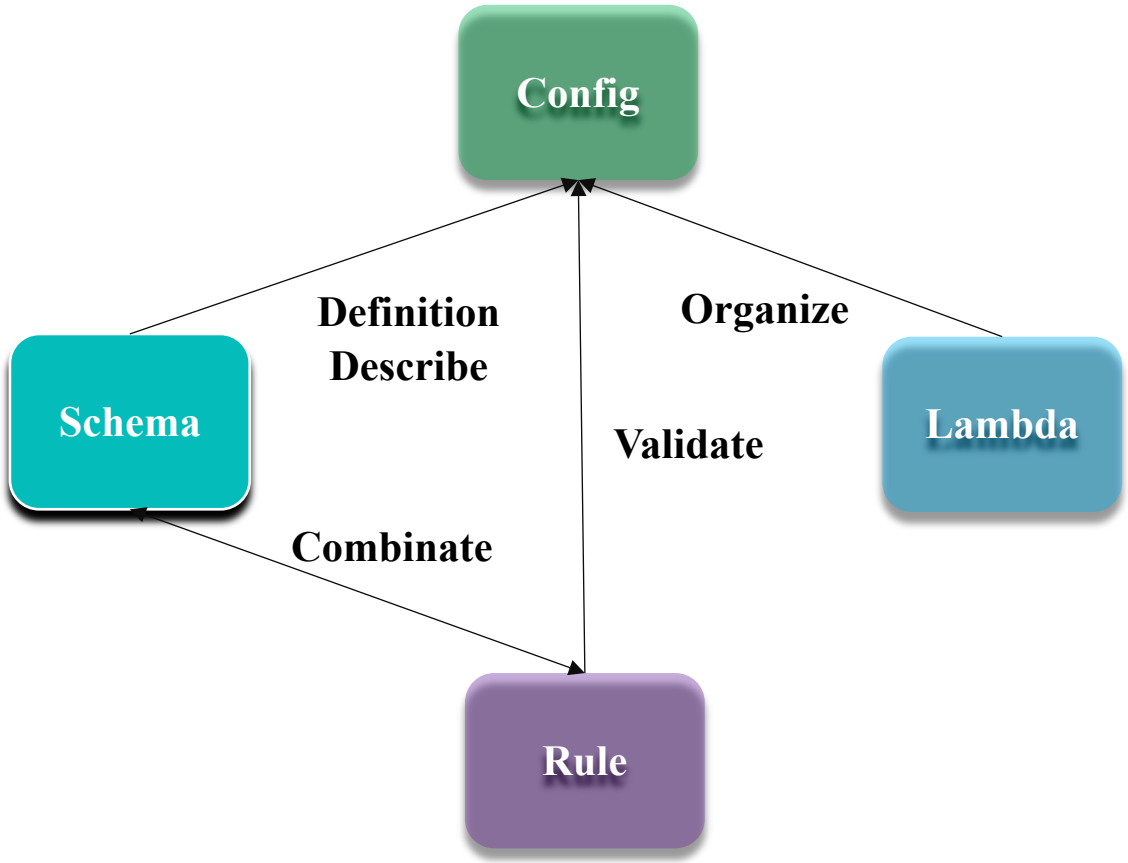
KCL = Config + Schema + Rule + Lambda




```
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 }]
            }]
        }
    }
}
```

Concepts

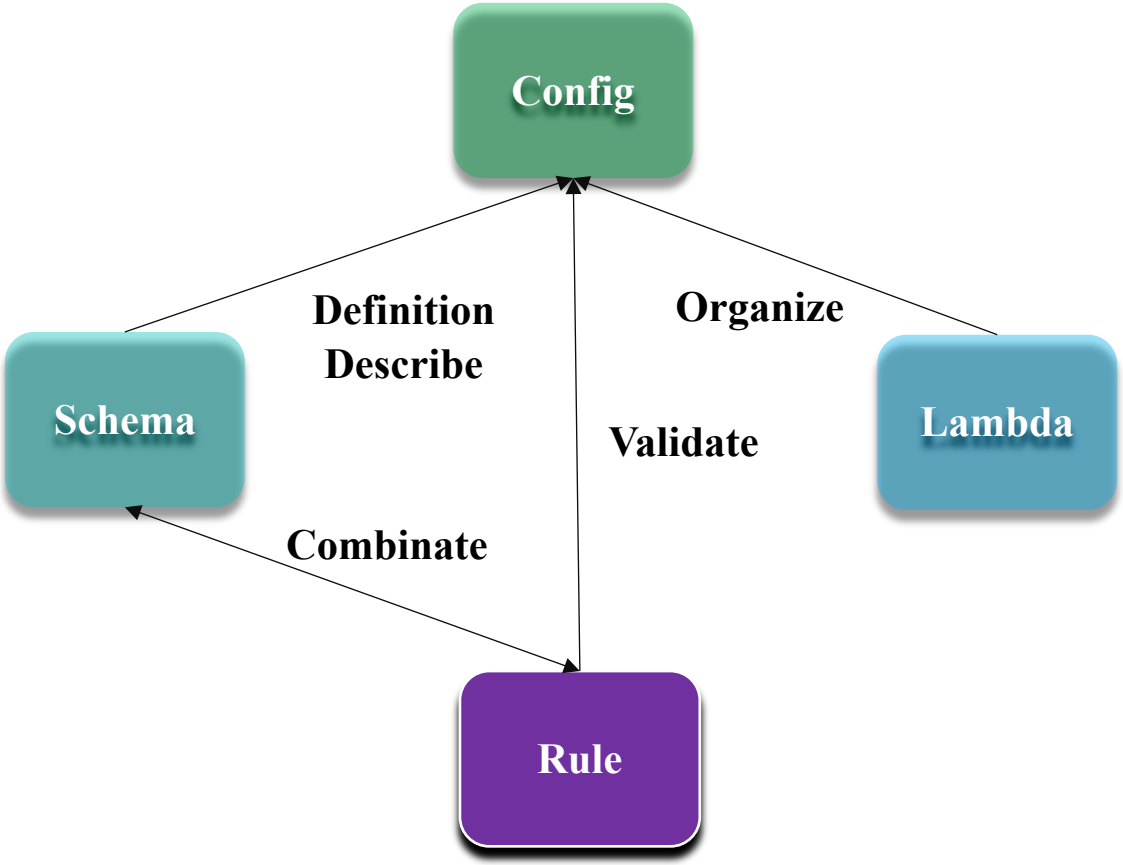
KCL = Config + Schema + Rule + Lambda




```
import units

type UnitType = units.NumberMultiplier
# Define a schema named Resource with
# three attributes and constraits.
schema Resource:
    cpu: int | UnitType = 1
    memory: UnitType = 1024Mi
    disk: UnitType = 10Gi

    check:
        0 < cpu <= 64
        0 < memory <= 64Gi
        0 < disk <= 1Ti
```

Concepts

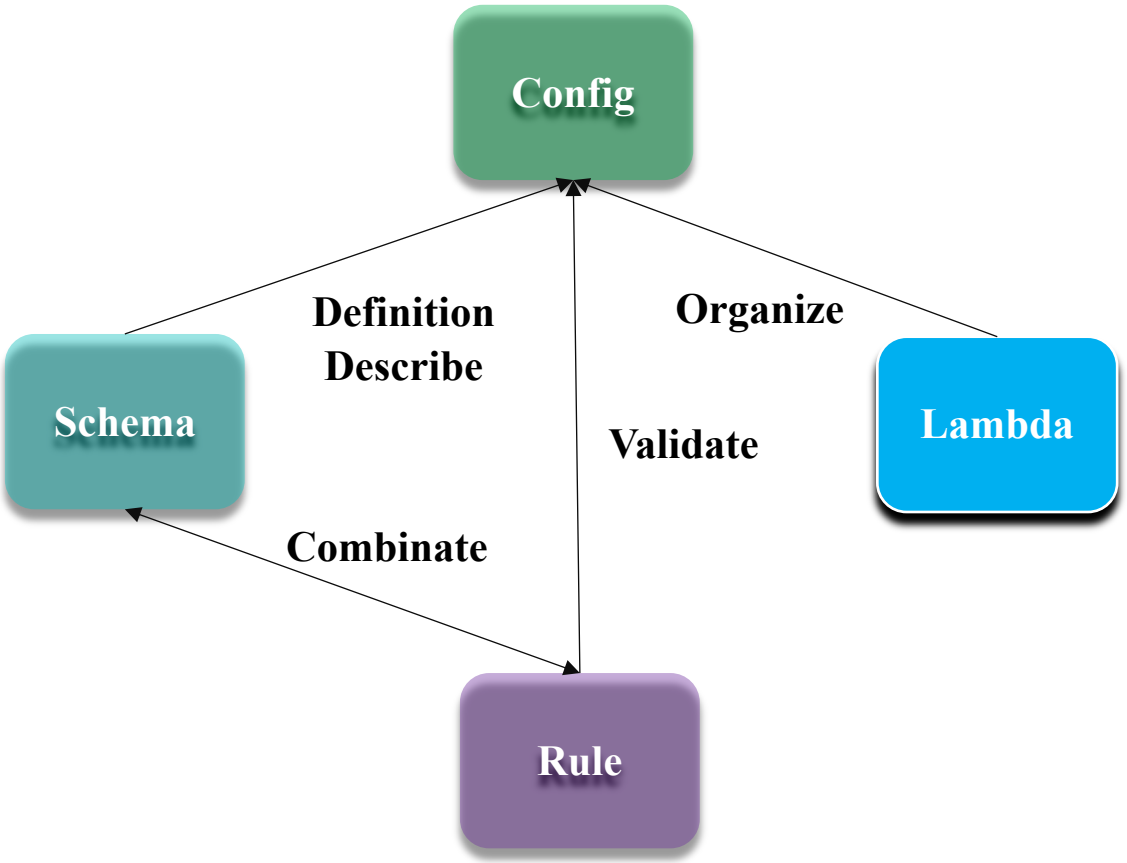
KCL = Config + Schema + Rule + Lambda




```
data: Data = option("data")
input: Input = option("input")
# Define a RBAC rule
rule Allow:
    any grant in UserIsGranted() {
        input.action == grant.action and input.type == grant.type
    }
    any user in data.user_roles[input.user] { user == "admin" }

rule UserIsGranted:
    [
        grant
        for role in data.user_roles[input.user]
        for grant in data.role_grants[role]
    ]

allow = Allow() or False
```

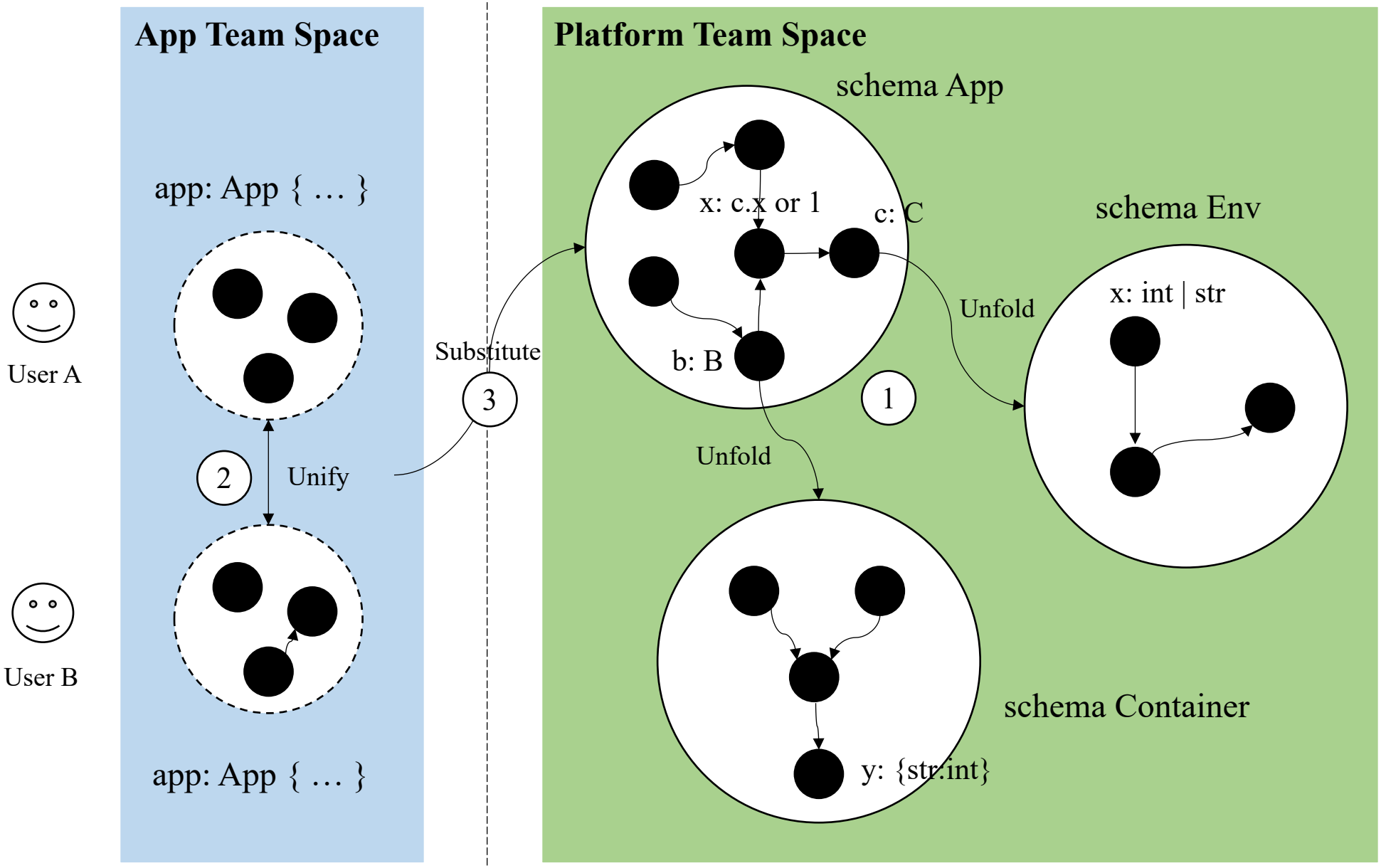
Concepts

KCL = Config + Schema + Rule + Lambda




```
# Transform input resrource and change annotations.
transformer = lambda res {
    res | {
        metadata.annotations: {
            "managed-by" = "kcl"
        }
    } if res.kind == "Deployment" else res
}

output = [transformer(res) for res in option("input")]
```

Graph Model



- ✓ **Config Generation**
  - ✓ Automatic merge
  - ✓ Multiple Merge Strategy
- ✓ **Config Mutation**
  - ✓ Data Integration
  - ✓ CRUD API
- ✓ **Config Validation**
  - ✓ Static Type
  - ✓ Validation
  - ✓ Immutability
- ✓ **Config Abstraction**
  - ✓ Role based collaborative

Dependency

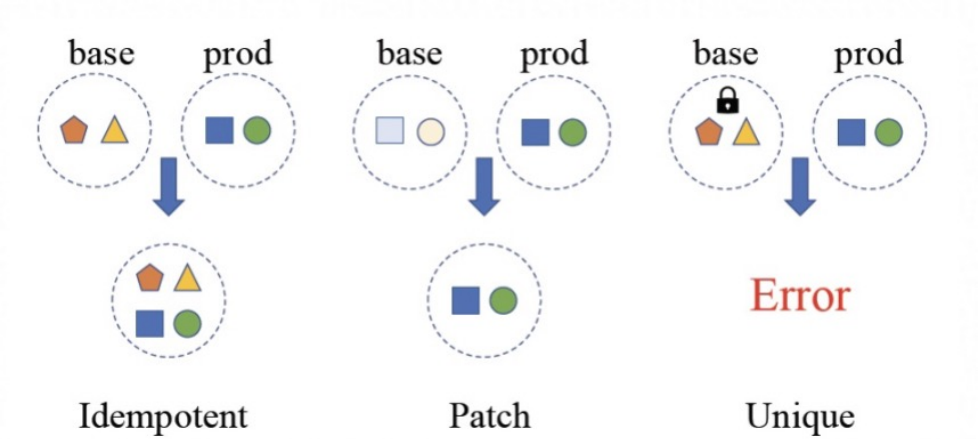
Attribute

Data

Schema Definition

KCL SETTA 2022 Paper：https://link.springer.com/chapter/10.1007/978-3-031-21213-0_6

Configuration



- base.k

```
# Application Configuration
appConfiguration: frontend.Server {
    # Main Container Configuration
    mainContainer.ports = [
        {containerPort = 80}
    ]
    image = "nginx:1.7.8"
}
```

- prod.k

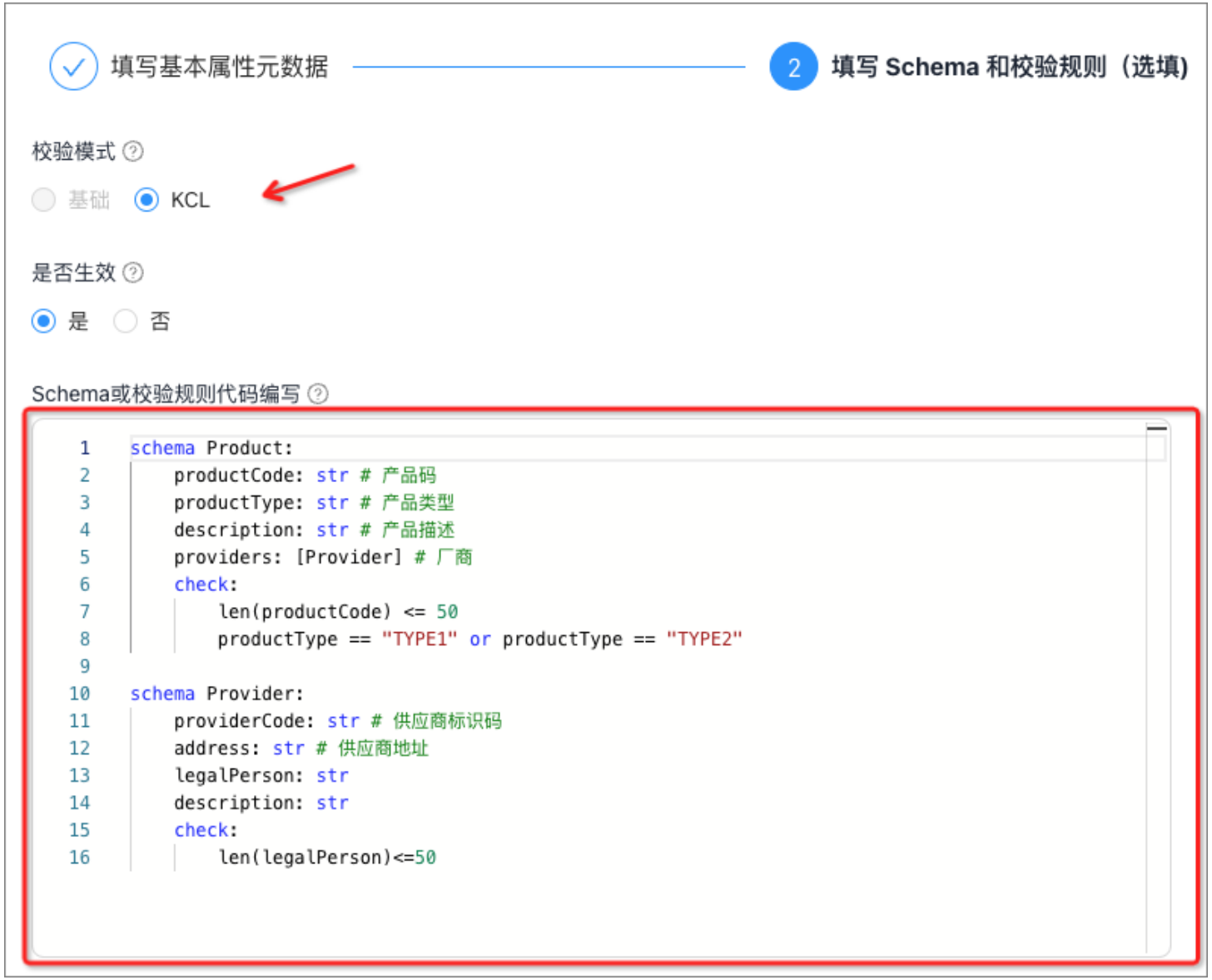
```
appConfiguration: frontend.Server {
    schedulingStrategy.resource = res.Resource {
        cpu = 100m
        memory = 100Mi
        disk = 1Gi
    }
}
```

Equivalent code

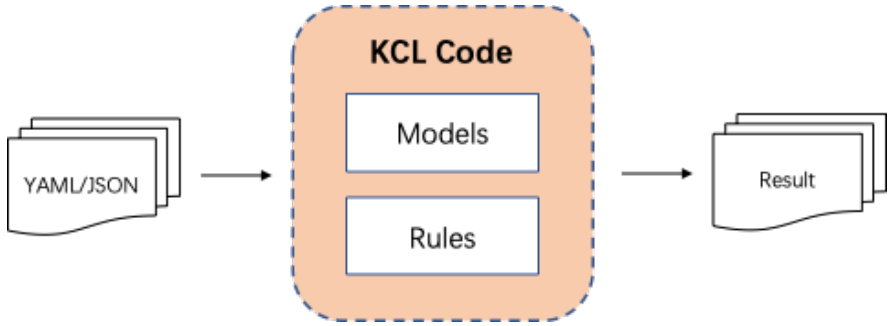
```
# Application Configuration
appConfiguration: frontend.Server {
    # Main Container Configuration
    mainContainer.ports = [
        {containerPort = 80}
    ]
    image = "nginx:1.7.8"
    schedulingStrategy.resource = res.Resource {
        cpu = 100m
        memory = 100Mi
        disk = 1Gi
    }
}
```

Multi Environment, Multi Tenant

Validation




```
schema Product:
    productCode: str # 产品码
    productType: str # 产品类型
    description: str # 产品描述
    providers: [Provider] # 厂商
    check:
        len(productCode) <= 50
        productType == "TYPE1" or productType == "TYPE2"

schema Provider:
    providerCode: str # 供应商标识码
    address: str # 供应商地址
    legalPerson: str
    description: str
    check:
        len(legalPerson)<=50
```



- ✓ JSON/YAML Support
- ✓ Open API Support
- ✓ Custom Error Message
- ✓ Model/Rule Reuse
- ✓ Schema Migration
- ✓ Definition Export

Abstraction

```
import .app

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

Application Model

Docker Compose
Transformer

```
services:
  app:
    image: ngnix
    ports:
      - published: 80
        target: 80
        protocol: TCP
```

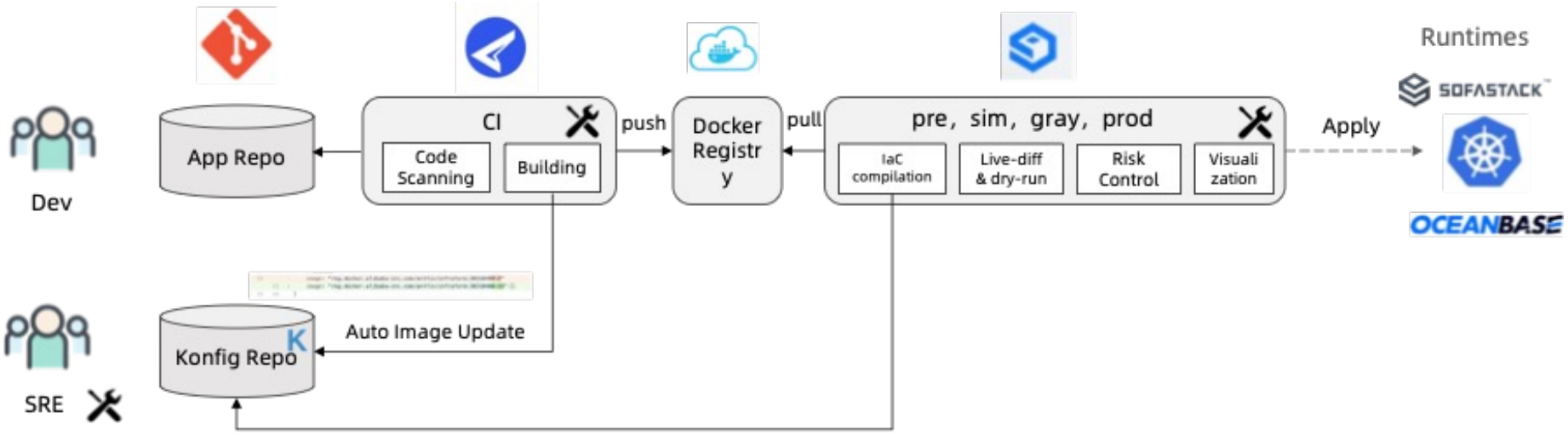
Kubernetes
Transformer

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app
  labels:
    app: app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app
```

X
Transformer

Your Infra or Platform Spec

Automation



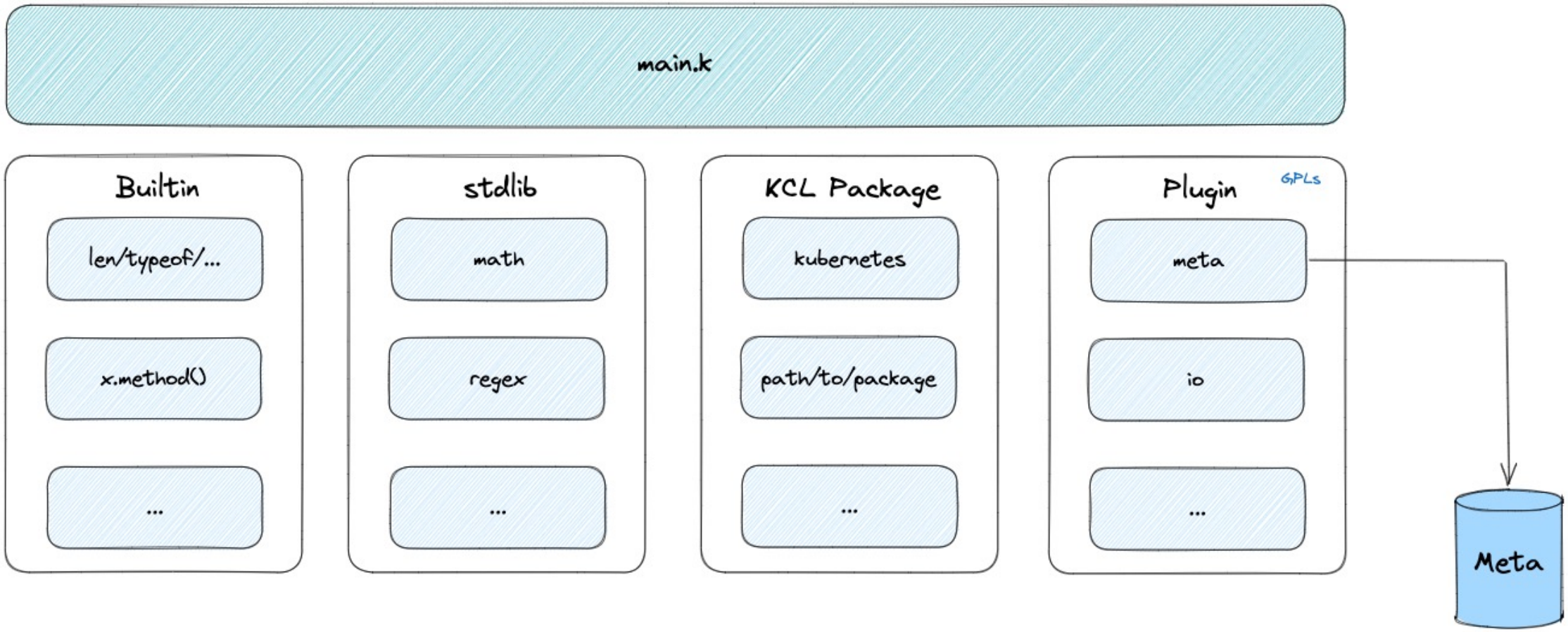

```
import base.pkg.kusion_models.kube.frontend
import base.pkg.kusion_models.kube.frontend.service
import base.pkg.kusion_models.kube.frontend.container
import base.pkg.kusion_models.kube.templates.resource as res_tpl

# Application Configuration
appConfiguration: frontend.Server {
    # Main Container Configuration
    mainContainer = container.Main {
        ports = [
            {containerPort = 80}
        ]
    }
    image = "nginx:1.7.8"
}

```

```
import base.pkg.kusion_models.kube.frontend
import base.pkg.kusion_models.kube.frontend.service
import base.pkg.kusion_models.kube.frontend.container
import base.pkg.kusion_models.kube.templates.resource as res_tpl

# Application Configuration
appConfiguration: frontend.Server {
    # Main Container Configuration
    mainContainer = container.Main {
        ports = [
            {containerPort = 80}
        ]
    }
    image = "nginx:1.7.9"
}

```

CI/CD Integration, GitOps Friendly

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

Modules

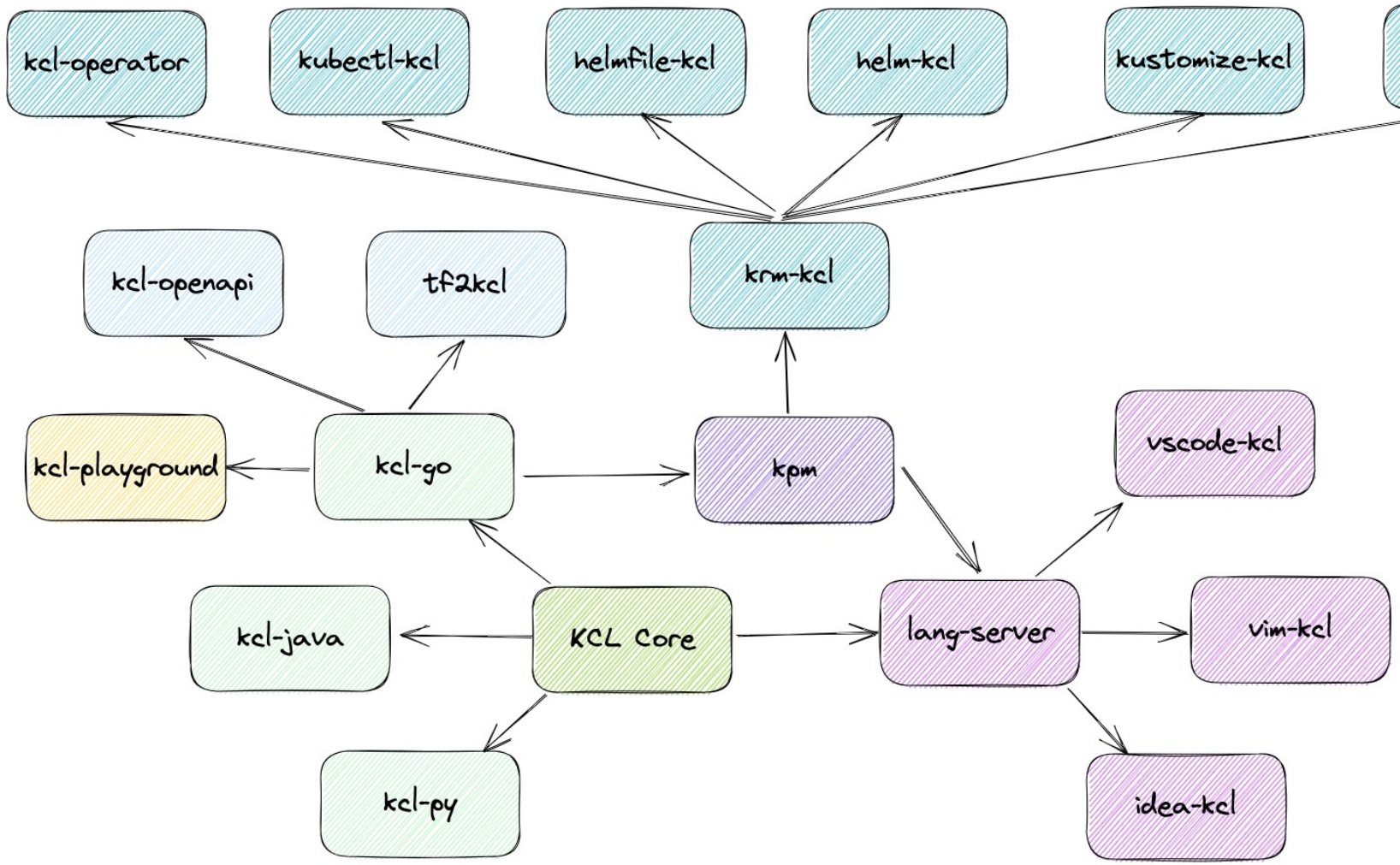


100+ built-in functions, modules with additional KPM package manager tools, OCI Registry and plugins

Scenarios

03

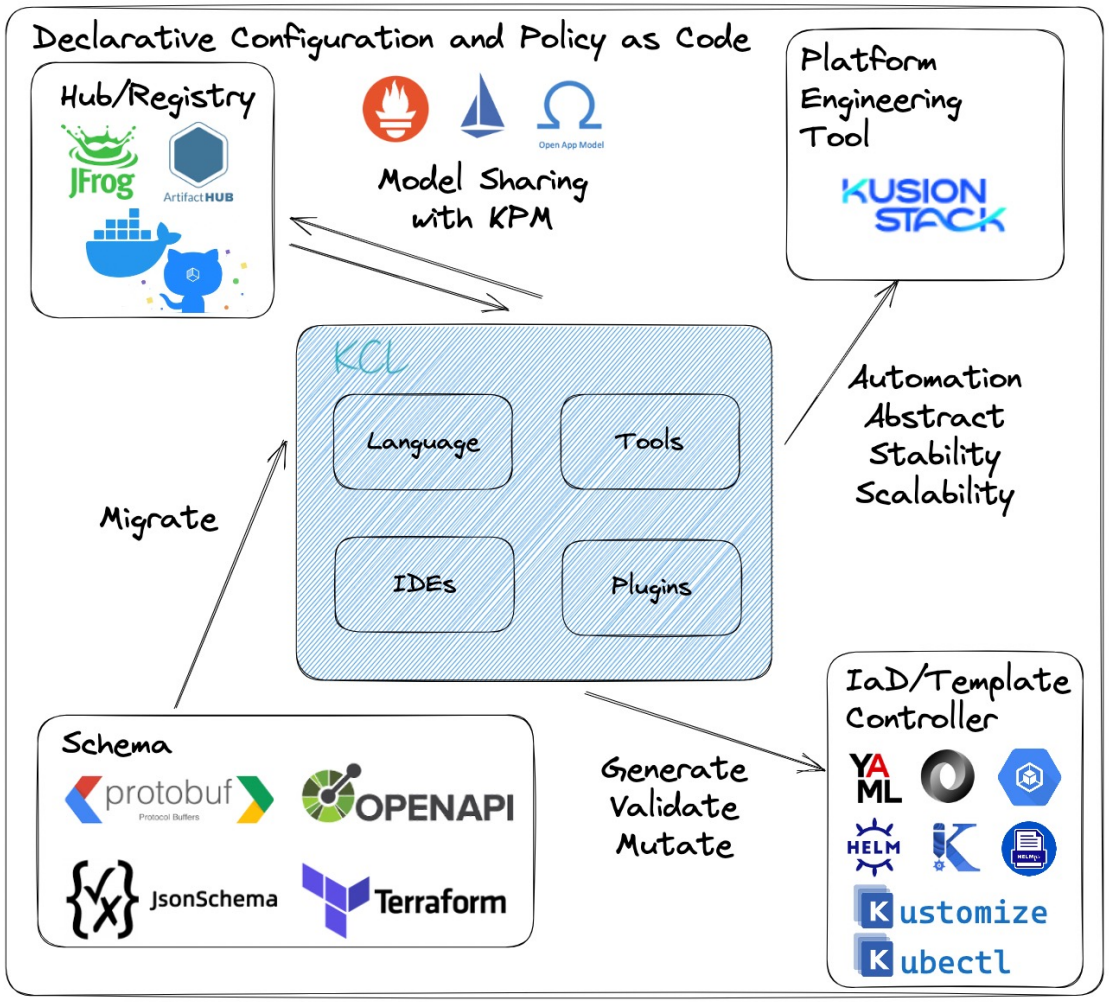
Components



How to (For App/Platform dev & SRE)

- Lang
- SDKs
- Tools
- IDE
- Playground
- UI
- Cloud-native Tool Integrations

Integrations



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

Server Admission Request ↔ Mutating/Validating Webhook

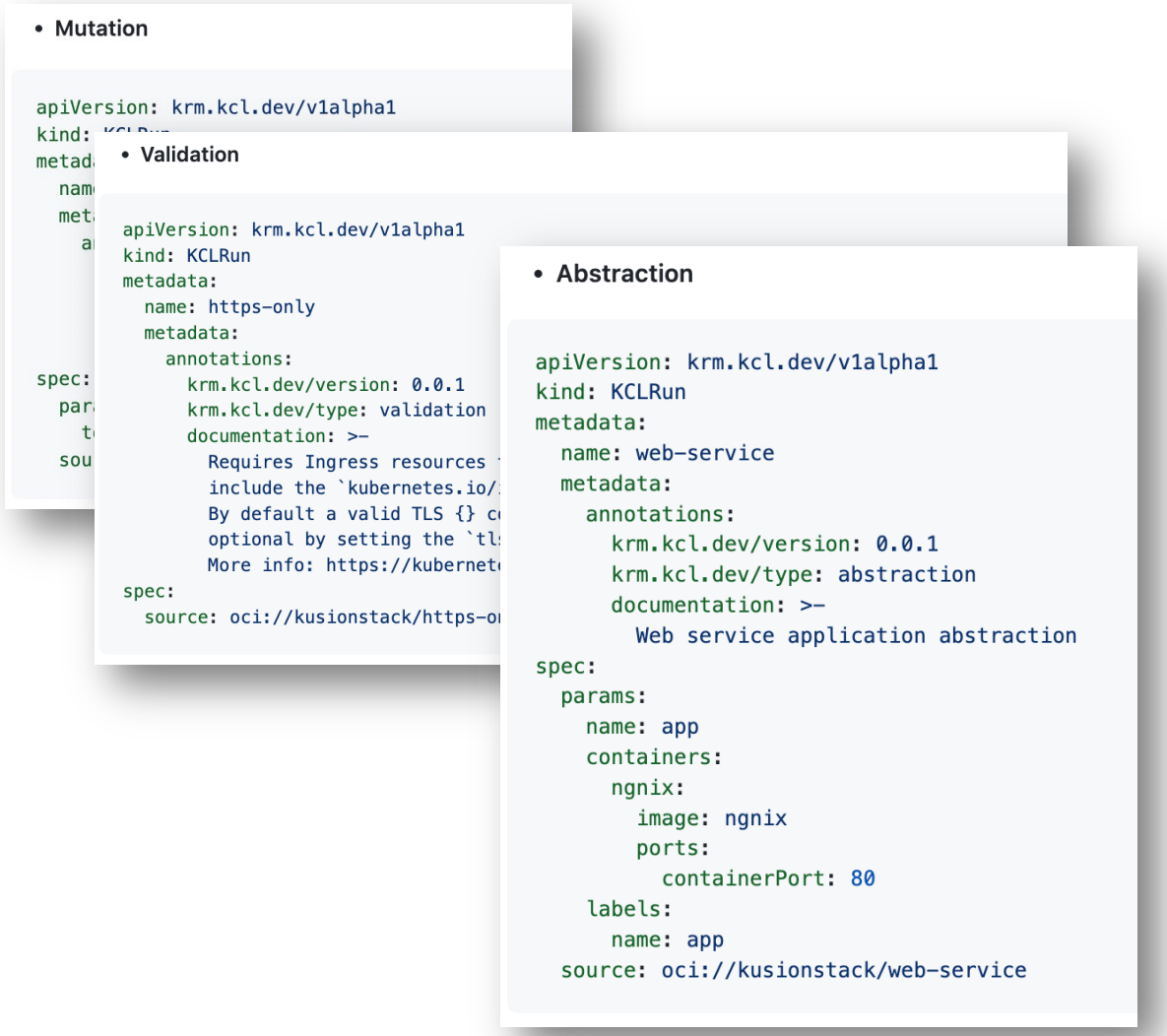
Binding Rust, Go, Java, Python

Enhancement/Glue

- KRM Support: Unified KRM KCL Spec
- Migration: Data, Schema
- Sharing: Modules, Functions
- GitOps
- Lang Binding

Integrations

KRM-KCL Specification



Guides for Developing KCL

Here's what you can do in the KCL script:

- Read resources from `option("resource_list")`. The `option("resource_list")` complies with the KRM Functions Specification. You can read the input resources from `option("resource_list")["items"]` and the `functionConfig` from `option("resource_list")["functionConfig"]`.
- Return a KPM list for output resources.
- Return an error using `assert {condition}, {error_message}`.
- Read the environment variables. e.g. `option("PATH")` (Not yet implemented).
- Read the OpenAPI schema. e.g. `option("open_api")["definitions"]["io.k8s.api.apps.v1.Deployment"]` (Not yet implemented).

Evaluation

04

Related Works

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
- …

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
- …

Pros.

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

Cons.

- Insufficient type constraints
- Insufficient restraint ability
- Runtime error

Tech.

- GCL
- HCL
- JSONNET
- …

Product

- Terraform
- …

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
- …

Product

- KubeVela
- …

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**
- …

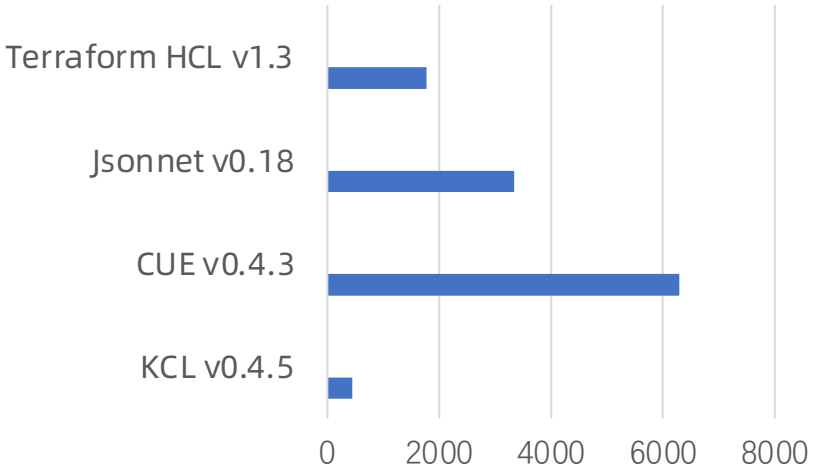
Product

- KusionStack
- …

Performance

Simple Computing

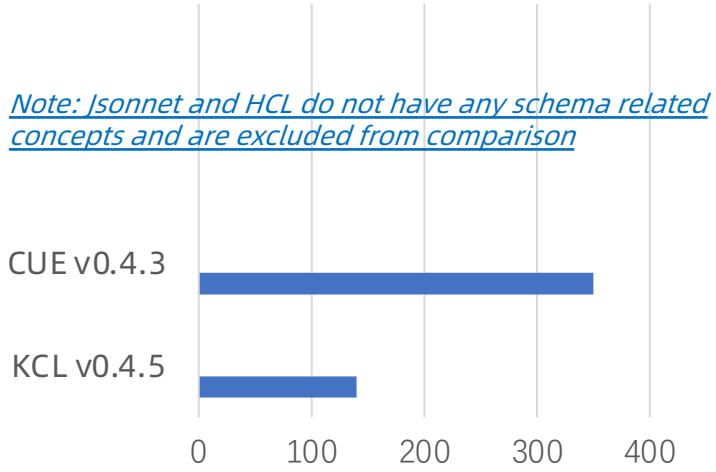
```
a = lambda x: int, y: int -> int {
    max([x, y])
}
temp = {"a${i}": a(1, 2) for i in range(10000)}
```



Kubernetes Schema 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)

Adopters

ANT GROUP

kcl

15	32	2000+	100+	600+
BG	BU	Projects	Clusters	Committers

1K/day	40K+	120K+	~1M	10K+/day
Pipelines	PRs	Commits	KCL Codes	KCL Compilations

有赞

Use the KCL OpenKruise -> Application abstraction to support the middleware containerization configuration management in over 100+ clusters

HUAWEI

Use KCL to solve the problems such as the difficulty in abstracting Terraform HCL and the lack of scalability can be solved to meet its SaaS configuration UI definition requirements. 2000+ KCL model code, 3+ KCL Plugins

Tech Roadmap

- **v0.4.6 Release**
- Ease of Use and Error Improvement
- Helm KCL Schema Plugin Design
- KPT KCL SDK Deign
- KCL KPM Pre-release (Git Repo, Lock, Dependency management)
- ConfVet Design
- Go SDK: Better Installation, Go Struct/KCL Schema Conv API，List Instance API, etc.
- KCL New IDE Plugin Pre-release
- KCL Playground Code Share

2023.4

- **v0.5.0 Release**
- Improvement of Grammar and Standard Libraries
- KCL Abstract/Constraint Specification Standards and Best Practices Manual
- Helm KCL Schema Plugin Release
- KPT KCL SDK Release
- KCL KPM Release
- KCL New Test Tool Design
- ConfVet Release
- More SDK Support e.g., Java, Node.js
- KCL New IDE Plugin Stable-release (Syntax error recovery, semantic highlighting, semantic completion, etc.)
- More KCL Playground Code Samples

2023.6

- **v0.6.0 Release**
- KCL New UI Design
- Kustomize KRM KCL Plugin Design
- KCL KPM OCI Registry
- KCL New Test Tool Release
- CompilerBase Core Components
- More KCL IDE Support e.g., Vim

2023.9

- **v0.7.0 Release**
- KCL New UI Release
- Kustomize KRM KCL Plugin Release
- KCL KPM S3/OSS
- CompilerBase Grammar and Sematic

2023.12

- Language
- Tool
- SDK
- Scene & Integration
- Technical Subproject
- IDE Plugin

https://github.com/kcl-lang/kcl/issues/29

Resources

- **Website**
  - https://kcl-lang.io/
- **GitHub Organization**
  - https://github.com/kcl-lang/kcl
- **Community**
  - https://github.com/kcl-lang/community
- **Twitter**
  - @kcl language

THANKS