

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

Pengfei Xu ([xpf6677@gmail.com](mailto:xpf6677@gmail.com))

July 2023

# Agenda

**01 Background**

**02 Design**

**03 Scenarios**

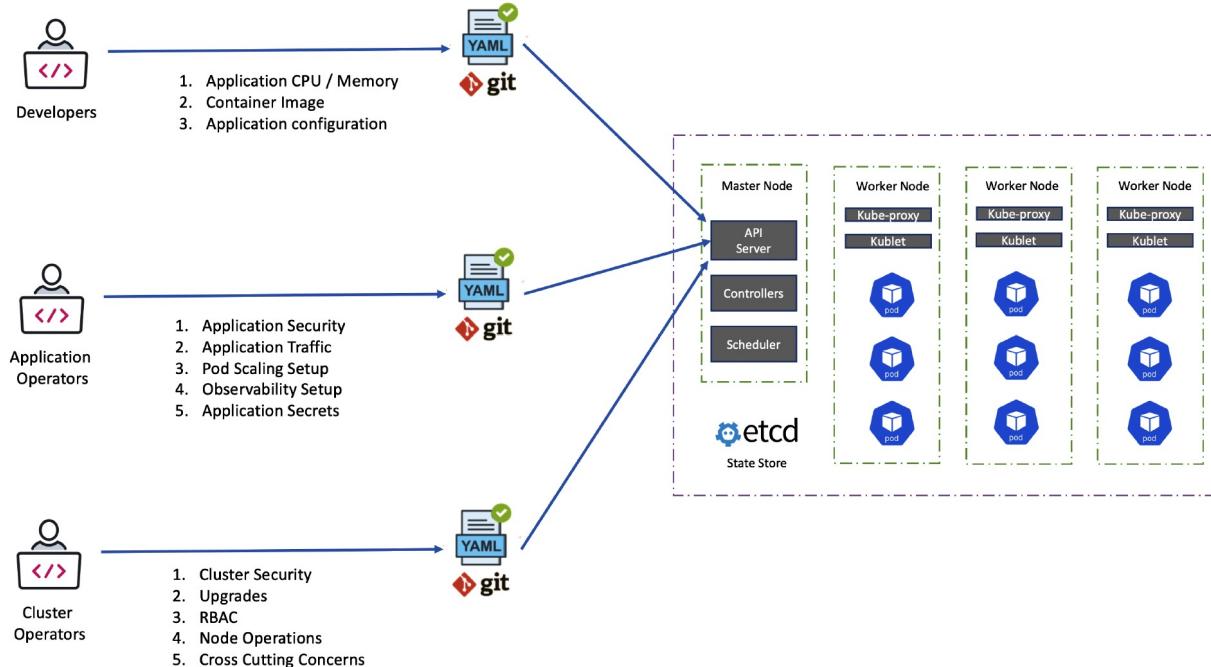
**04 Evaluation**

# 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

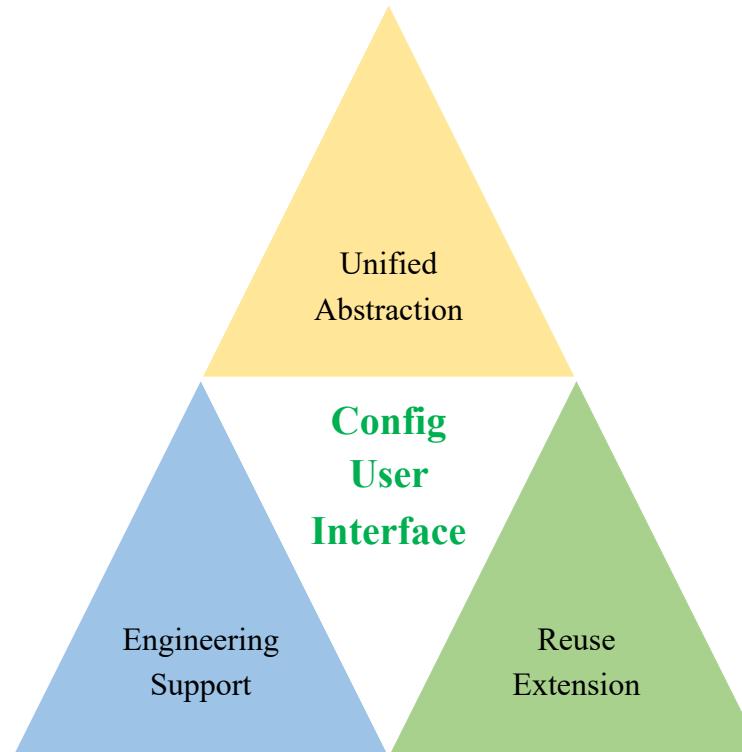
## Low Efficiency/Reliability

- 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

**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**

# 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

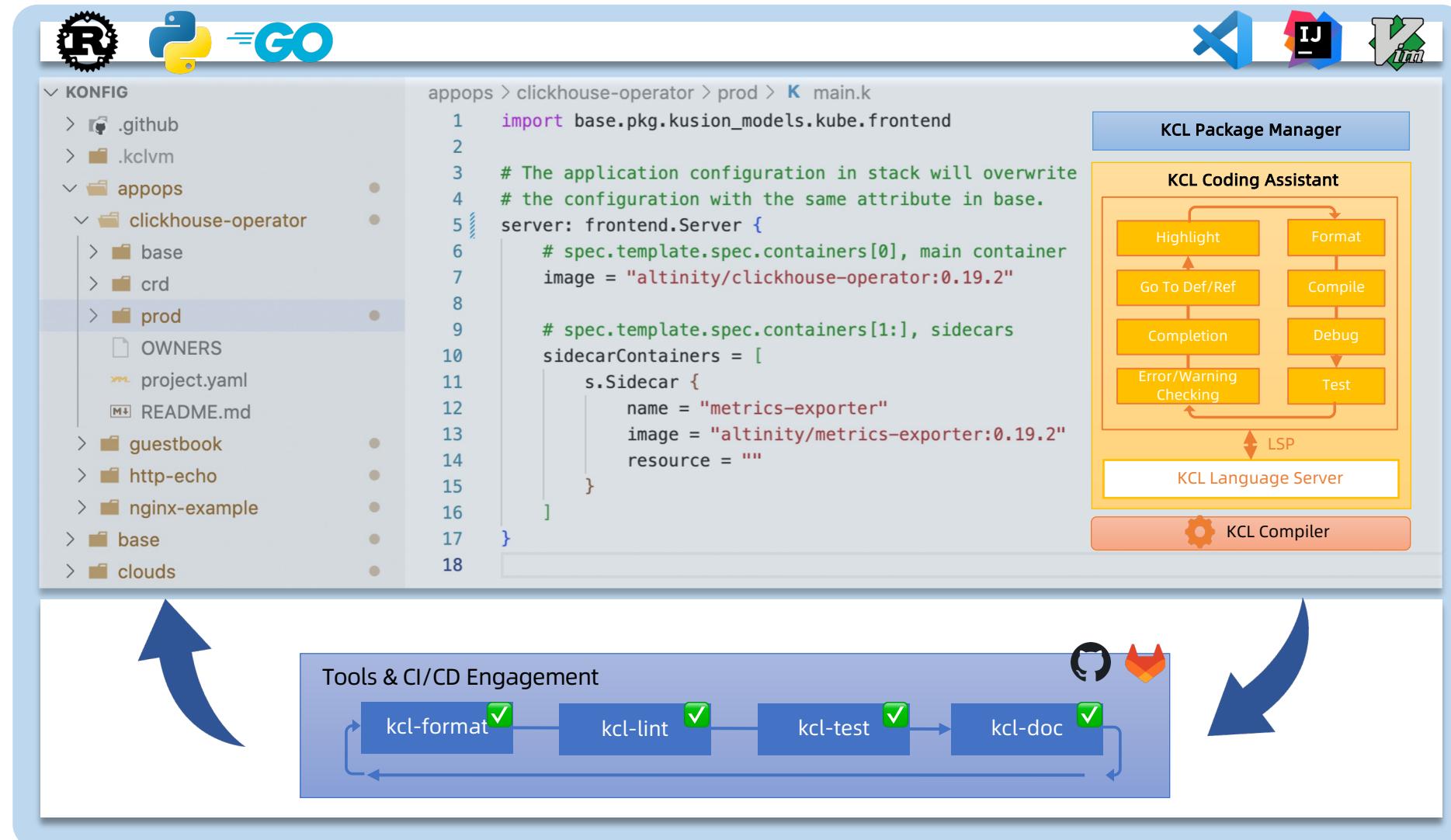
---

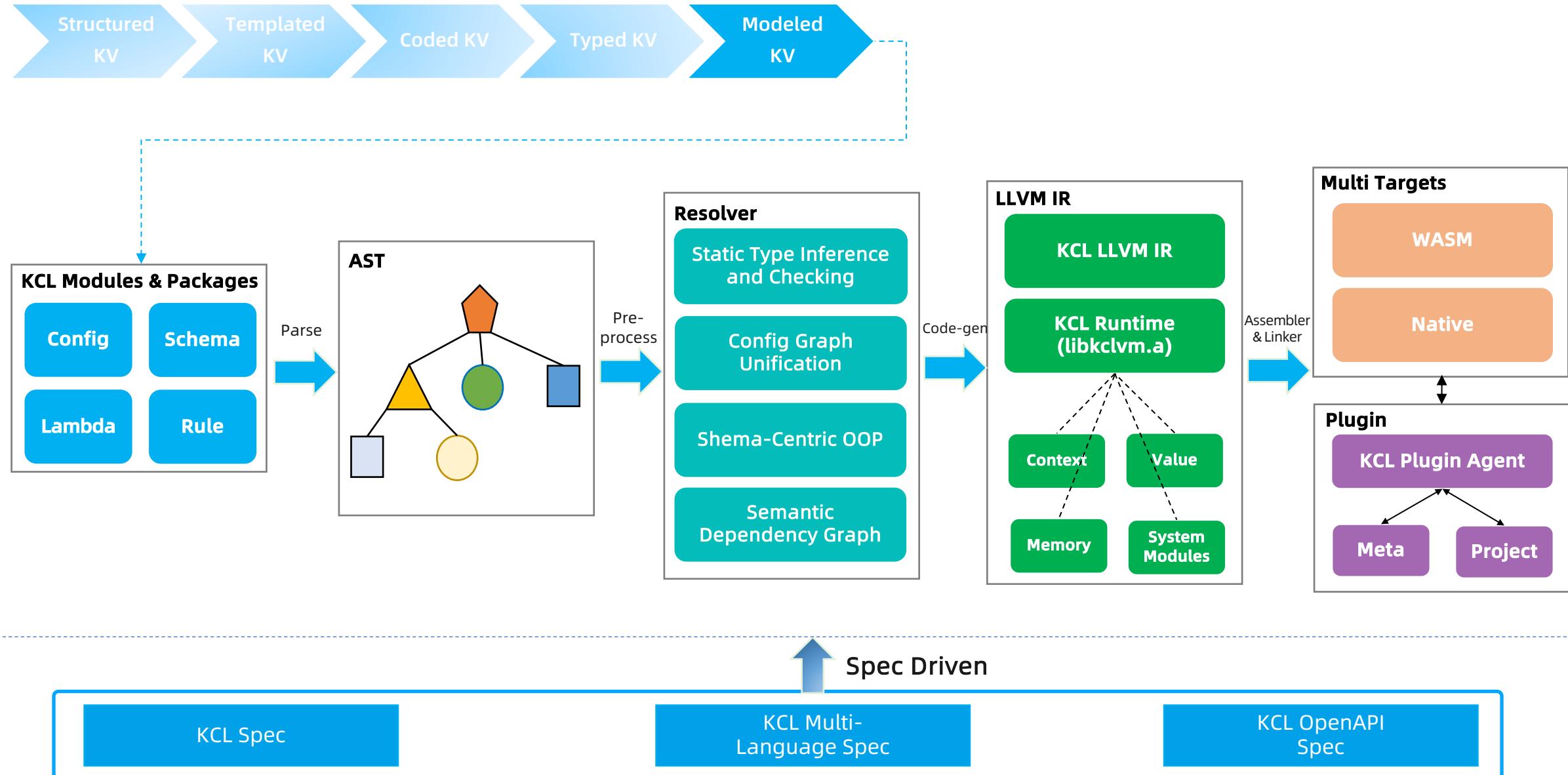
02

# Overview

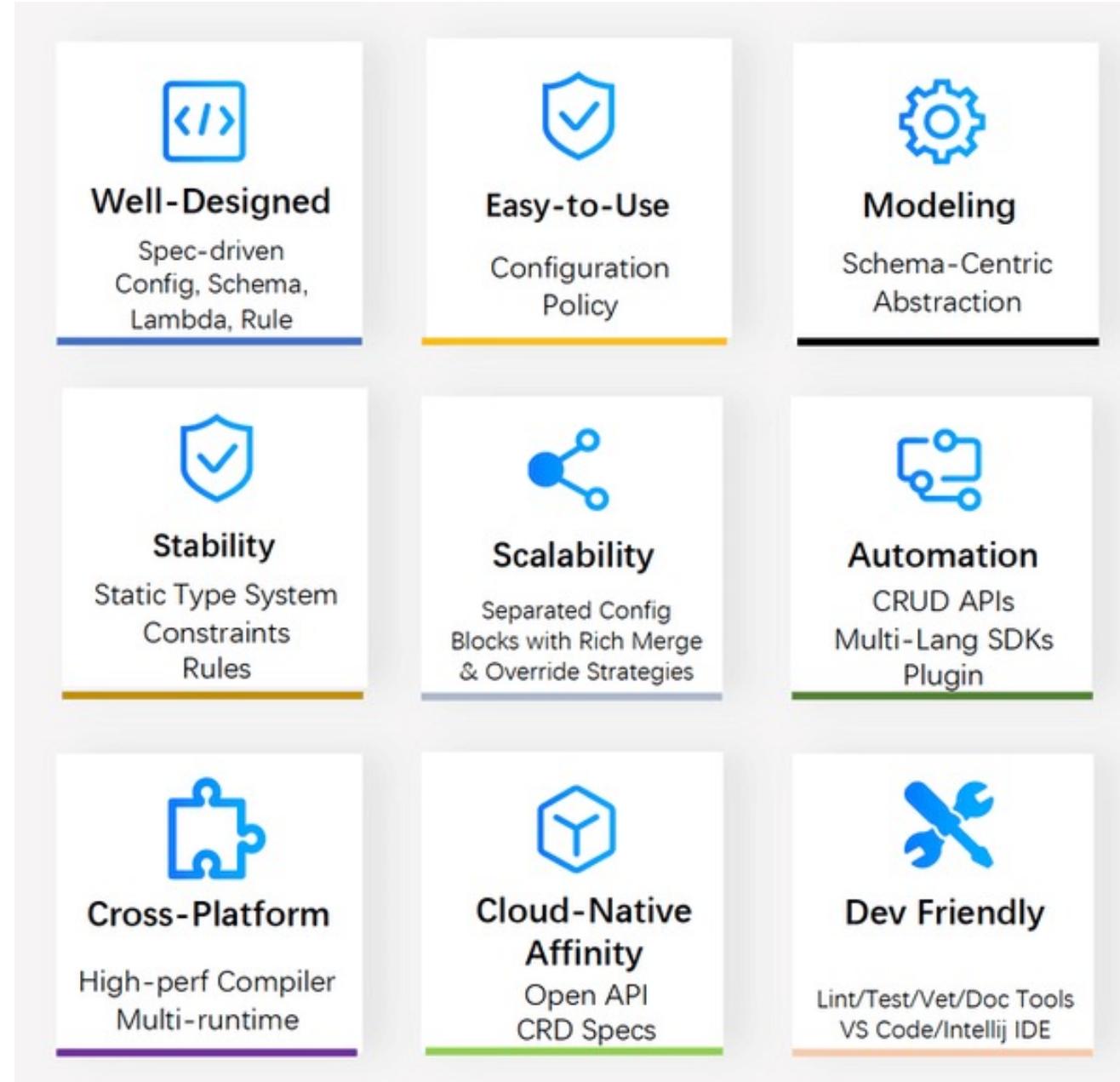


Language + Tools + IDEs + SDKs + Plugins





# Features



# 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.

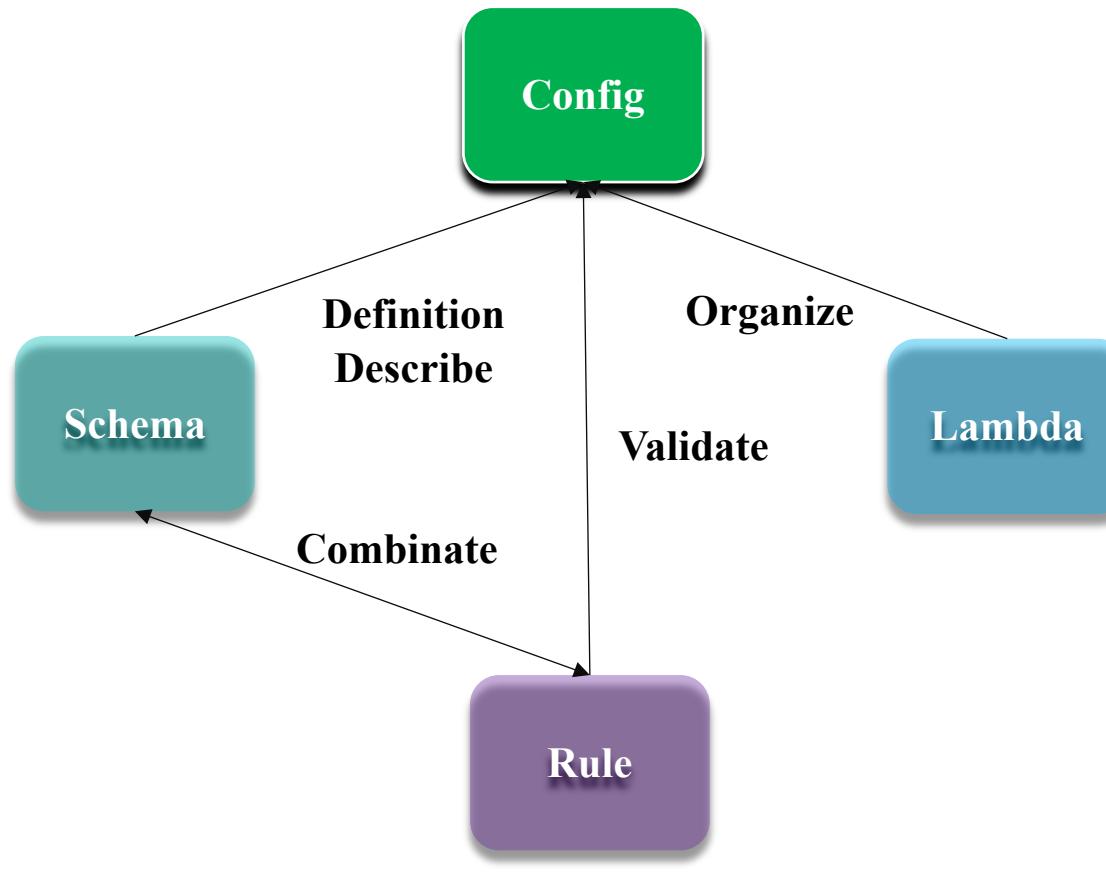
This is an example of generating kubernetes manifests.

```
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**

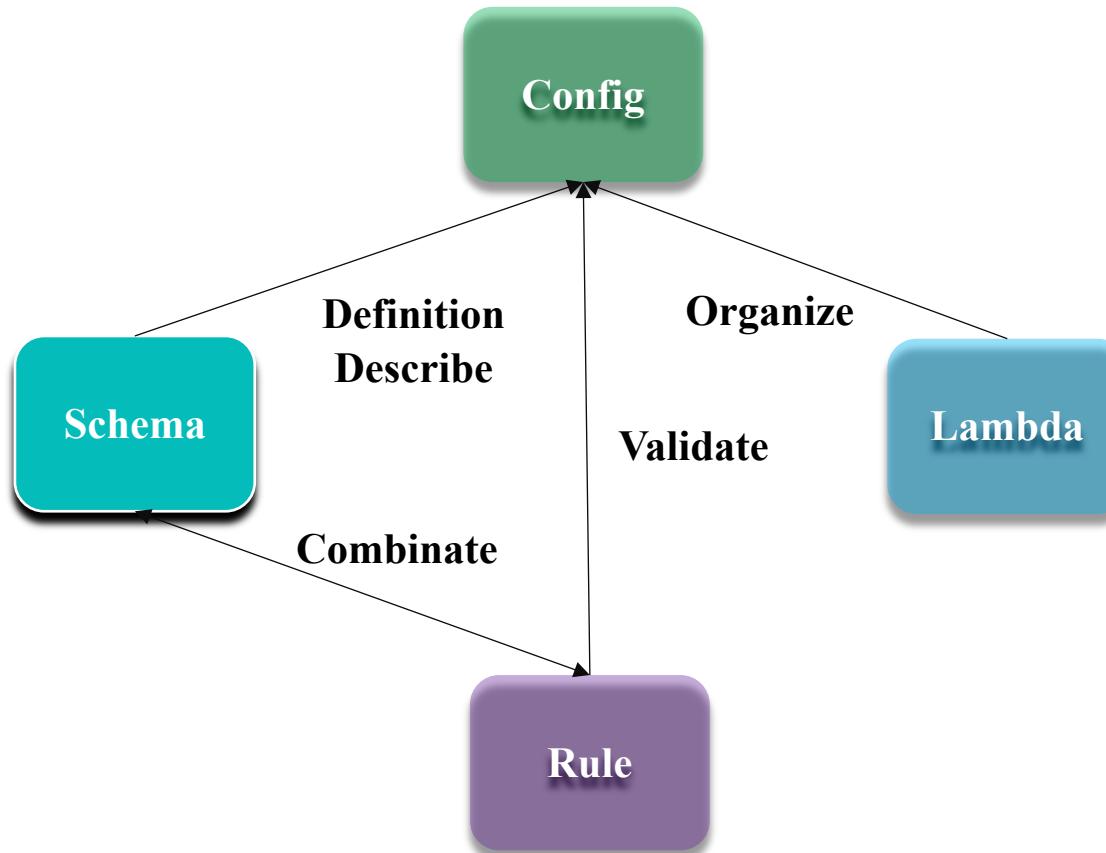


```
import k8s.core.v1
# Create a Kubernetes Deployment resource.
v1.Deployment {
    metadata.name = "nginx"
    metadata.labels.app = metadata.name
    spec = {
        replicas = 3
        selector.matchLabels.app = metadata.name
        template = {
            metadata.labels.app = metadata.name
            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 constraints.  
schema Resource:  
    cpu: int | UnitType = 1  
    memory: UnitType = 1024Mi  
    disk: UnitType = 10Gi
```

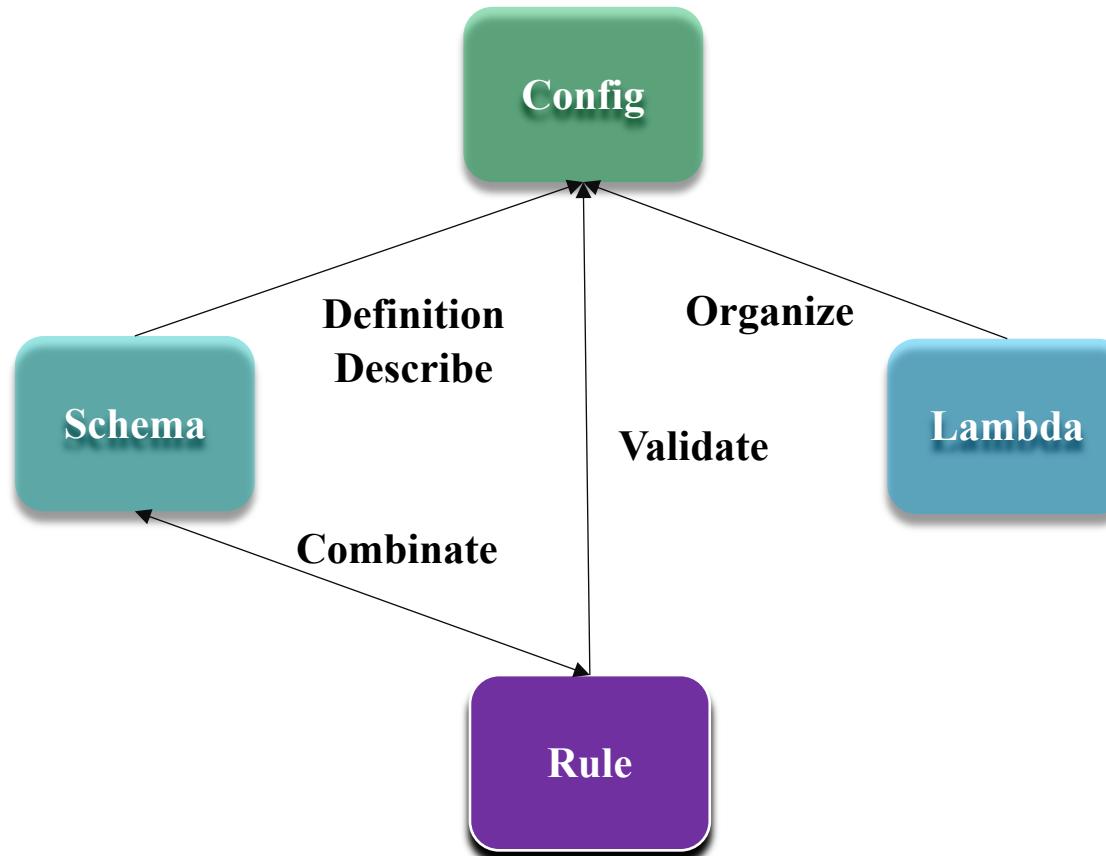
`check:`

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

# Concepts



**KCL = Config + Schema + Rule + Lambda**



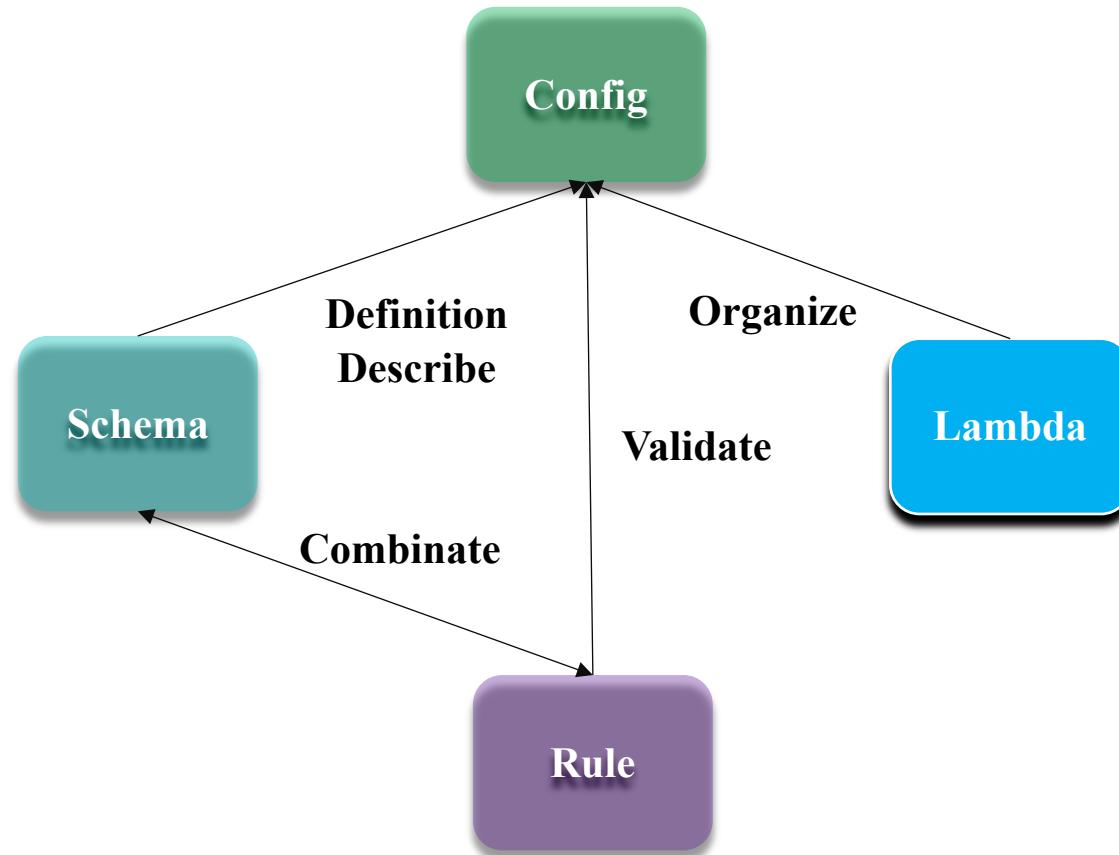
```
data: Data = option("data")
input: Input = option("input")
# Define a RBAC rule
rule Allow:
    any grant in UserIsGranted() {
        input.action == grant.action and input.type == grant.type
    }
    any user in data.user_roles[input.user] { user == "admin" }

rule UserIsGranted:
    [
        grant
        for role in data.user_roles[input.user]
        for grant in data.role_grants[role]
    ]
allow = Allow() or False
```

# Concepts

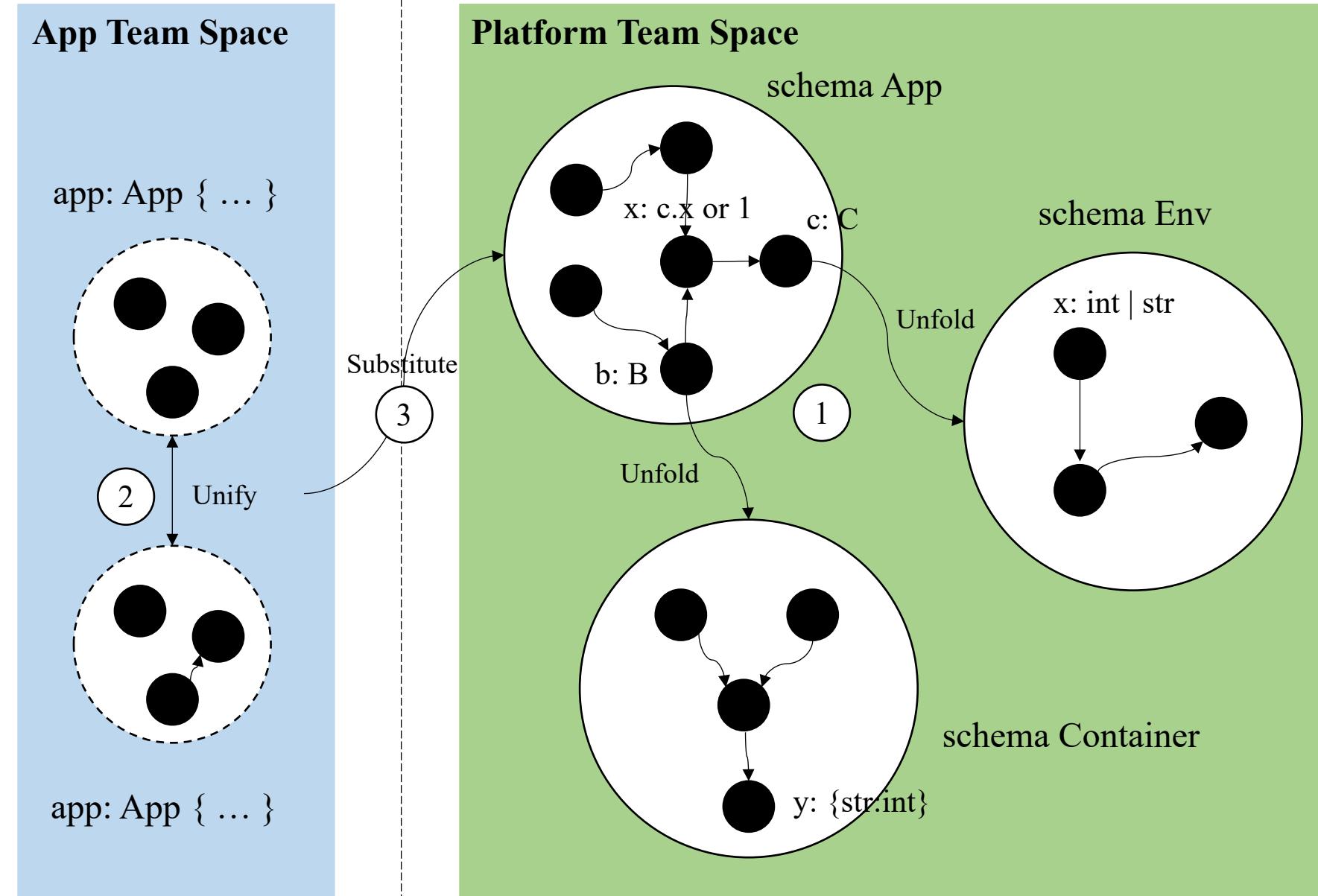


**KCL = Config + Schema + Rule + Lambda**



```
# Transform input resource 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

# Configuration

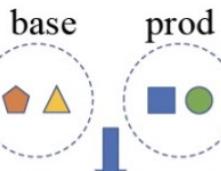


- base.k

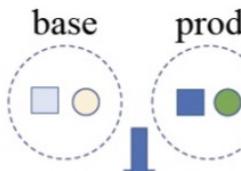
```
# Application Configuration
appConfiguration: frontend.Server {
    # Main Container Configuration
    mainContainer.ports = [
        {containerPort = 80}
    ]
    image = "nginx:1.7.8"
}
```

- prod.k

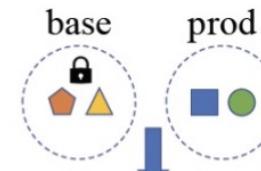
```
appConfiguration: frontend.Server {
    schedulingStrategy.resource = res.Resource {
        cpu = 100m
        memory = 100Mi
        disk = 1Gi
    }
}
```



Idempotent



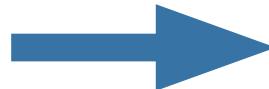
Patch



Error

Unique

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



✓ 填写基本属性元数据

2 填写 Schema 和校验规则（选填）

校验模式 ②

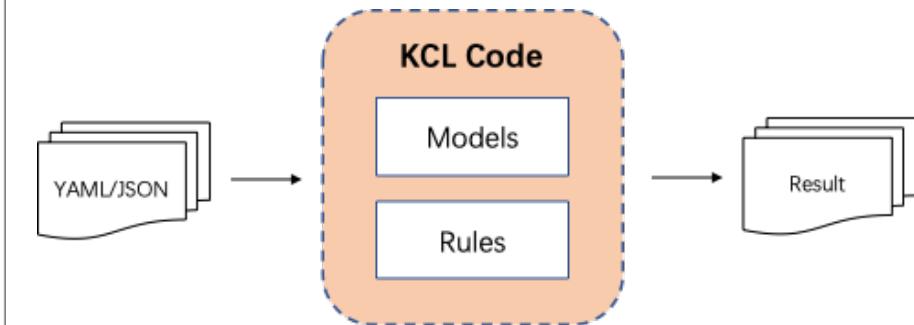
基础  KCL ↗

是否生效 ②

是  否

Schema或校验规则代码编写 ②

```
1 schema Product:
2     productCode: str # 产品码
3     productType: str # 产品类型
4     description: str # 产品描述
5     providers: [Provider] # 厂商
6     check:
7         len(productCode) <= 50
8         productType == "TYPE1" or productType == "TYPE2"
9
10    schema Provider:
11        providerCode: str # 供应商标识码
12        address: str # 供应商地址
13        legalPerson: str
14        description: str
15        check:
16            len(legalPerson)<=50
```



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

# Abstraction



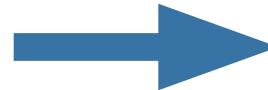
```
import .app

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

*Application Model*

*Docker Compose*

Transformer



```
services:
```

```
app:
```

```
    image: nginx
```

```
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
```

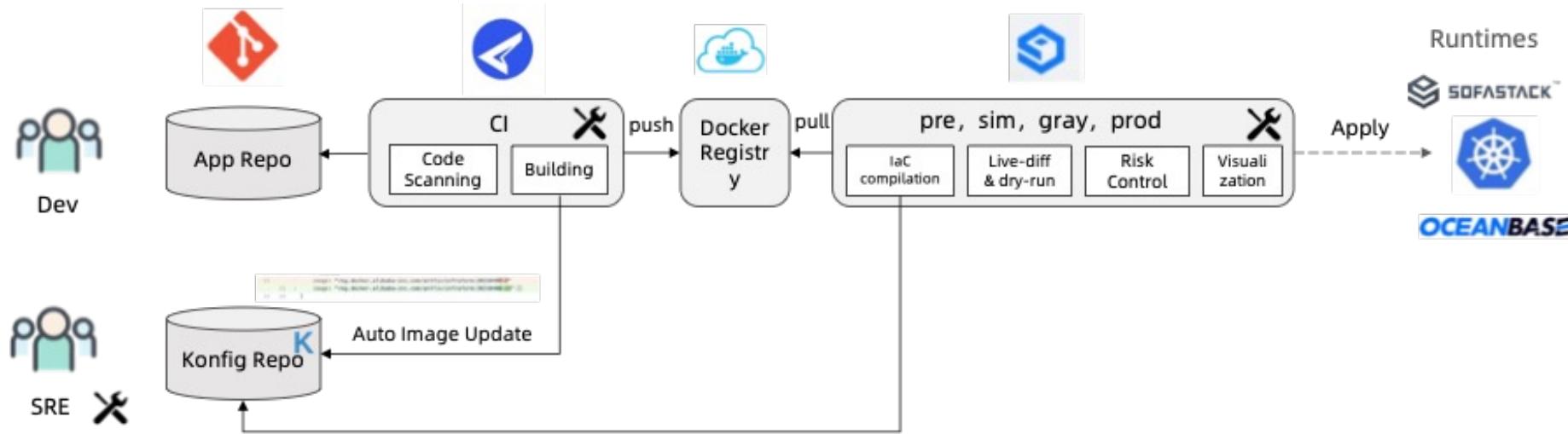
```
template:
```

*X*  
Transformer



Your Infra or Platform Spec

# Automation



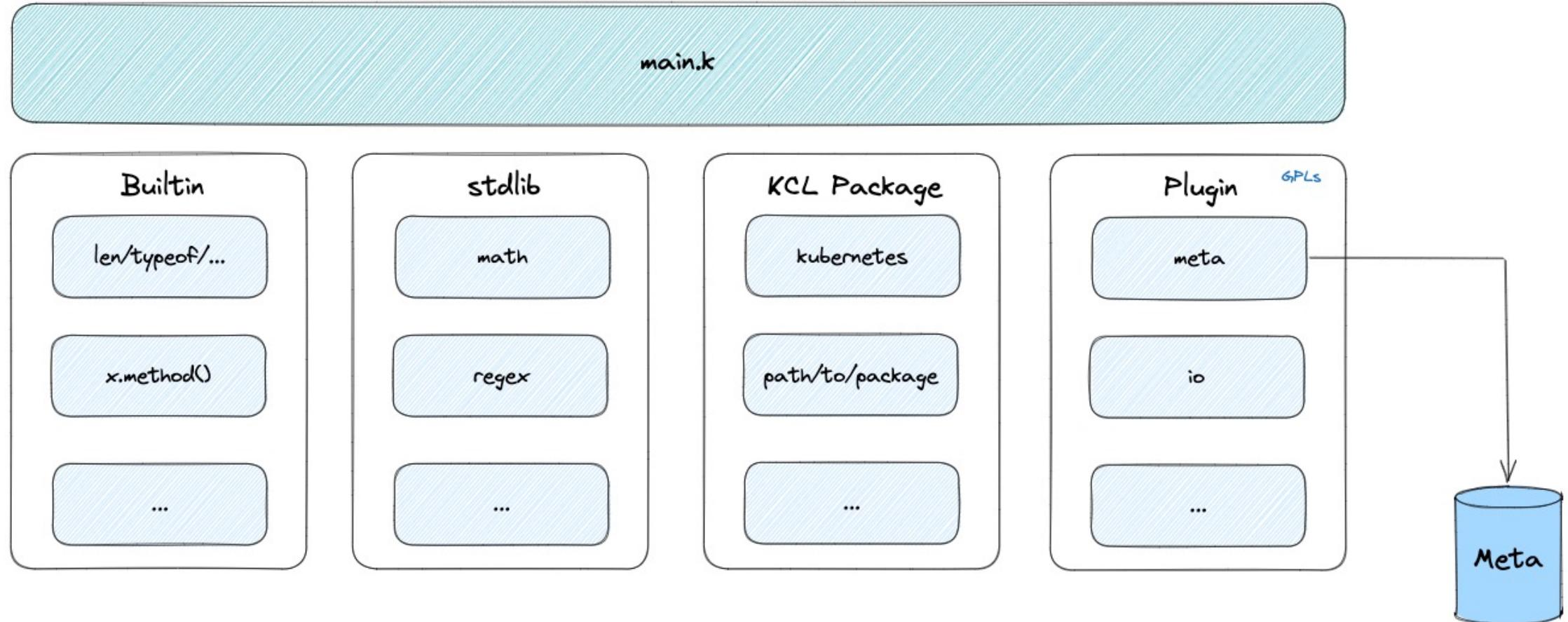
```
1 import base.pkg.kusion_models.kube.frontend
2 import base.pkg.kusion_models.kube.frontend.service
3 import base.pkg.kusion_models.kube.frontend.container
4 import base.pkg.kusion_models.kube.templates.resource as res_tpl
5
6 # Application Configuration
7 appConfiguration: frontend.Server {
8     # Main Container Configuration
9     mainContainer = container.Main {
10         ports = [
11             {containerPort = 80}
12         ]
13     }
14     image = "nginx:1.7.8"
15 }
16
```

```
1 import base.pkg.kusion_models.kube.frontend
2 import base.pkg.kusion_models.kube.frontend.service
3 import base.pkg.kusion_models.kube.frontend.container
4 import base.pkg.kusion_models.kube.templates.resource as res_tpl
5
6 # Application Configuration
7 appConfiguration: frontend.Server {
8     # Main Container Configuration
9     mainContainer = container.Main {
10         ports = [
11             {containerPort = 80}
12         ]
13     }
14+     image = "nginx:1.7.9"
15 }
16
```

*CI/CD Integration, GitOps Friendly*

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

# Modules



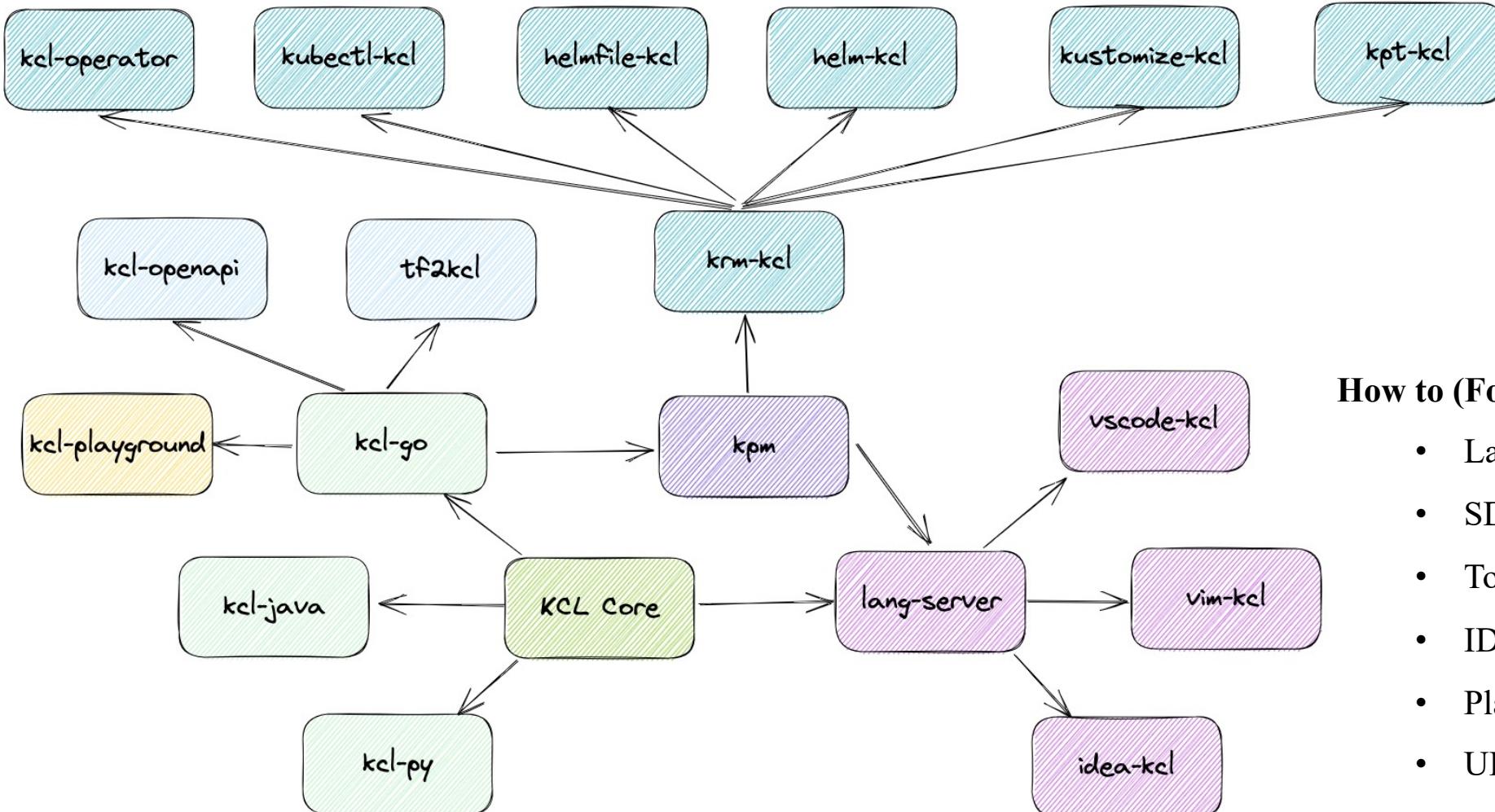
100+ built-in functions, modules with additional KPM package manager tools, OCI Registry and plugins

# Scenarios

---

03

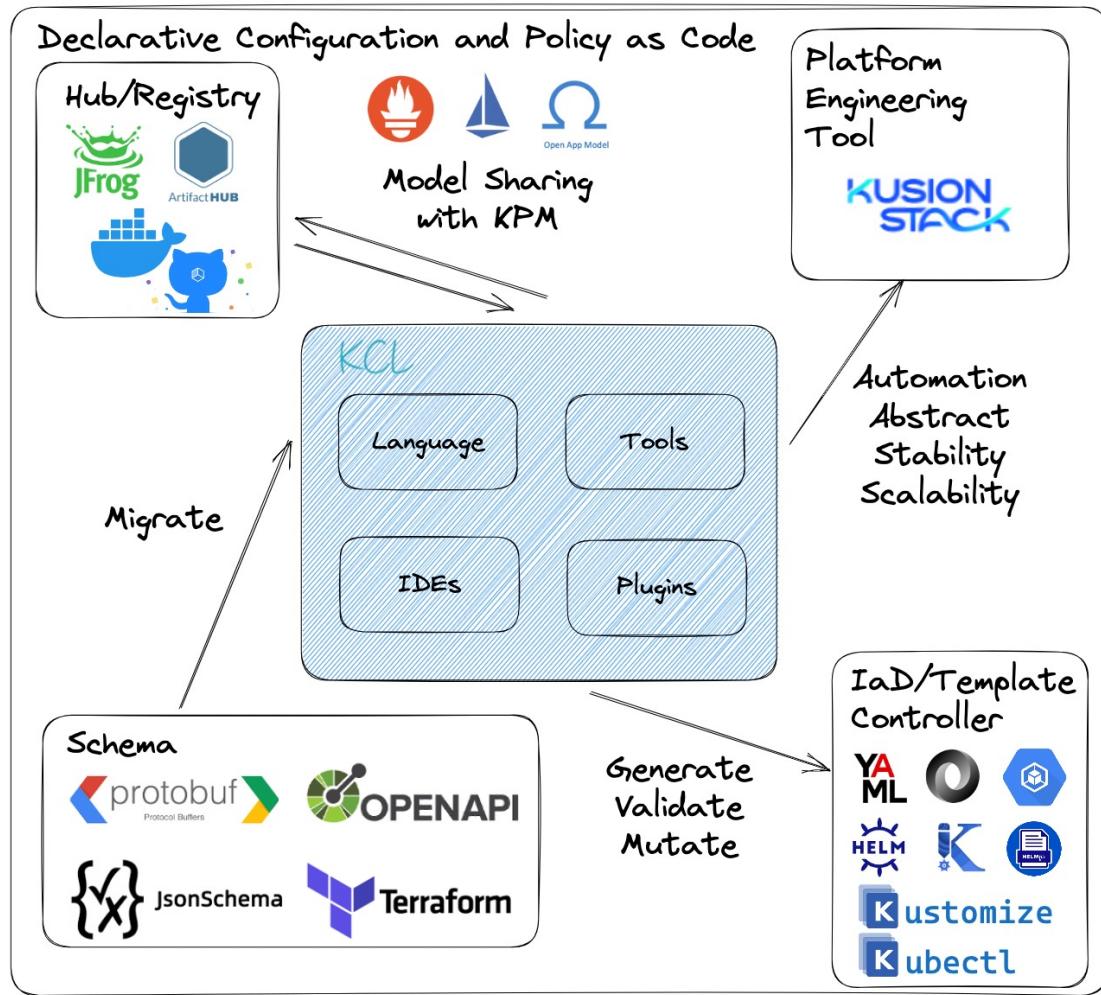
# Components



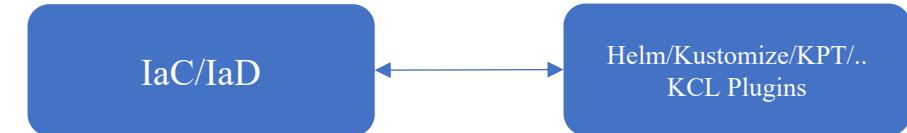
## How to (For App/Platform dev & SRE)

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

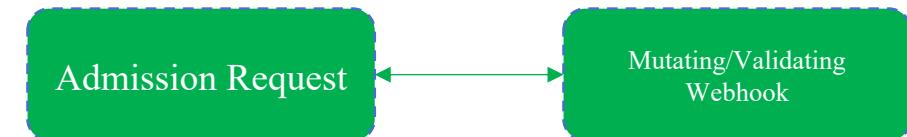
# Integrations



## Client



## Server



## Binding



## Enhancement/Glue

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

# Integrations



## KRM-KCL Specification

- Mutation

```
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: https-only-validation
  annotations:
    krm.kcl.dev/version: 0.0.1
    krm.kcl.dev/type: validation
  documentation: >-
    Requires Ingress resources to be present.
    include the `kubernetes.io/ingress.class` annotation.
    By default a valid TLS {} certificate will be generated
    optional by setting the `tls` field to true.
    More info: https://kubernetes.io/docs/concepts/services-networking/ingress/
spec:
  parameters:
  source:
    spec:
      source: oci://kusionstack/https-only-validation
```

- Abstraction

```
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: web-service
  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://kusionstack/web-service
```

## 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. it difficult to maintain abstraction, constraint, ...

## Tech.

- JSON
- YAML

## Product

- Kustomize
- ...

## Pros.

- Simple config logic support
- Dynamic argument input

## Cons.

- Increase of argument makes abstraction, constraint, ...
- Insufficient functionality e.g. ...

## 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

## Cons.

- Static type system & analysis
- High Performance

## Cons.

- Expansion of different models requires investment in R&D

## Tech.

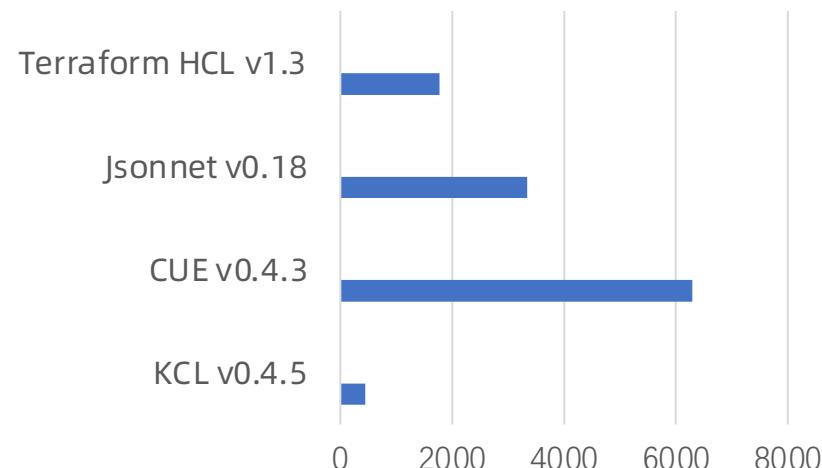
- KCL
- ...

## Product

- KusionStack
- ...

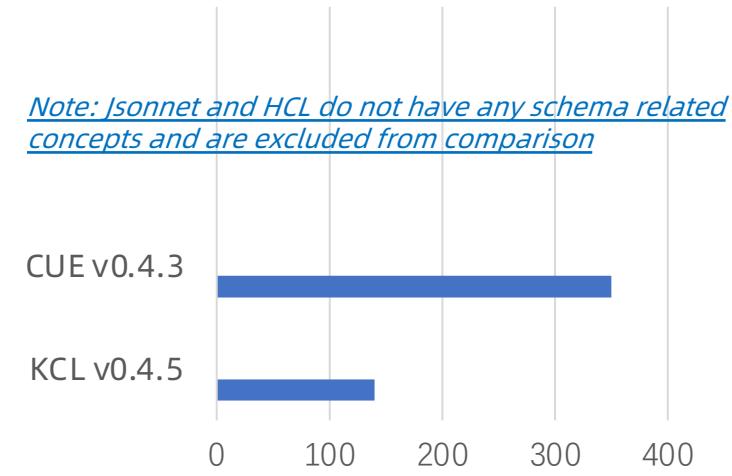
## Simple Computing

```
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



15	32	2000+	100+	700+
BG	BU	Projects	Clusters	Committers
1K/day	40K+	120K+	~1M	10K+/day
Pipelines	PRs	Commits	KCL Codes	KCL Compilations



Use the [KCL OpenKruise -> Application abstraction](#) to support the middleware containerization configuration management in over [100+](#) clusters



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



# Resources



- **Website**
  - <https://kcl-lang.io/>
- **GitHub Organization**
  - <https://github.com/kcl-lang/kcl>
- **Community**
  - <https://github.com/kcl-lang/community>
- **Twitter**
  - [@kcl language](https://twitter.com/kcl_language)

# THANKS