KCL: A Declarative Language for Large-scale Configuration and Policy Management

Xiaodong Duo, Pengfei Xu, Zheng Zhang, Shushan Chai, Rui Xia and Zhe Zong

AntGroup

October 28, 2022

Agenda

Background

01

Background

Configurations and Policies are important and crucial

Stability

Scalability

Efficiency

Time	Event
2021.07	The Bilibili website in China went down because SLB Lua configuration code fell into an infinite loop with calculation errors
2021.10	KT Company in South Korea suffers major network interruption nationwide due to wrong routing configuration

Why Design KCL

KUSION STACK

Growing importance of modeling, constraint and scalability

Structured KV → Templated KV → Programmable KV → Typed KV → Modeled KV

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
- …

Programmable KV

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

Cons.
- Insufficient type constraints
- Insufficient restraint ability
- Runtime error

Tech.
- GCL
- HCL
- JSONNET
- …

Product
- Terraform
- …

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
- …

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

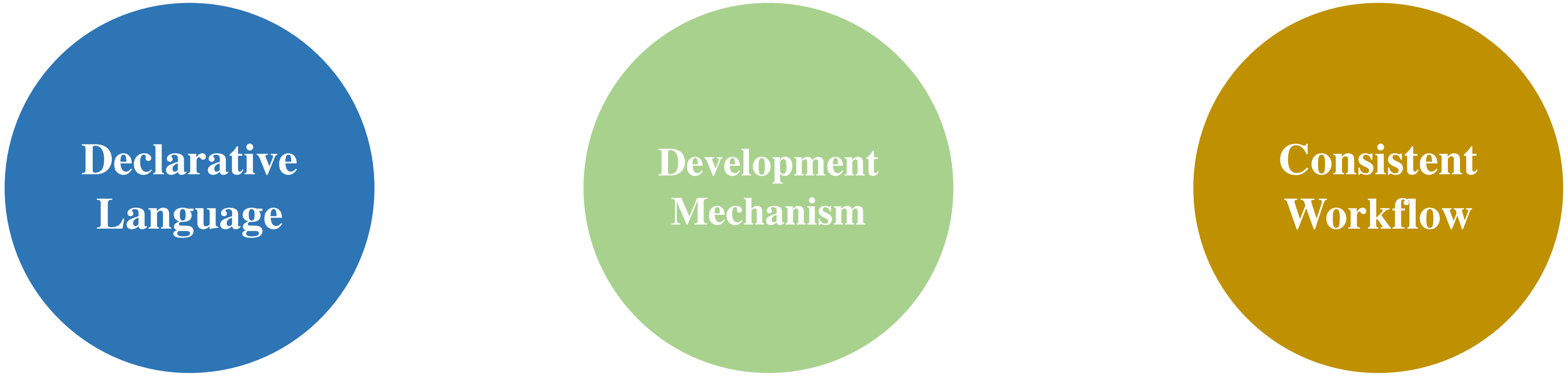
Tech.
- **KCL**
- …

Product
- KusionStack
- …

Contribution

- Proposing the **KCL declarative language, development mechanism, and consistent workflow** to improve the large-scale efficiency and liberate multi-team collaborative productivity of operational development and operation systematically while ensuring stability for large-scale configuration and policy management.

- To date, the KCL has been used in more than 800 projects, and the average configuration writing and distributing time is shortened from more than 25 days to 2 days.



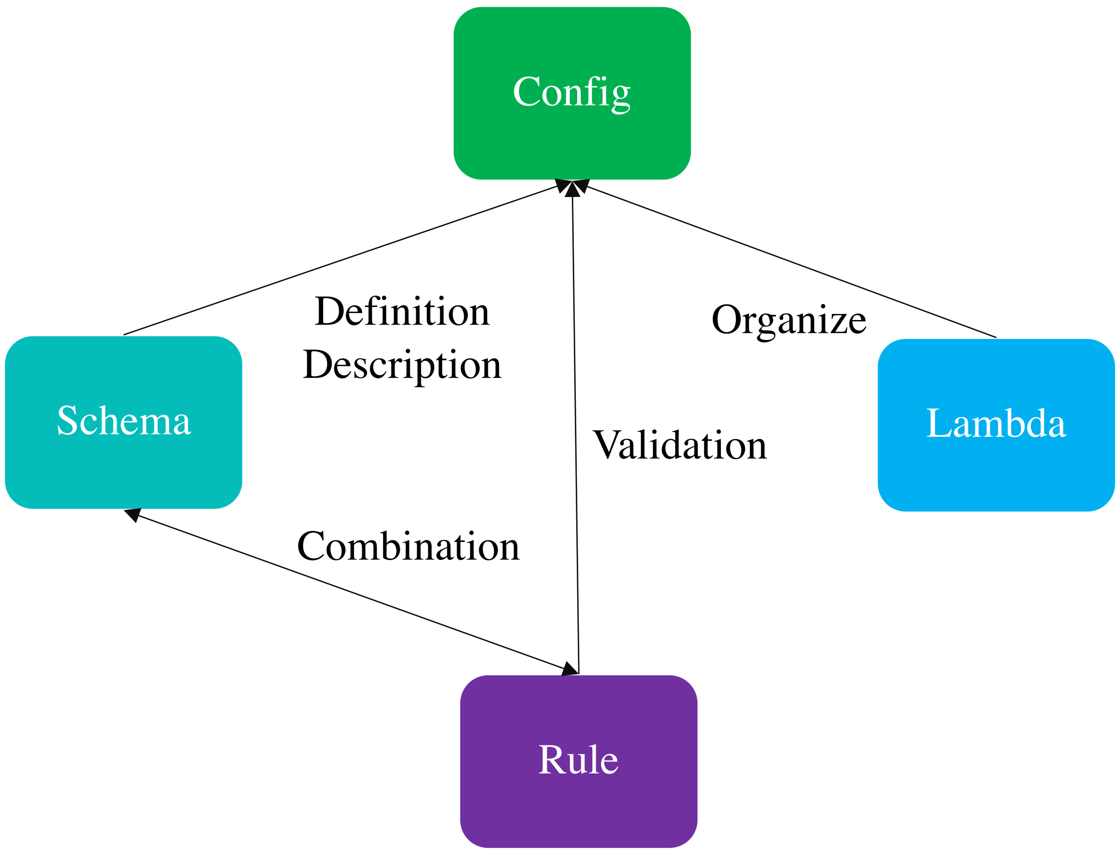
Design

02

Overview

KUSION STACK

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




```
import units

type UnitType = units.NumberMultiplier

sc
—
}
```

```
import kubernetes.core.v1

deployment = v1.Deployment {
    metadata.name = "nginx-deployment"
    metadata.labels.app = "nginx"
    spec = {
        replicas = 3
        selector.matchLabels.app = "nginx"
```

```
mixin OverQuotaMixin:
    overQuotaToleration = Toleration {
        key = "sigma.ali/is-over-quota"
        operator = "Equal"
        value = "true"
        effect = "NoSchedule"
    } if data.overQuota else Undefined

    overQuotaMatchExpression = NodeSelectorRequirement {
        key = "sigma.ali/is-over-quota"
        operator = "In"
        values = ["true"]
    } if data.overQuota else Undefined
```

https://github.com/KusionStack/KCLVM

Grammar

KUSIONSTACK

$$
\begin{aligned}
KCL &::= stmt^* \\
stmt &::= schema \mid assert \mid import \mid assign \mid \ldots \\
schema &::= [\textbf{schema}]\ id[`('parent\_id`)']\ [`['arguments`]']`:'schema\_body \\
schema\_body &::= [mixin\_stmt]\ [schema\_context] \\
&\quad\ [check\_block] \\
mixin\_stmt &::= \textbf{mixin}\ `['\ id^*\ `]' \\
schema\_context &::= [attr|if \mid assign \mid assert \mid expr]^* \\
attr &::= id[`?']\ `:'\ Type\ [`='expr] \\
check\_block &::= \textbf{check}\ `:'\ check\_stmt^* \\
check\_stmt &::= expr[`,'\ msg] \\
assign &::= id\ `='\ expr \\
assert &::= expr[`,'\ msg] \\
\ldots &\quad\ \ \ldots
\end{aligned}
$$

Fig. 1: (Subset of) KCL grammar

Core Pattern $k\ op\ (T)\ v,$

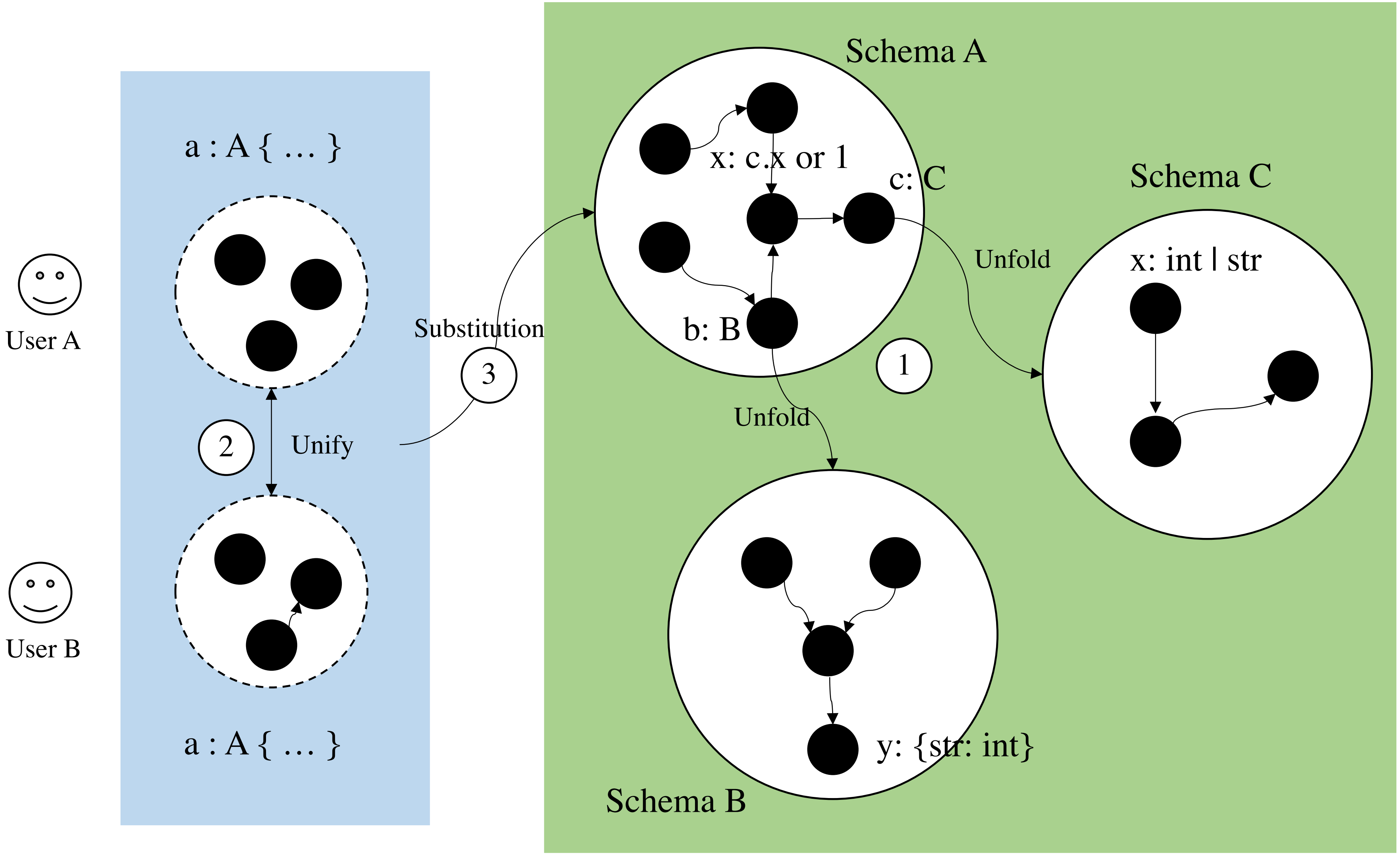
Type

Table 1: KCL type notions

Type notion	Notation
Boolean	$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash boolean}$
Integer	$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash integer}$
Float	$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash float}$
String	$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash string}$
Literal	$\frac{c \in \{boolean, integer, float, string\}}{\Gamma \vdash literalof(c)}$
List	$\frac{\Gamma \vdash T \quad T \neq Void}{\Gamma \vdash listof(T)}$
Dict	$\frac{\Gamma \vdash T_1 \quad \Gamma \vdash T_2 \quad T_1 \neq Void \quad T_2 \neq Void}{\Gamma \vdash dictof(T_k = T_1, T_v = T_2)}$
Structure	$\frac{\Gamma \vdash T_1 \quad \ldots \quad \Gamma \vdash T_n \quad T_i \neq Void \quad K_i \neq K_j, \text{when } i \neq j}{\Gamma \vdash structof(K_1 : T_1, \ldots, K_n : T_n)}$
Union	$\frac{\Gamma \vdash T_1 \quad \ldots \quad \Gamma \vdash T_n \quad T_i \neq Void}{\Gamma \vdash unionof(T_1, \ldots, T_n)}$
Void	$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash Void}$
Any	$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash Any}$
Nothing	$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash Nothing}$

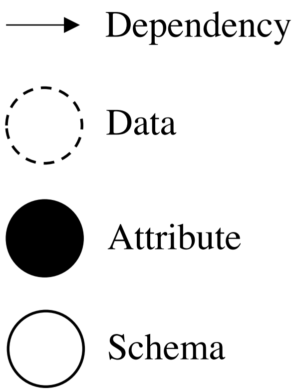
Well-formed type $\Gamma \vdash \Diamond$.

Compilation



✓ DAG Model

✓ Irrelevant Order



Configuration Substitution

Algorithm 1: The configuration substitution process.

Input: the input configuration C_i
Output: the merged configuration C_o

```
C_o ← {} ;
cache ← {} ;
foreach k, e in C_i do
    if k in cache then
        C_o[k] = cache[k];
    else
        v = lookup(k, e);
        cache[k] = v;
    end
end

function lookup(k, e)
v ← ∅ ;
foreach k_d in dependence(e) do
    if k_d in cache then
        v = cache[k_d];
    else
        v = lookup(k_d, C_i[k_d]);
    end
end
return v;
end
```

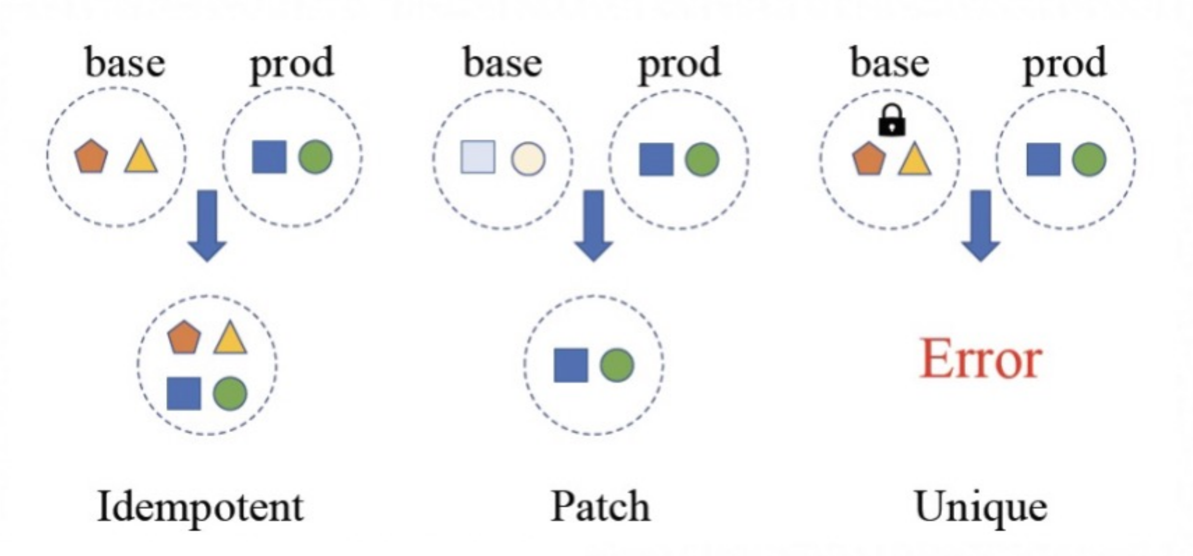
Main Steps

1. Traverse all the key k and expression e of the input configuration C_i, and use the lookup function to substitute its dependent value recursively.

2. Store the calculated value in the output configuration and cache, which is used to avoid multiple calculations.

3. When all the key calculations are completed, we get the substitution completed configuration C_o.

Configuration Merge



Pattern $$\sigma = \sum_{i=1}^{N} \{k_i, v_i, o_i\},$$

Merge $$\sigma_u \cup \sigma_v = \sum_{j=1}^{N_u+N_v} \{k_j, v_j, o_j\}.$$

Result $$\sum_{k=1}^{M} \{k_s, v_k, o_k\} = \{k_s, \{v_1, o_1\} \oplus \{v_2, o_2\} \oplus \ldots \oplus \{v_M, o_M\}\},$$

Configuration Properties

$$\sum_{k=1}^{M}\{k_s, v_k, o_k\} = \{k_s, \{v_1, o_1\} \oplus \{v_2, o_2\} \oplus ... \oplus \{v_M, o_M\}\},$$

where $\oplus$ denotes the entry value merge operator.

Properties

- When the operation o_1 is the unique configuration operation, the calculation is invalid.

- When the operation o_1 is not the unique configuration operation, the configuration entry on the right side of the $\oplus$ operator has a higher priority, so we can get $\{v_1, o_1\} \oplus \{v_2, o_2\} = \{v_1 \oplus v_2, o_2\}$

- When o_2 is an idempotent merge operation, there is a commutative law $\{v_2 \oplus v_1, o_2\} = \{v_1 \oplus v_2, o_2\}$ [37], and when the recursive partial order relationship $v_2 \subseteq v_1$ is satisfied, the calculation is valid. Besides, the partial

Configuration Properties

order calculation of two values can be defined as

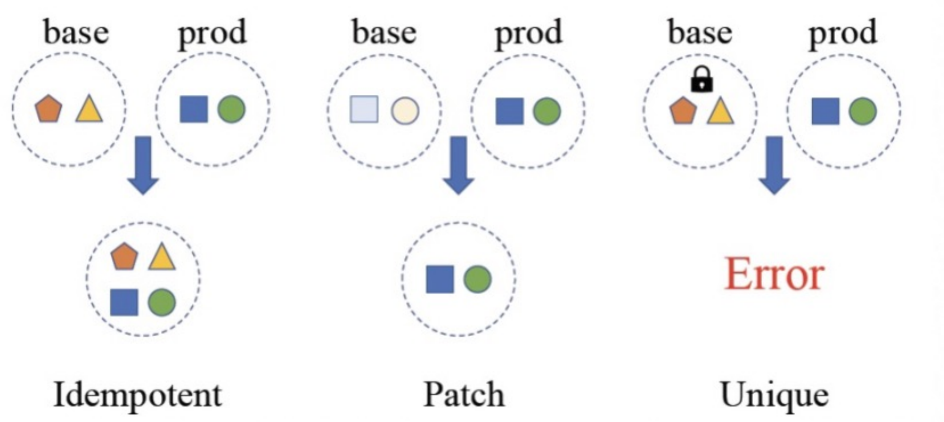
$$
v_2 \subseteq v_1 \Leftrightarrow \begin{cases} v_2 == v_1 \ if \ v_1 \in \{Int, Float, Boolean, String\} \\ \text{len}(v_2) == \text{len}(v_1) and \ \forall \ i, \ \text{item}(v_2, i) \subseteq \text{item}(v_1, i) \\ \qquad if \ v_1 \in List \ and \ if \ v_2 \in List \\ \forall \ k, \text{item}(v_2, k) \subseteq \text{item}(v_1, k) \\ \qquad if \ v_1 \in \{Dict, Structure\} \ and \\ \qquad if \ v_2 \in \{Dict, Structure\} \end{cases}
$$

where i denotes the index of the list element, len denotes the length of the list element, k denotes the key of the dict and structure element, and item denotes the item of the list, dict and structure element.

– When the operation o_2 is an overwrite operation, $\{v_1 \oplus v_2, o_2\} = \{v_2, o_2\}$.

– When the operation o_2 is an append operation, it will try to add v_2 to the list of v_1, and stop the calculation when v_1 is not a list type.

Example



- 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
    }
}
```

Automation

kcl -O appConfiguration.image='"nginx:1.7.9"'

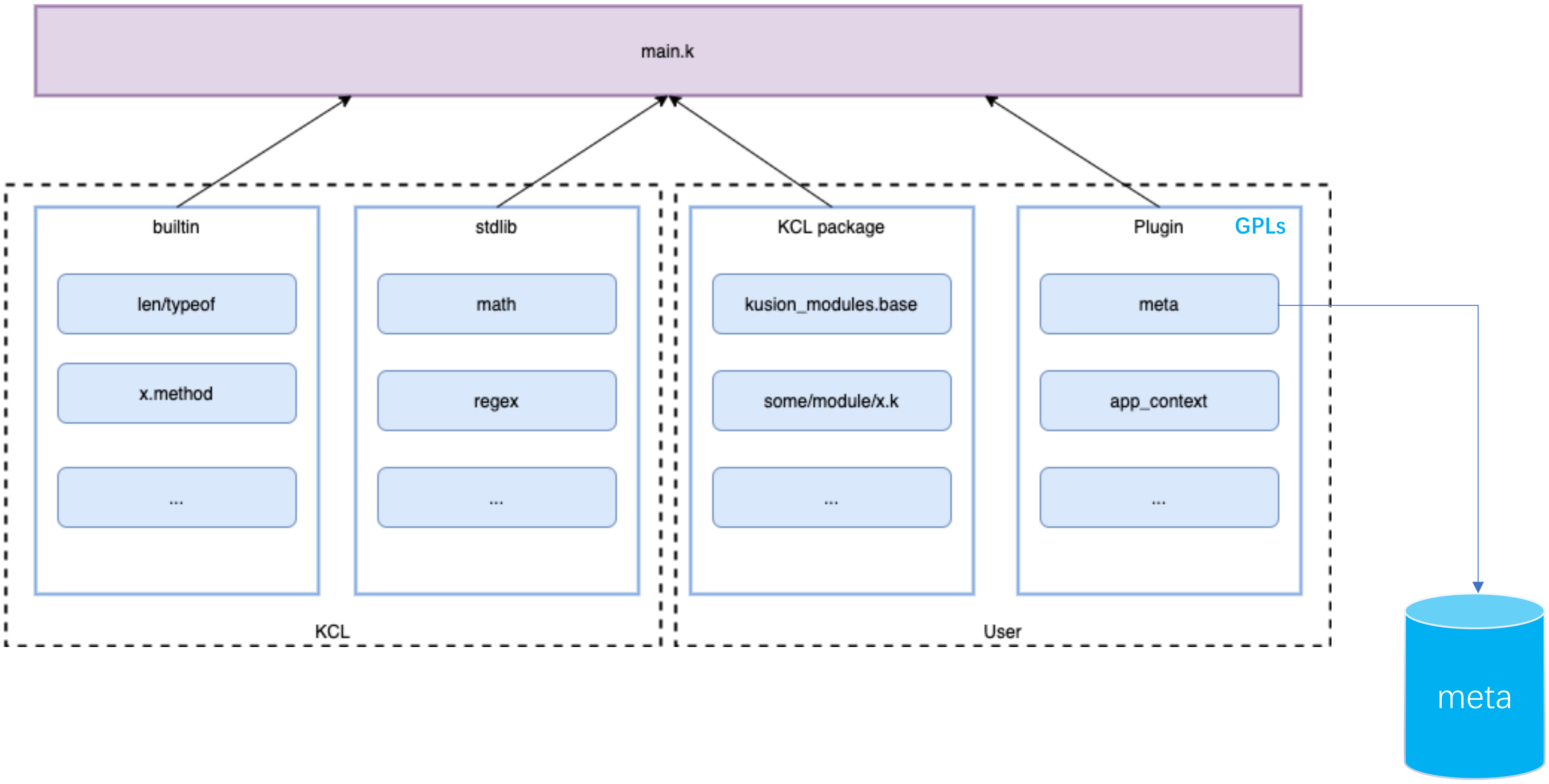
```
import base.pkg.kusion_models.kube.frontend
import base.pkg.kusion_models.kube.frontend.service
import base.pkg.kusion_models.kube.frontend.container
import base.pkg.kusion_models.kube.templates.resource as res_tpl

# 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"
}
```

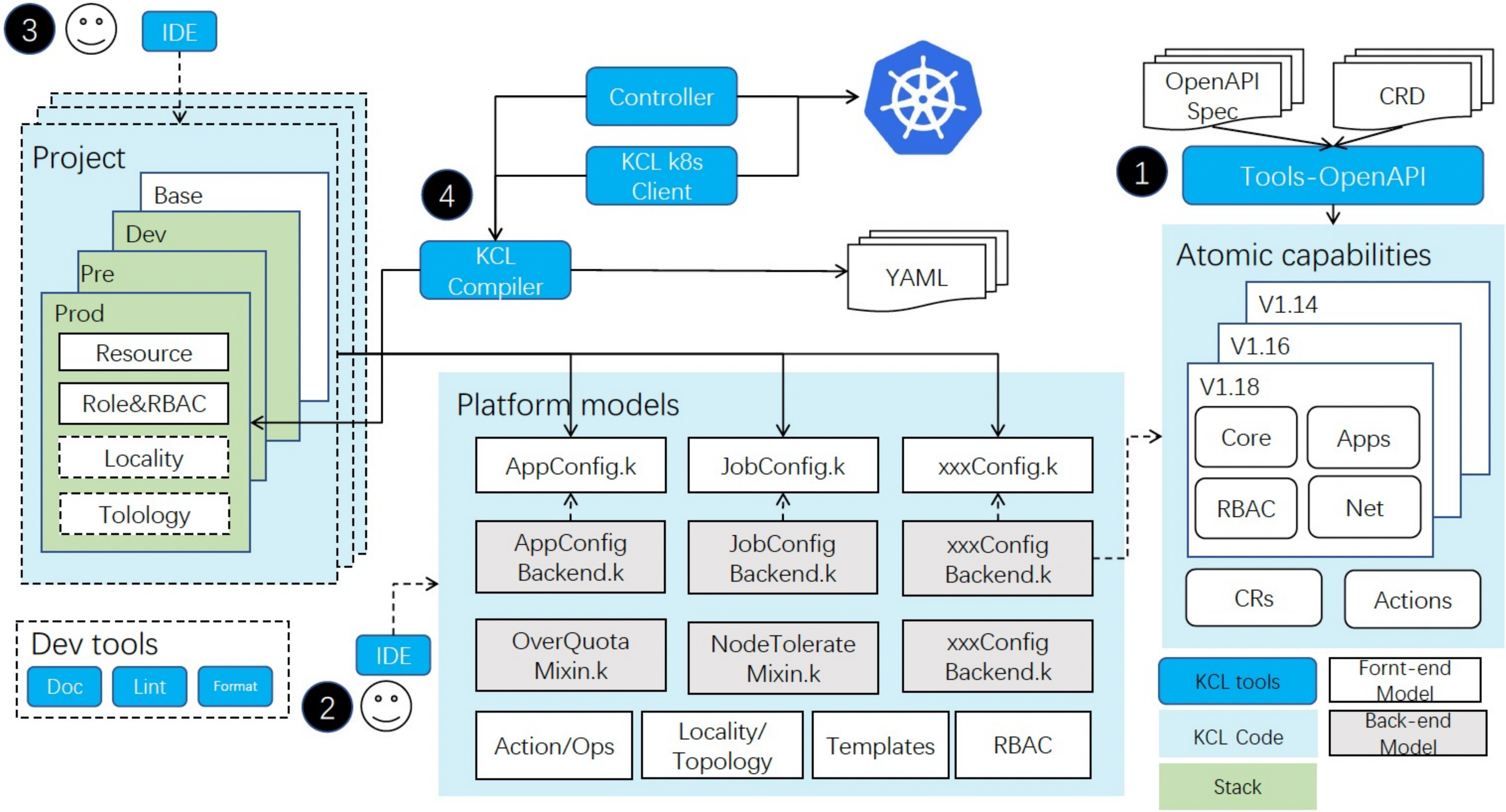
Modules


main.k
builtin
len/typeof
x.method
...
stdlib
math
regex
...
KCL
KCL package
kusion_modules.base
some/module/x.k
...
Plugin
GPLs
meta
app_context
...
User
meta

Workflow

03

Workflow



3
IDE
Project
Base
Dev
Pre
Prod
Resource
Role&RBAC
Locality
Tolology
Dev tools
Doc
Lint
Format
IDE
2
4
Controller
KCL k8s
Client
KCL
Compiler
YAML
Platform models
AppConfig.k
JobConfig.k
xxxConfig.k
AppConfig
Backend.k
JobConfig
Backend.k
xxxConfig
Backend.k
OverQuota
Mixin.k
NodeTolerate
Mixin.k
xxxConfig
Backend.k
Action/Ops
Locality/
Topology
Templates
RBAC
OpenAPI
Spec
CRD
1
Tools-OpenAPI
Atomic capabilities
V1.14
V1.16
V1.18
Core
Apps
RBAC
Net
CRs
Actions
KCL tools
Fornt-end
Model
KCL Code
Back-end
Model
Stack

Evaluation

04

Related Works

Table 2: Features of configuration languages

	KCL	GCL	CUE	Jsonnet	HCL	Dhall
Variables	✓	✓	✓	✓	✓	✓
Reference	✓	✓	✓	✓	✓	✓
Data types	✓	✓	✓	✓	✓	✓
Schema	✓	✓	✓	✗	✗	✓
Inherited	schema	tuple	✗	obejct	data	data
Arithmetic&Logic	✓	✓	✓	✓	✓	✓
Loop	list/dict/schema comprehension	list comprehension	list comprehension	list/object comprehension	for splat (comprehension)	list generate function
Conditional	✓	✓	✓	✓	✓	✓
Built-in function	✓	✓	✓	✓	✓	✓
Function definition	✓	✗	✗	✓	✗	✓
Import	✓	✓	✓	✓	✓	✓
Type check	✓	✓	✓	✗	✗	✓
Testable	✓	✗	✗	✓	✗	✓
Mixin	✓	✗	✗	✗	✗	✗
Data integration	✓	✗	✗	JSON	✗	✗
Dynamic configuration	✓	✗	✗	✓	✗	✗
Policy	✓	✗	✗	✗	✗	✗
Merge	idempotent merge/ patch/ unique configuration	patch	idempotent merge	patch	✗	idempotent merge/ patch

Key Results

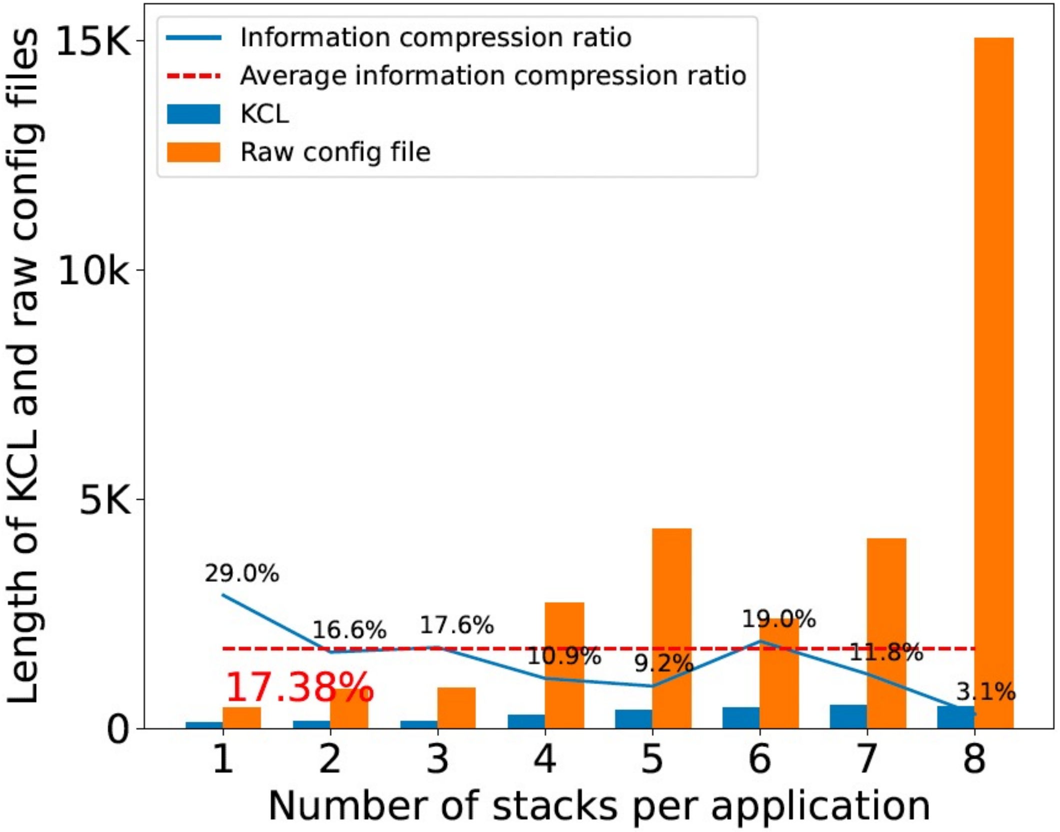


Fig. 10: Information compression ratio of KCL and raw config files

Key Results

10 K/day

KCL
Compilations

~600 K

KCL
Code

~18 K

PRs

800

Projects

25 days –
2 days

Average
configuration writing
and shipping time

https://kusionstack.io/blog/2022-sense-of-open-day

Future Work

Policy

- We will better support policy capabilities such as logic writing and data query to satisfy scenarios such as service authentication.

Automati on

- We will provide a more complete KCL tool-set including create, query and update to meet more automation scenarios.

Security

- We will improve language security through static model checking and theorem proving and let more problems be exposed to compile time as much as possible.

Summary

- We proposed the **KCL declarative language**, which is an open source constraint-based record & functional language mainly used in configuration and policy scenarios.
- In KCL, we are making special design including **modeling, constraint and workflow for the stability and scalability** of configuration, including configuration graph unification and carrying multiple configuration merging strategies.
- To date, the KCL has been used in more than **800 projects**, and the average configuration writing and distributing time is shortened from more than **25 days to 2 days**. We have demonstrated the feasibility of our contribution through a large number of code practices and achievements.

https://github.com/KusionStack/KCLVM

THANKS