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

Pengfei Xu (Ant Group)

Aug. 2023

Agenda

01 Background

02 Concepts

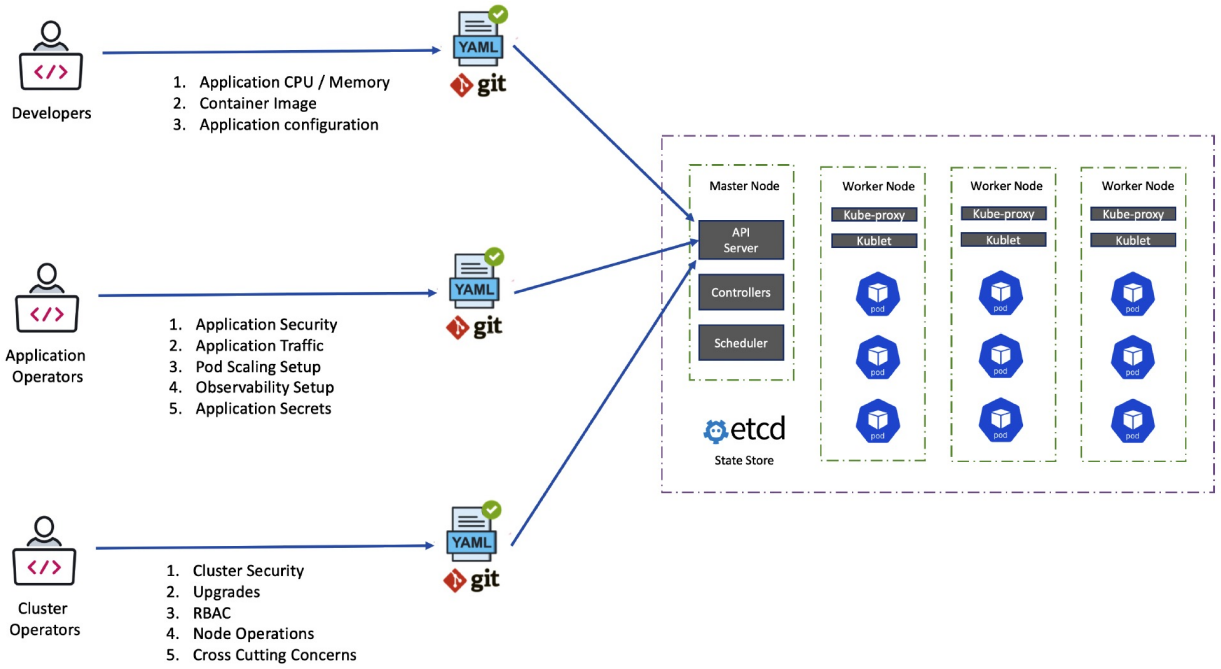
03 Experience

04 Appendix

Background

01

Background



Cognitive Loading

- Complex infrastructure and platform concepts
- Kubernetes is a platform of platform, lack of lightweight declarative configuration composition methods at client side

Static Config

- YAML Bloat
- Burdensome and configuration drift for all involved teams

Low Efficiency/Reliability

- Lack of standardized testing and validation methods
- Lack of effective tools for team collaboration

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#
CNCF Platforms White Paper: https://tag-app-delivery.cncf.io/whitepapers/platforms/
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 management

- ***Enhancement for configuration tools e.g., Helm, Kustomize, kpt.***
  - Mutation
  - Validation

Concepts

02

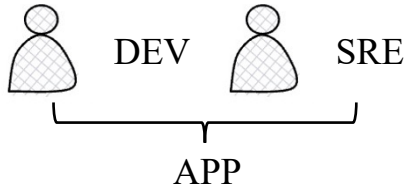
Configuration *Standalone KCL*

OCI Registry

- Abstraction

```
import .app

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

Application Schema



Docker Compose
Transformer

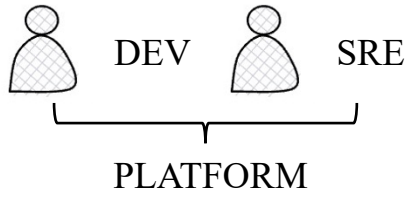
```
services:
  app:
    image: ngnix
    ports:
      - published: 80
        target: 80
        protocol: TCP
```

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



- Validation: Use KCL schema and rule to validate resources
- Mutation: Use KCL Function to transform resources.

- ✓ *Declarative and application-centric configuration interface that developers can understand*
- ✓ *Scalable and dynamic configuration management*
- ✓ *Separation of concerns*
- ✓ *Real time errors/warnings with schema and constraints*
- ✓ *Package management and OCI Registry support*

Configuration *KRM KCL*

- **Mutation**

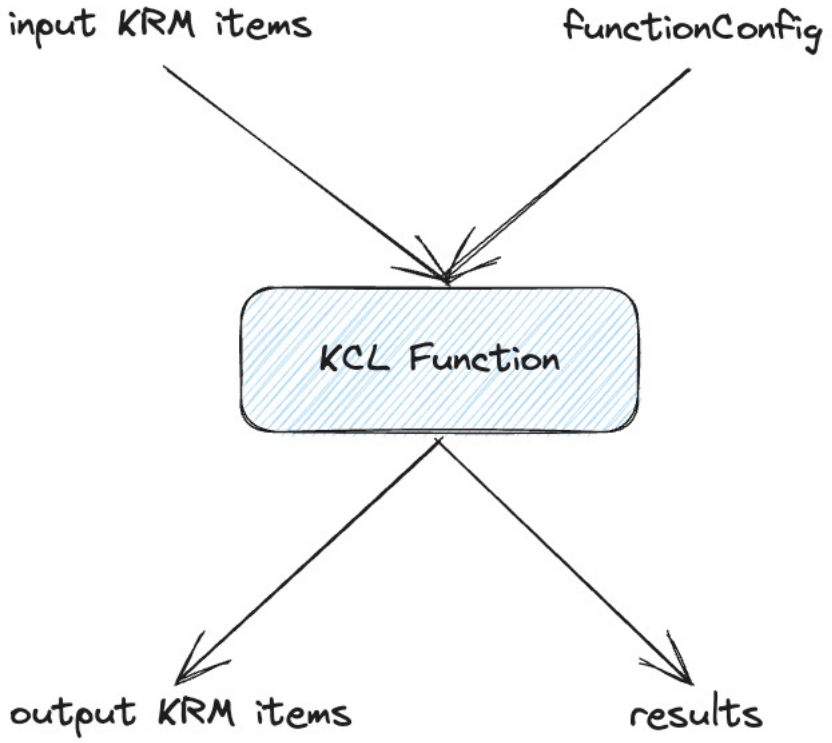
```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: set-annotations
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: mutation
      documentation: >-
        Add or change annotations
spec:
  params:
    toAdd: addValue
  source: oci://ghcr.io/kcl-lang/set-annotation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

- **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**

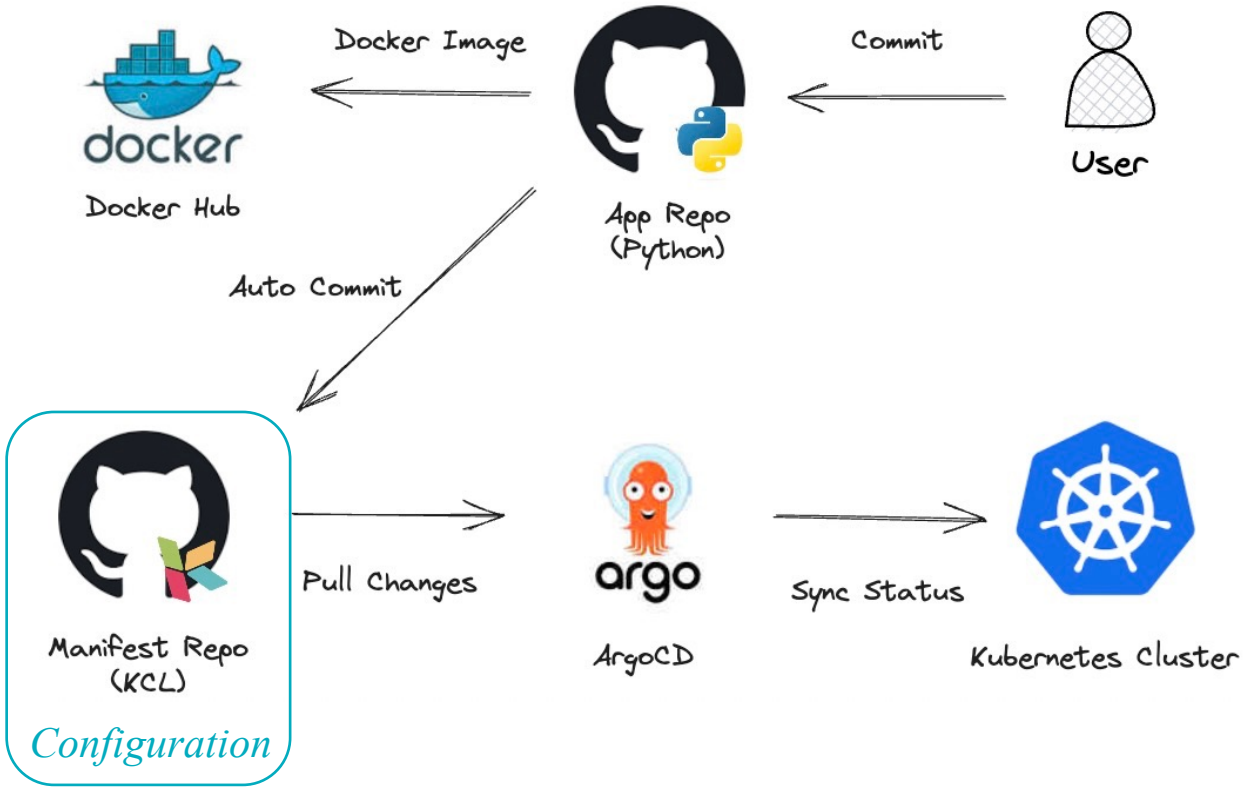
```yaml
apiVersion: krm.kcl.dev/v1alpha1
kind: KCLRun
metadata:
  name: https-only
  metadata:
    annotations:
      krm.kcl.dev/version: 0.0.1
      krm.kcl.dev/type: validation
      documentation: >-
        Requires Ingress resources to be HTTPS only.  Ingress resources must
        include the `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
```



- ***Unified Spec with KCL Function***
- ***Programmable and Extensible***
- ***Multiple Source Support****: OCI, Git, Https, …*

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

Automation



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**.

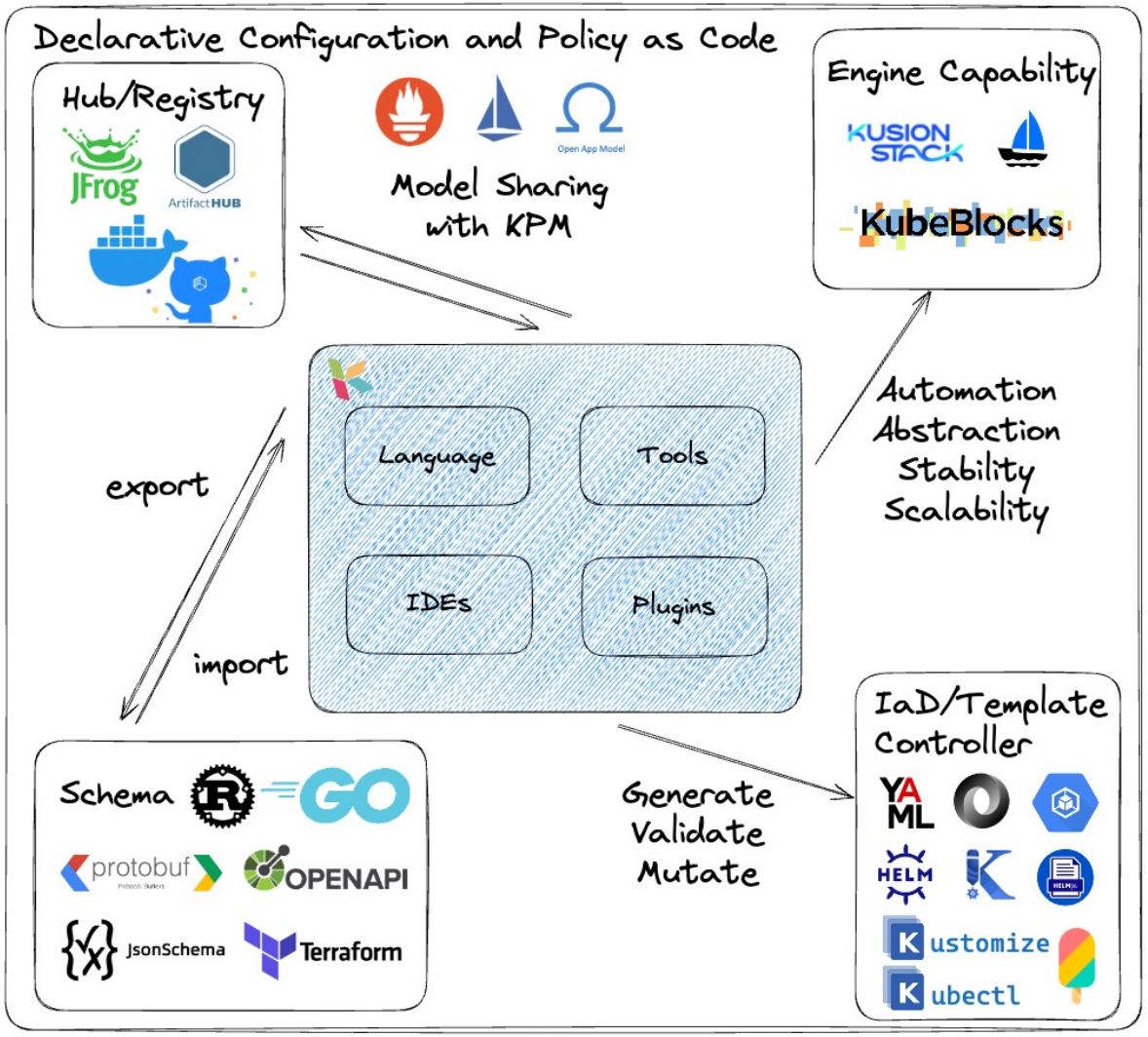
2 main.k

```
@@ -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      }
```

Configuration source driven workflow: Supports both standalone and KRM KCL form
CI/CD Integration, GitOps Friendly

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

Integrations



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

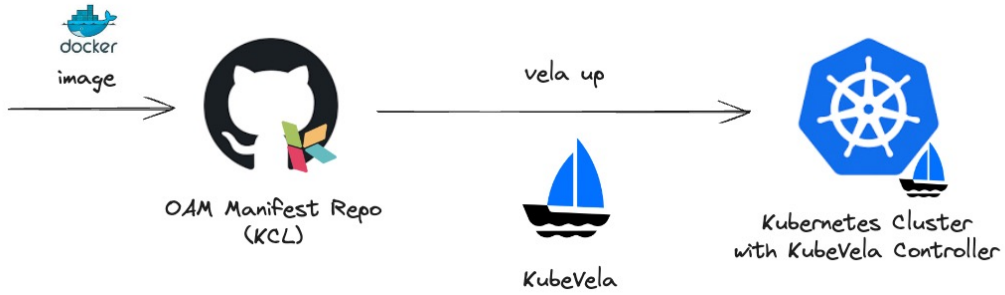
Server Admission Request Mutating/Validating Webhook ↔ KCL Operator

SDKs Rust, Go, Java, Python

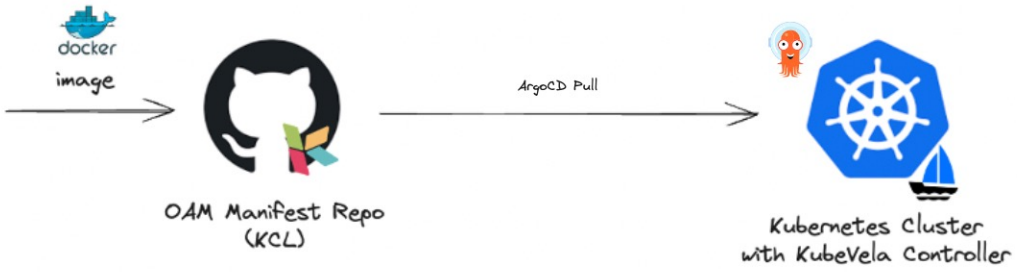
- ***Multi-Lang Binding***: *Go SDK, Python SDK, Java SDK, etc.*
- ***Package Management***: *KPM Tool & Registry*
- ***Data and Schema Migration***: *KCL Import tool*
- ***Cluster Integration***: *KCL Operator instead of deploying webhooks.*
- ***KRM Support***: *Unified spec and multiple tools e.g., kubectl-kcl plugin, helm-kcl plugin, kustomize-kcl plugin, kpt-kcl-plugin, crossplane-kcl-function …*
- ***Engine tool integrations: As the Dynamic Configuration Management (DCM) language to deliver applications with KusionStack, KubeVekla***

KubeVela/OAM Integration

- Workflow1



- Workflow2



- Create a new project and add OAM dependencies.

```
kcl mod init kcl-play-svc && cd kcl-play-svc && kcl mod add oam
```

- Write the following code in main.k.

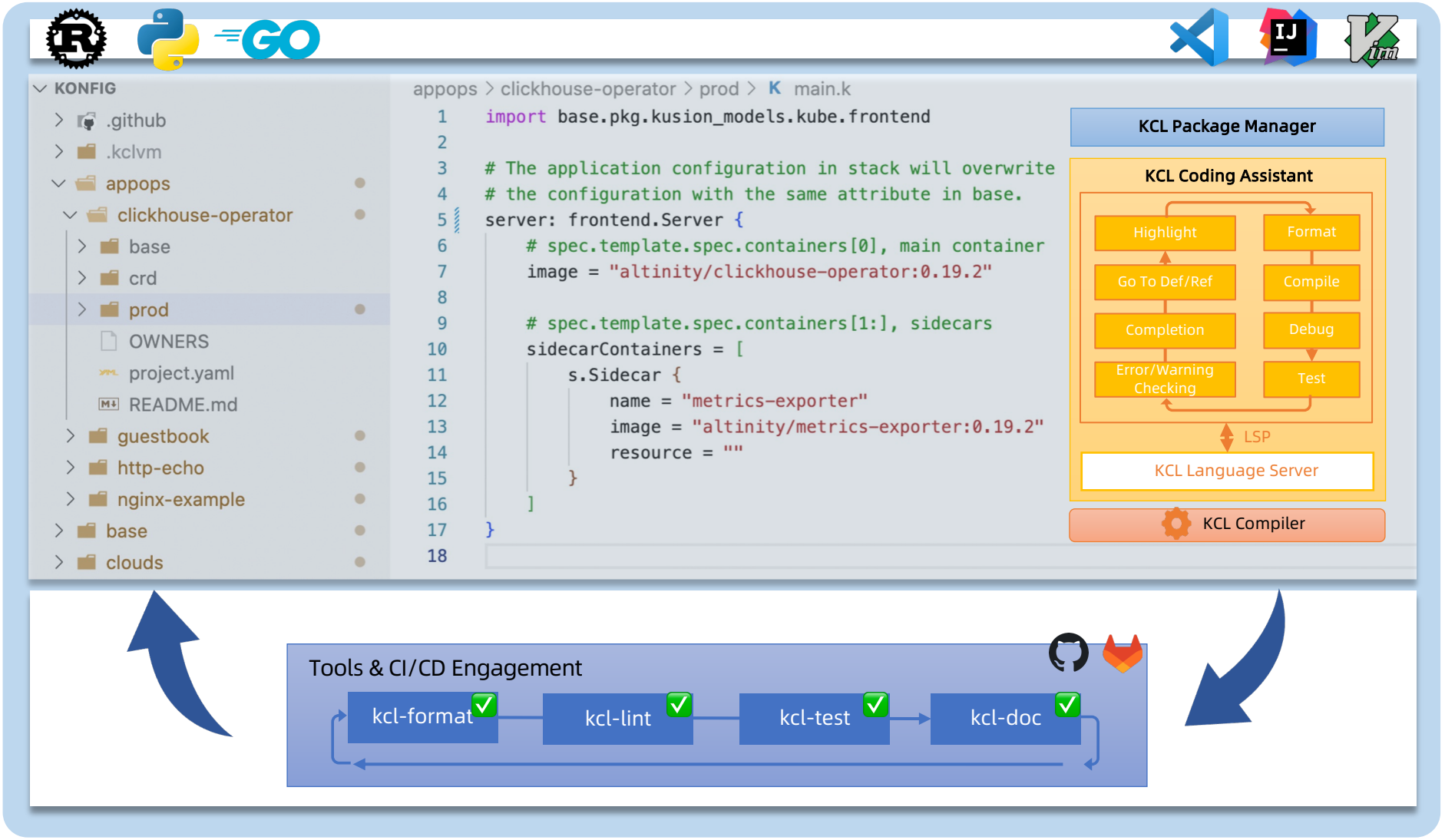
```
import oam

oam.Application {
    metadata.name = "kcl-play-svc"
    spec.components = [{
        name = metadata.name
        type = "webservice"
        properties = {
            image = "kcllang/kcl"
            ports = [{port = 80, expose = True}]
            cmd = ["kcl", "play"]
        }
    }]
}
```

Experience

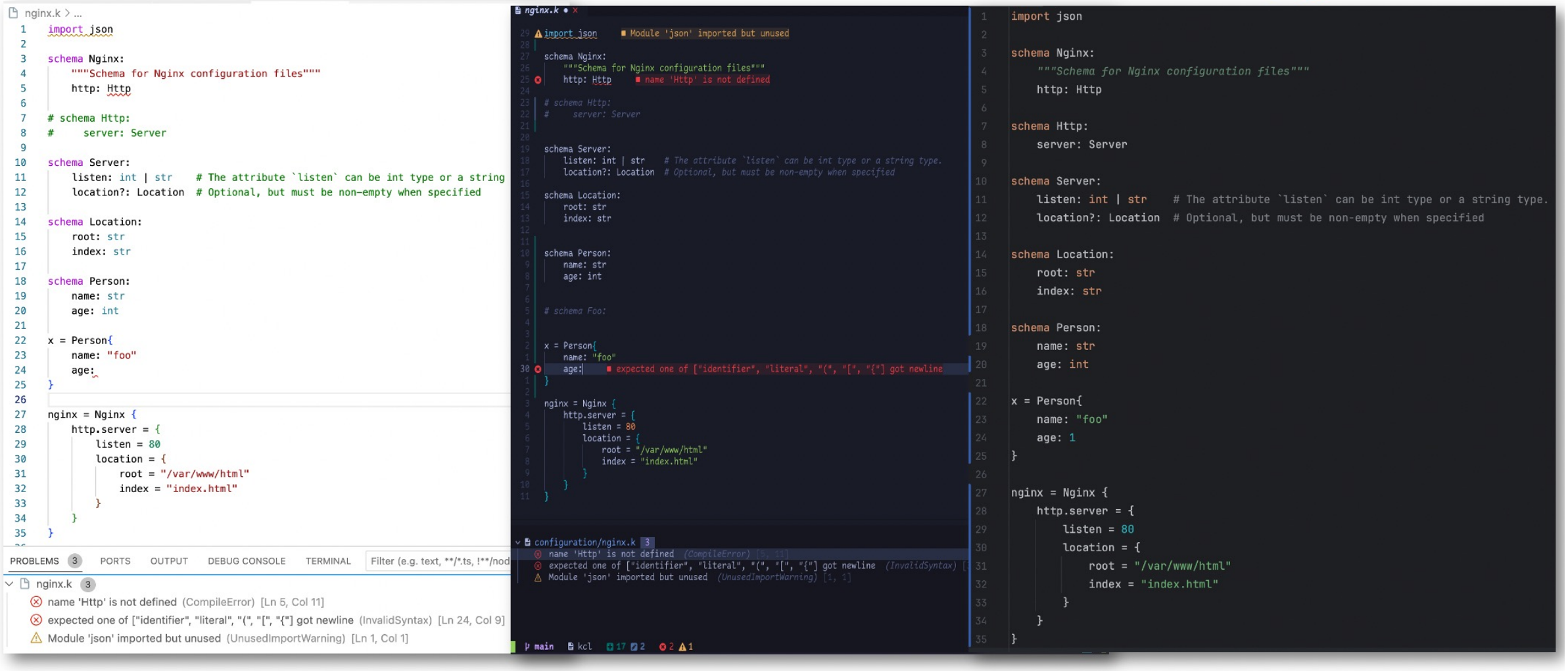
03

Workspace

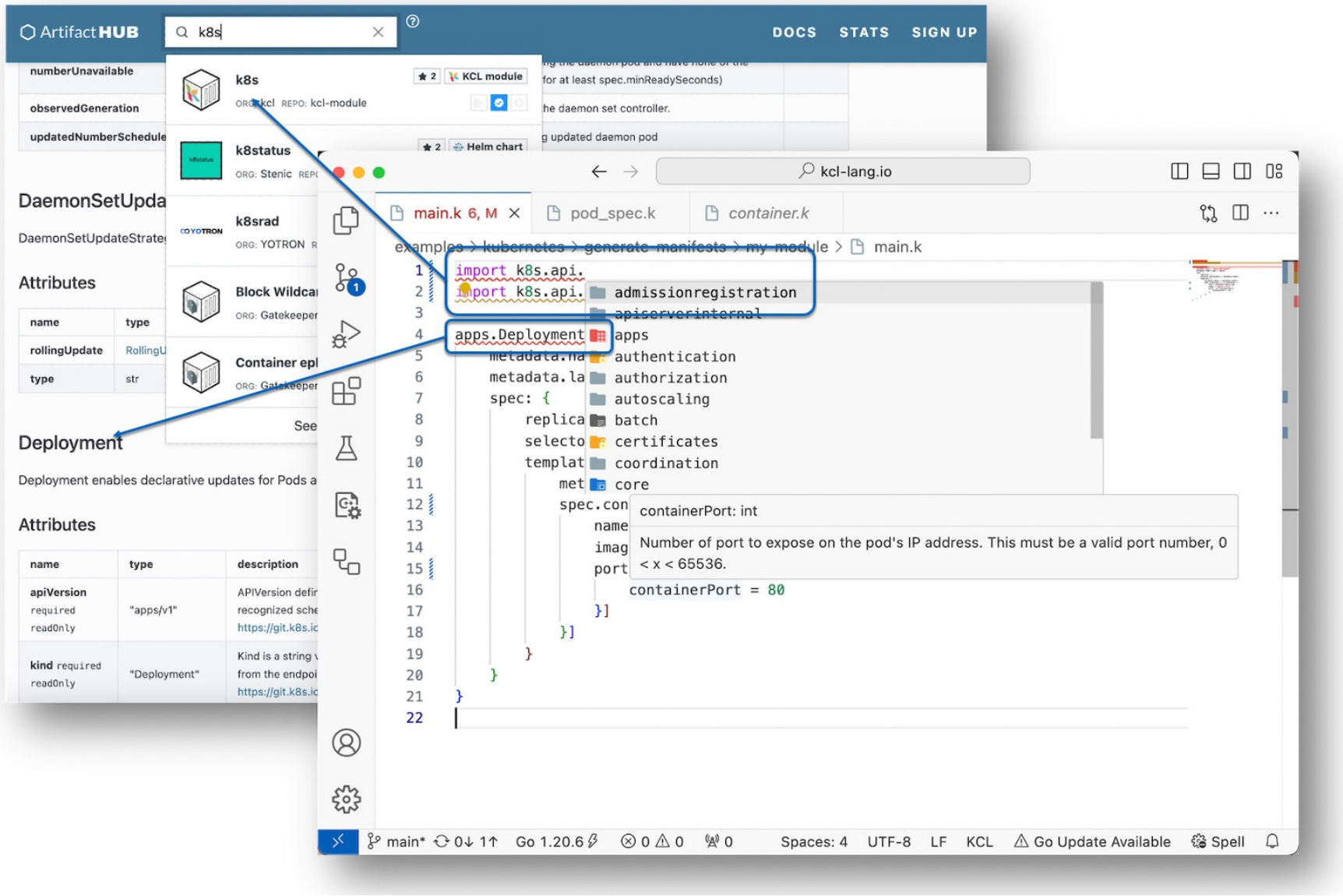
Language + Tools + IDEs + SDKs + Plugins



IDE


nginx.k > ...
import json
schema Nginx:
"""Schema for Nginx configuration files"""
http: Http
schema Http:
server: Server
schema Server:
listen: int | str # The attribute `listen` can be int type or a string
location?: Location # Optional, but must be non-empty when specified
schema Location:
root: str
index: str
schema Person:
name: str
age: int
x = Person{
name: "foo"
age:
}
nginx = Nginx {
http.server = {
listen = 80
location = {
root = "/var/www/html"
index = "index.html"
}
}
}
PROBLEMS 3
PORTS
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, **/*.ts, !**/nod
nginx.k 3
name 'Http' is not defined (CompileError) [Ln 5, Col 11]
expected one of ["identifier", "literal", "(", "[", "{"] got newline (InvalidSyntax) [Ln 24, Col 9]
Module 'json' imported but unused (UnusedImportWarning) [Ln 1, Col 1]
nginx.k
import json Module 'json' imported but unused
schema Nginx:
"""Schema for Nginx configuration files"""
http: Http name 'Http' is not defined
schema Http:
server: Server
schema Server:
listen: int | str # The attribute `listen` can be int type or a string type.
location?: Location # Optional, but must be non-empty when specified
schema Location:
root: str
index: str
schema Person:
name: str
age: int
# schema Foo:
x = Person{
name: "foo"
age: expected one of ["identifier", "literal", "(", "[", "{"] got newline
}
nginx = Nginx {
http.server = {
listen = 80
location = {
root = "/var/www/html"
index = "index.html"
}
}
}
configuration/nginx.k 3
name 'Http' is not defined (CompileError) [5, 11]
expected one of ["identifier", "literal", "(", "[", "{"] got newline (InvalidSyntax)
Module 'json' imported but unused (UnusedImportWarning) [1, 1]
main
kcl
17 2
2 1
import json
schema Nginx:
"""Schema for Nginx configuration files"""
http: Http
schema Http:
server: Server
schema Server:
listen: int | str # The attribute `listen` can be int type or a string type.
location?: Location # Optional, but must be non-empty when specified
schema Location:
root: str
index: str
schema Person:
name: str
age: int
x = Person{
name: "foo"
age: 1
}
nginx = Nginx {
http.server = {
listen = 80
location = {
root = "/var/www/html"
index = "index.html"
}
}
}

Modules

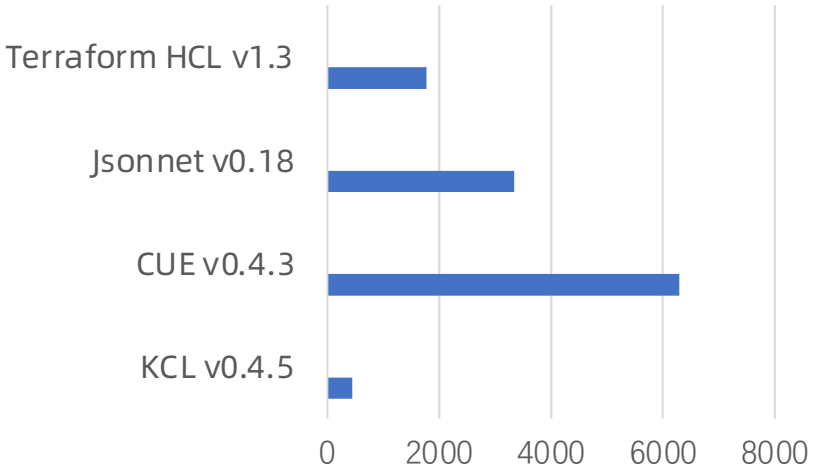


300+ out of the box modules

Performance

Loop and Function

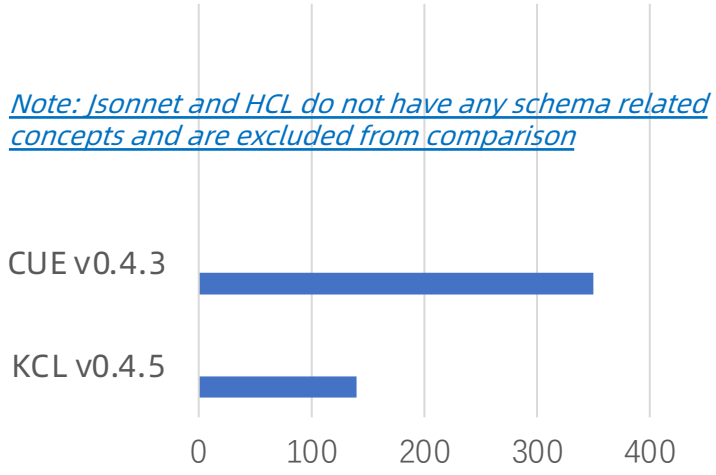
```
a = lambda x: int, y: int -> int {
    max([x, y])
}
temp = {"a${i}": a(1, 2) for i in range(10000)}
```



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)

Appendix

04

Resources

- **Website**
  - *https://kcl-lang.io/*
- **GitHub**
  - *https://github.com/kcl-lang/kcl*
- **Community**
  - *https://github.com/kcl-lang/community*
- **Twitter**
  - *@kcl language*
- **Slack**
  - *https://cloud-native.slack.com/archives/C05TC96NWN8*

Related 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

THANKS