为什么我们用 Rust 重写了自己的项目

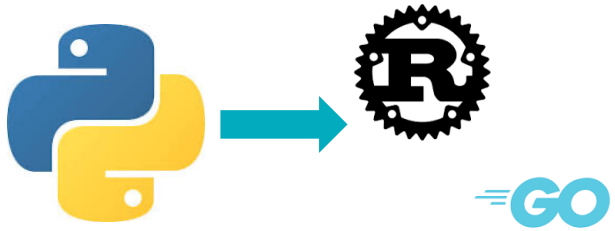
CNCF KCL Owner / 徐鹏飞

个人介绍

徐鹏飞 (GitHub: Peefy)，开源爱好者

对云原生、Serverless、区块链、AI 等领域感兴趣，目前主要维护下面几个开源项目

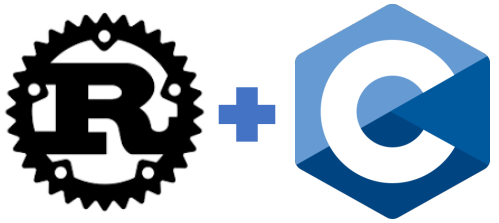
https://github.com/kcl-lang



https://github.com/pluto-lang

SIR

https://github.com/AntChainOpenLabs/Smart-Intermediate-Representation



提纲

01 重构 - 凤凰涅槃 🔥

02 演进 - 奔腾跃进 🐎

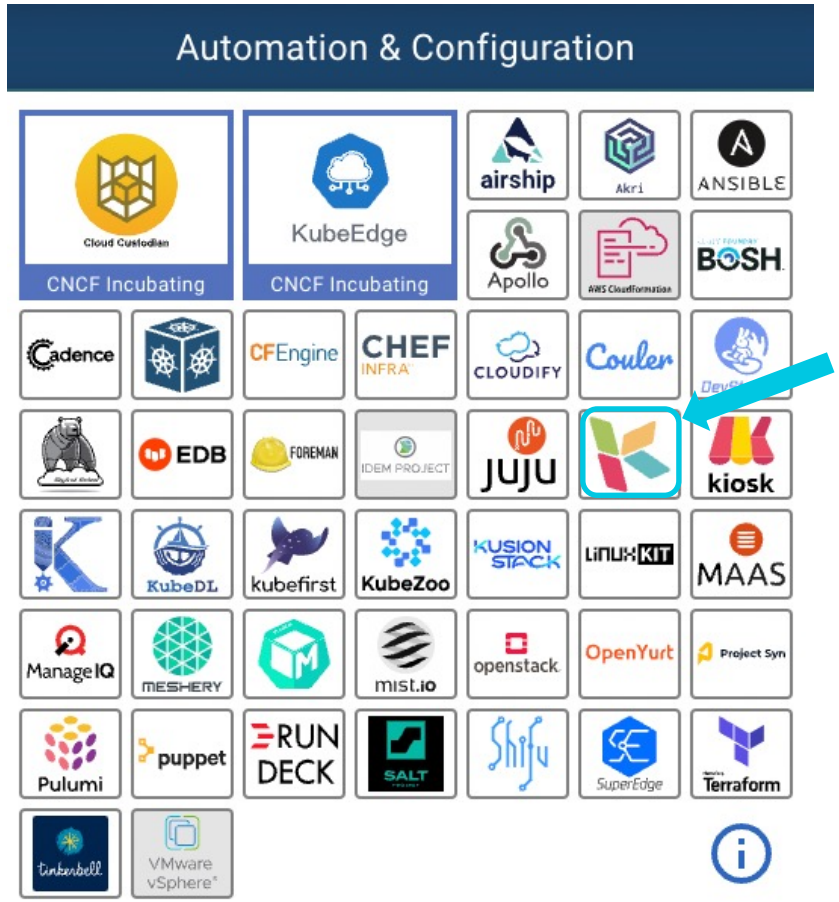
03 展望 - 星辰大海 🌌

01

重构 - 浴火重生 🔥

简介 (广告时间 😑)

KCL 专用配置策略语言 (2022.6 用 Rust 重写完成后正式开源，2023.9 成为 CNCF 基金会托管的 Sandbox 项目)



- ✓ **领域特定**：以收敛的语言和工具集合解决领域问题近乎无限的变化和复杂性，同时兼顾表达力和易用性
- ✓ **以数据和模型为中心**：开发者可以理解的声明式 Schema/配置/策略模型用于 AI 工程，云原生工程等场景
- ✓ **包含结构化定义和约束的核心数据结构 Schema**：为 AI 数据集或者云原生配置场景提供**原生的数据验证和转换**能力
- ✓ **可复用扩展**: OCI 等标准软件供应链集成和包管理工具支持，官方 Registry 提供 300+ 模型包
- ✓ **引擎解耦**：建立在一个完全开放的世界当中，几乎不与任何编排/引擎工具或者控制器绑定，可同时为客户端和运行时场景提供 API 抽象、组合和校验的能力
- ✓ **多语言 SDK**：**轻易集成到不同的业务场景和生态当中，目前提供了 Rust, Go, Python, Java 等 SDK**

项目地址：https://github.com/kcl-lang

当然，今天我们不聊 KCL，聊一聊和 Rust 那些事

重构

2022.1 - 2022.6 用 Rust 重写 KCL (资源投入 3 人)

问题

- 原有项目冷启动时间长，性能差
- Python None NPE 问题影响稳定性，虽然测试覆盖率一度高达 90%+
- 中型的 Python 项目维护成本，类型注解，mypy 等工具使用繁琐

50k Python Code

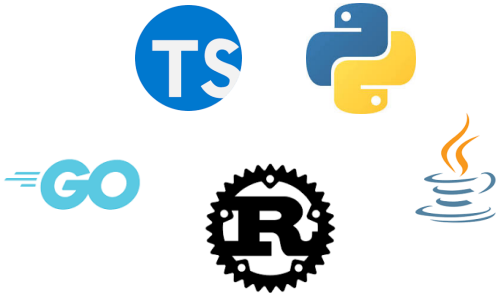
https://github.com/kcl-lang/kcl-py

2022.1

50k Rust Code
For KCL Core

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

2022.6



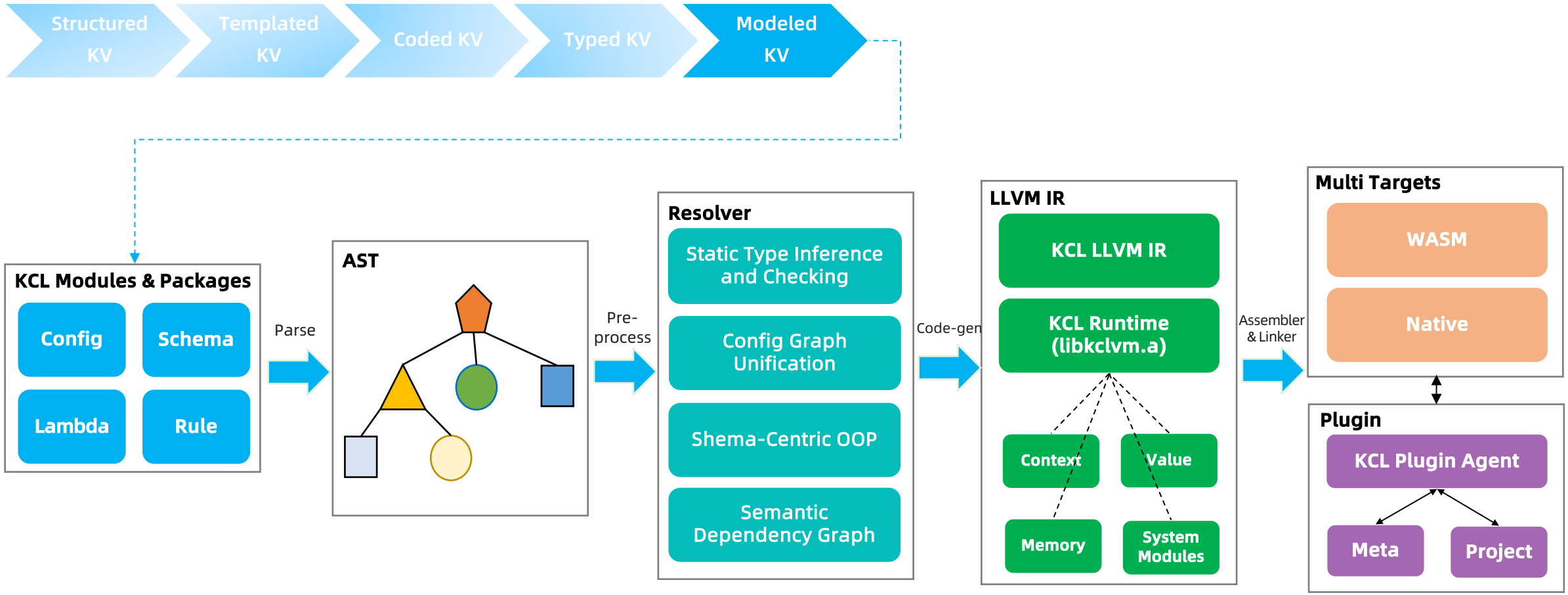
80k Rust Code
For KCL Core

Rust 核心+其他语言外围生态项目

至今

《我们用Rust重写了我们的项目……速度快了40倍》https://juejin.cn/post/7207263350488252453#comment

架构



分模块重写，从运行时到编译器前端

结果

综合考虑用户体验、稳定性和性能决定重写，各维度指标均有提升

66%

E2E Perf

20x

Parser

40x

Resolver

0

NPE Bugs

1/40

Cold Start

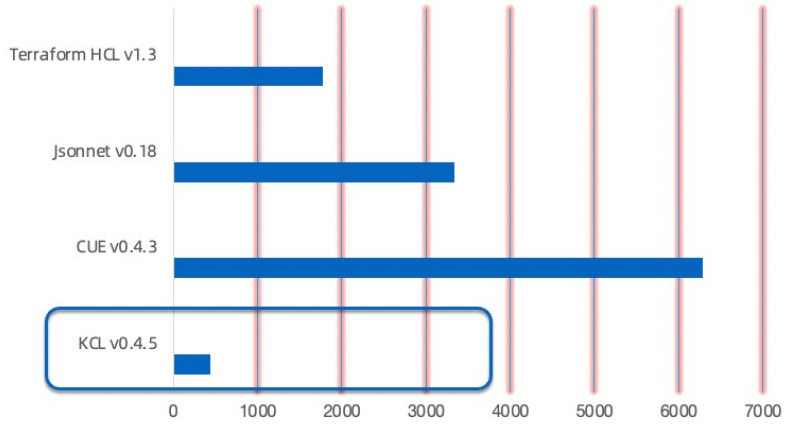
1/2

Runtime Memory

结果

Loop and Function

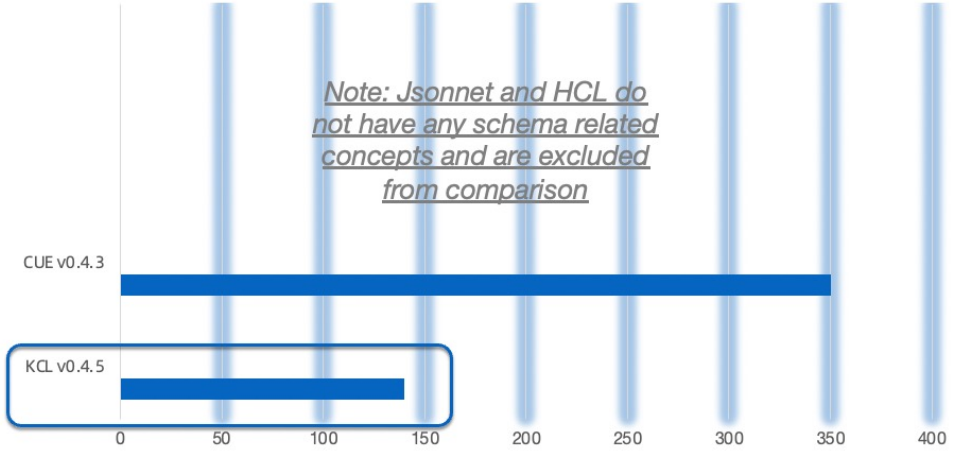
```
a = lambda x: int, y: int -> int {
    max([x, y])
}
temp = {"a${i}": a(1, 2) for i in range(10000)}
```



Kubernetes Configuration

```
import kubernetes.api.apps.v1

deployment = v1.Deployment {}
```

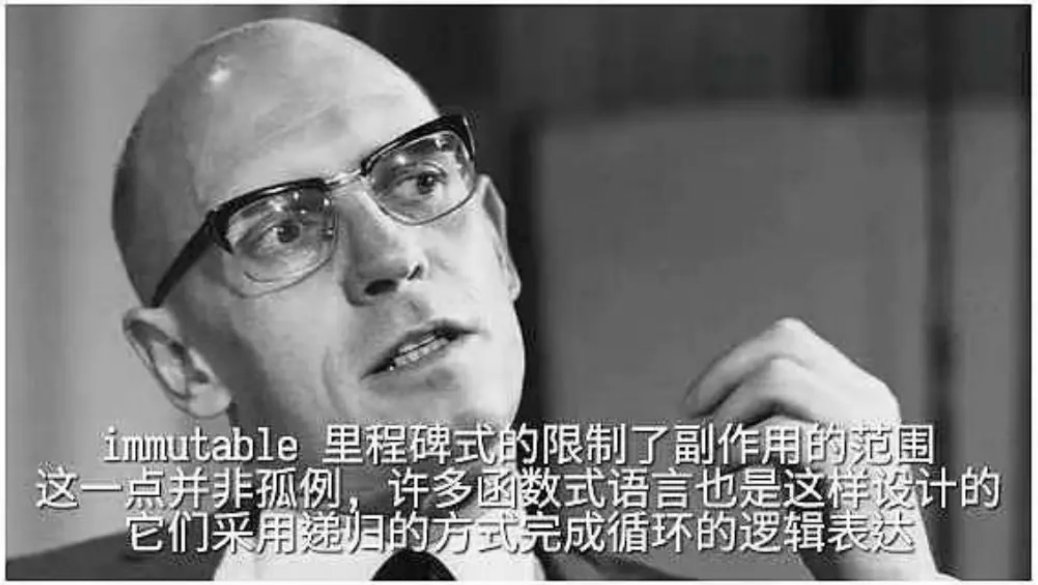


Test environment: single core macOS 10.15.7 CPU: i7-8850H 2.6GHz 32GB 2400Mhz DDR4 No NUMA, e2e run time (ms)

性能超越社区同类型项目

为什么选择 Rust

我从前跟别人谈论Rust



我现在跟别人谈论Rust

为什么选择 Rust

Stack Machine 虚拟机实现大比拼 (循环指令进行压测，最终 Rust 胜出)

StackMachine / src / main.py

Code Blame 313 lines (287 loc) · 7.82 KB Code 55% faste

```python
@dataclass
class VM:
    stack:   List[KCLObject]
    globals: Dict[str, KCLObject]
    locals:  Dict[str, KCLObject]

    def push(self, obj: KCLObject):
        self.stack.append(obj)

    def pop(self) -> KCLObject:
        return self.stack.pop()

    def peek_nth(self, i: int) -> KCLObject:
        return self.stack[len(self.stack)-i]

    def peek(self) -> KCLObject:
        return self.stack[len(self.stack)-1]

    def run(self, bytecode: ByteCode):
        i = 0
        while i < len(bytecode.instructions):
            elem = bytecode.instructions[i]
            if elem.opcode == BinaryAdd:
                pass
            elif elem.opcode == StoreGlobal:
                # print("StoreGlobal")
                name = bytecode.names[elem.operand3]
                self.globals[name] = self.pop()
```

StackMachine / src / main.rs

Code Blame 382 lines (382 loc) · 11.3 KB Code 55% faster with GitHub Copilot

```rust
struct VM {}
impl VM {
    pub fn run(self, bytecode: &ByteCode) {
        println!("Run KCLVM Rust!");
        let mut stack: Vec<KCLObjectRef> = vec![];
        let mut globals: HashMap<String, KCLObjectRef> = HashMap::new();
        let mut locals: HashMap<String, KCLObjectRef> = HashMap::new();
        let mut i: usize = 0;
        let mut j: usize = 0;
        while i < bytecode.instructions.len() {
            let elem = &bytecode.instructions[i];
            match elem.opcode {
                Opcode::BinaryAdd => {
                    println!("BinaryAdd");
                }
                Opcode::StoreGlobal => {
                    // println!("StoreGlobal");
                    let index = elem.operand3 as usize;
                    let name = bytecode.names[index].clone();
                    let obj = stack.pop().expect("empty stack");
                    globals.insert(name, obj);
                }
                Opcode::StoreLocal => {
                    // println!("StoreLocal");
                    let index = elem.operand3 as usize;
                    let name = bytecode.names[index].clone();
                    let obj = stack.last().expect("last object not found");
                    locals.insert(name, (*obj).clone());
                }
```

StackMachine / src / main.go

Code Blame 352 lines (345 loc) · 5.99 KB Code 55% faster w

```go
type ByteCode struct {
type VM struct {
        stack   []KCLObject
        globals map[string]KCLObject
        locals  map[string]KCLObject
}

func (vm *VM) push(obj KCLObject) {
        vm.stack = append(vm.stack, obj)
}
func (vm *VM) pop() KCLObject {
        last := vm.stack[len(vm.stack)-1]
        vm.stack = vm.stack[:len(vm.stack)-1]
        return last
}
func (vm *VM) peek_nth(i int) KCLObject {
        return vm.stack[len(vm.stack)-i]
}
func (vm *VM) peek() KCLObject {
        return vm.stack[len(vm.stack)-1]
}
func (vm *VM) run(bytecode *ByteCode) {
        i := 0
        for i < len(bytecode.instructions) {
                elem := bytecode.instructions[i]
                switch elem.opcode {
                case BinaryAdd:
                        {
                        }
                case StoreGlobal:
```

https://github.com/Peefy/StackMachine

为什么选择 Rust

- JS/TS/Python Infra -> Rust: 越来越多的编程语言的编译器或运行时特别是**前端基础设施以及 Python 基础设施**项目采用 Rust 编写或重构，此外基础设施层，数据库、搜索引擎、网络设施、云原生、UI 层和嵌入式等领域都有 Rust 的出现，至少在编程语言领域实现方面经过了可行性和稳定性验证

- 业务考量：考虑到后续的项目 **SmartIR 涉及区块链和智能合约**方向，而社区中大量的区块链和智能合约项目采用 Rust 编写

- 多语言集成：通过 Rust 获得更好的性能和稳定性，让系统更容易维护、更加健壮的同时，可以通过 FFI 暴露 C API 供多语言使用和扩展，**方便生态扩展与集成**

- WASM 友好：社区中大量 WASM 生态是由 Rust 构建，KCL 语言和编译器可以借助 Rust 编译到 WASM 并在浏览器中运行

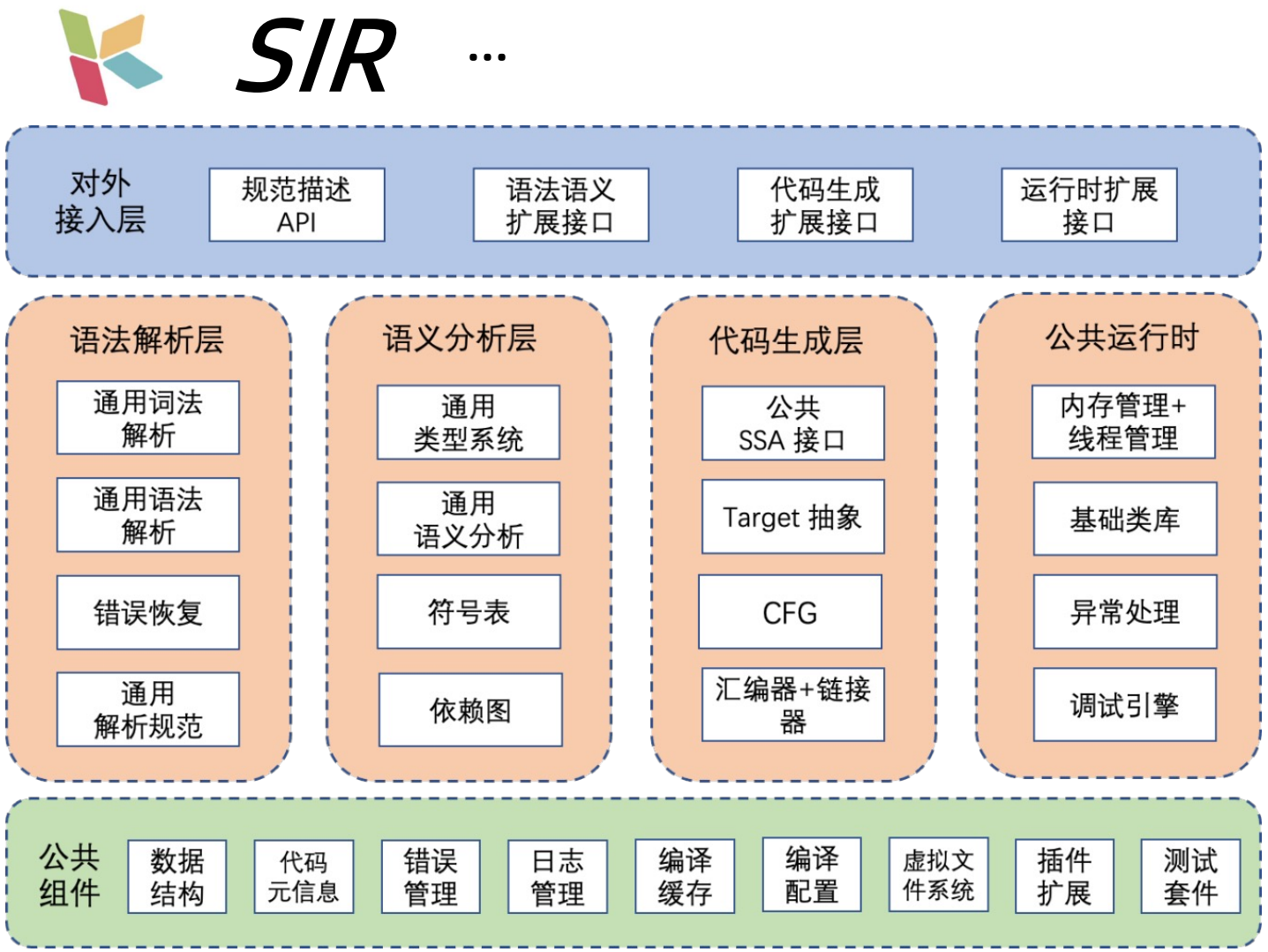
遇到了哪些困难

- **飞机上换引擎**：边重写，边测试，边生产落地，一共 2000+ 的单元测试需要通过
- **心智转换**：Rust 某些独有特性的学习成本：函数式，所有权，生命周期，异步等，不过入门了写多了就好了
- **Unsafe**: 对待 Unsafe 代码需要小心翼翼，一不小心就跑飞
- **循环引用之殇**：KCL 语言很多数据结构都是图这种，并且语义允许变量循环引用并在 Runtime 进行垃圾回收，比如 **Rc<RefCell<KCLValue>>** 和 **Arc<Mutex<KCLValue>>** 难以设计和编写，不像别的带 GC 的语言直接写就完事了

02

演进 - 奔腾跃进 

CompilerBase



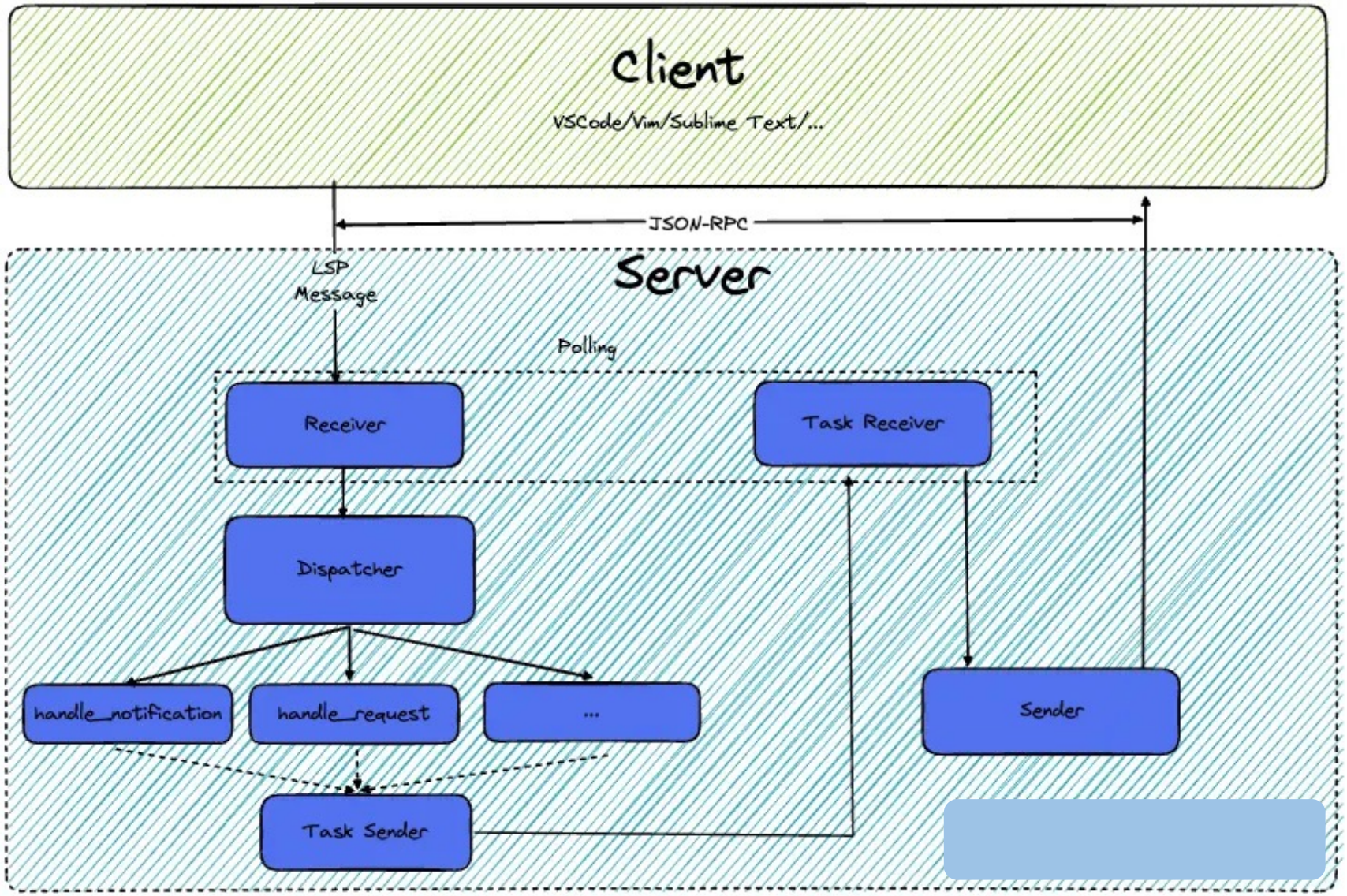
问题

- 前端的一些 Rust 基础设施如 SWC 等大量参考了 Rustc
- Rustc 项目中近乎是 Rust 的最佳实践，有些依赖太重，有些是非 Stable 的

解法

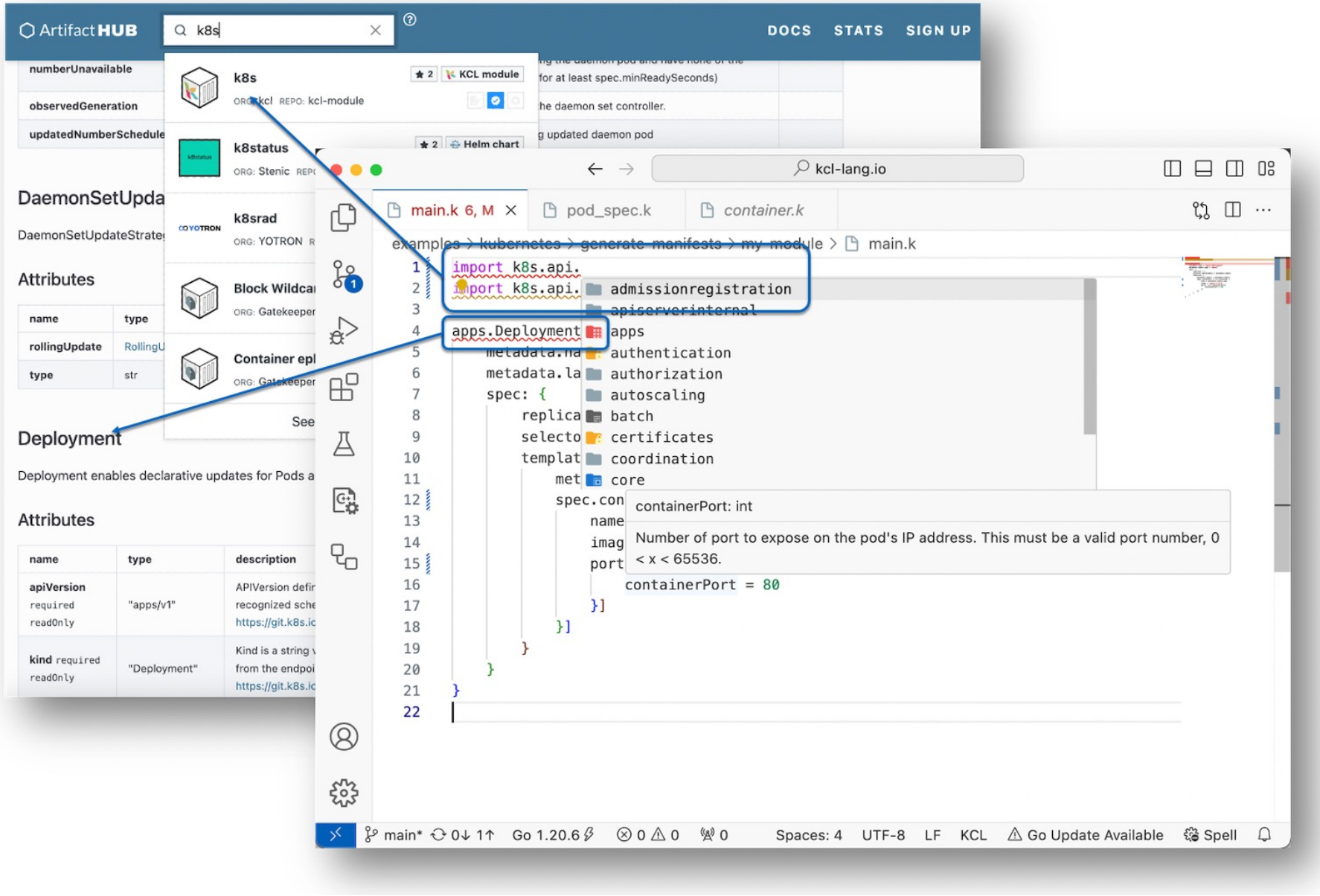
- 基于 Rustc Crates 和 Rustc 官方周边项目如错误处理、核心数据结构和算法等构建了一套编译器基础设施

Language Server



- 基于 Rust 性能依旧是几十倍几十倍的提升，用户体验有明显改善
- Language Server 借助 **Rust** 编译为 WASM，让 KCL 直接跑在浏览器中

IDE & Registry Based Rust LSP



多语言 SDK

libkcl.dylib/so/dll

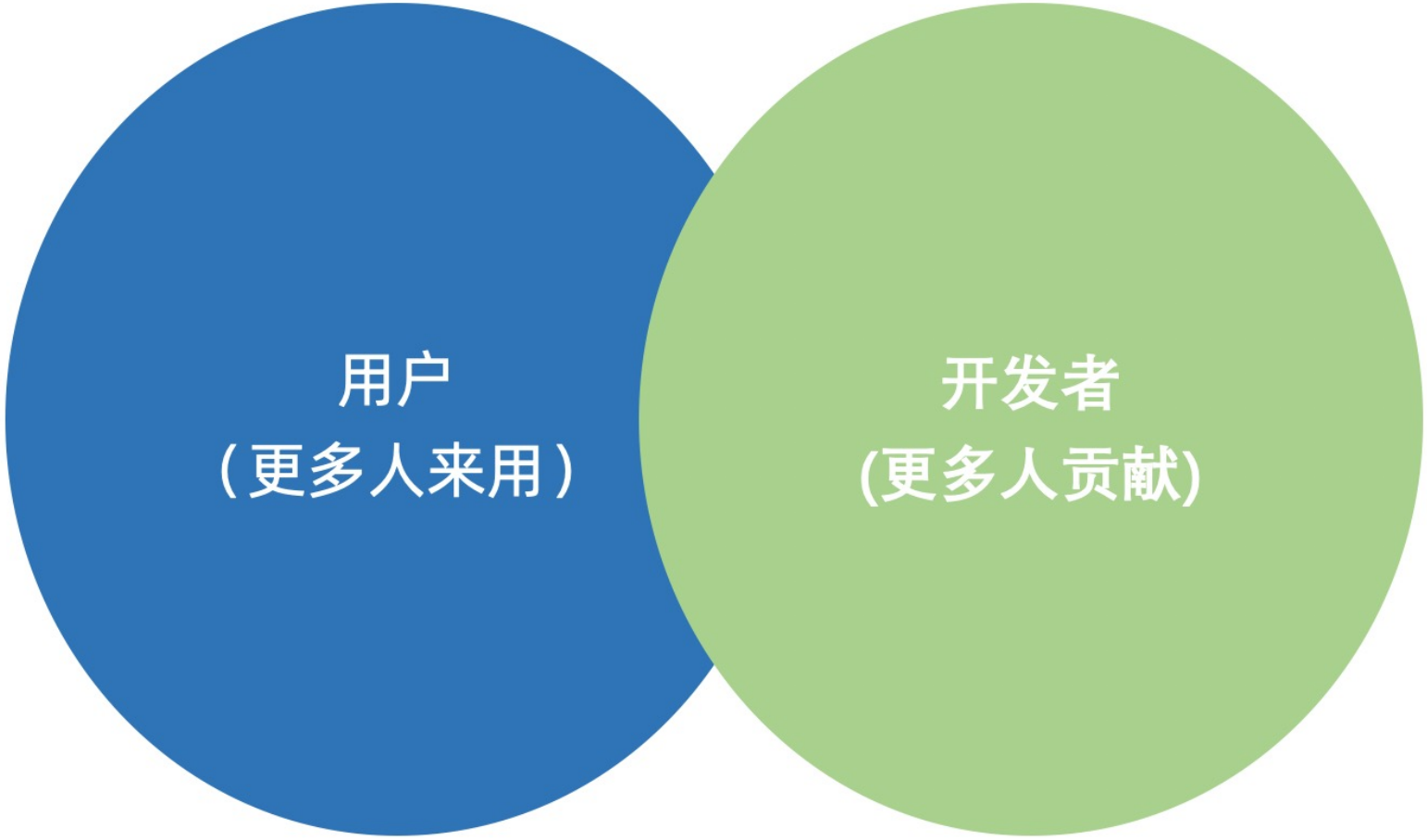
KCL Rust Core

- Rust Core 编译为动态链接库，通过多语言的包管理工具将 Rust Core 打包到不同语言 SDK 中。
- 通过 Protobuf Spec 定义多语言 API, 代码自动生成，省时省力
- Rust 多语言构建工具链生态成熟，用 Rust 开发一个 Python 库可能比 Python 开发一个 Python 库还敏捷
- 多语言 SDK 用于不同场景的同时也赋予我们相比于其他社区产品竞争优势

03

展望 - 星辰大海 

2024



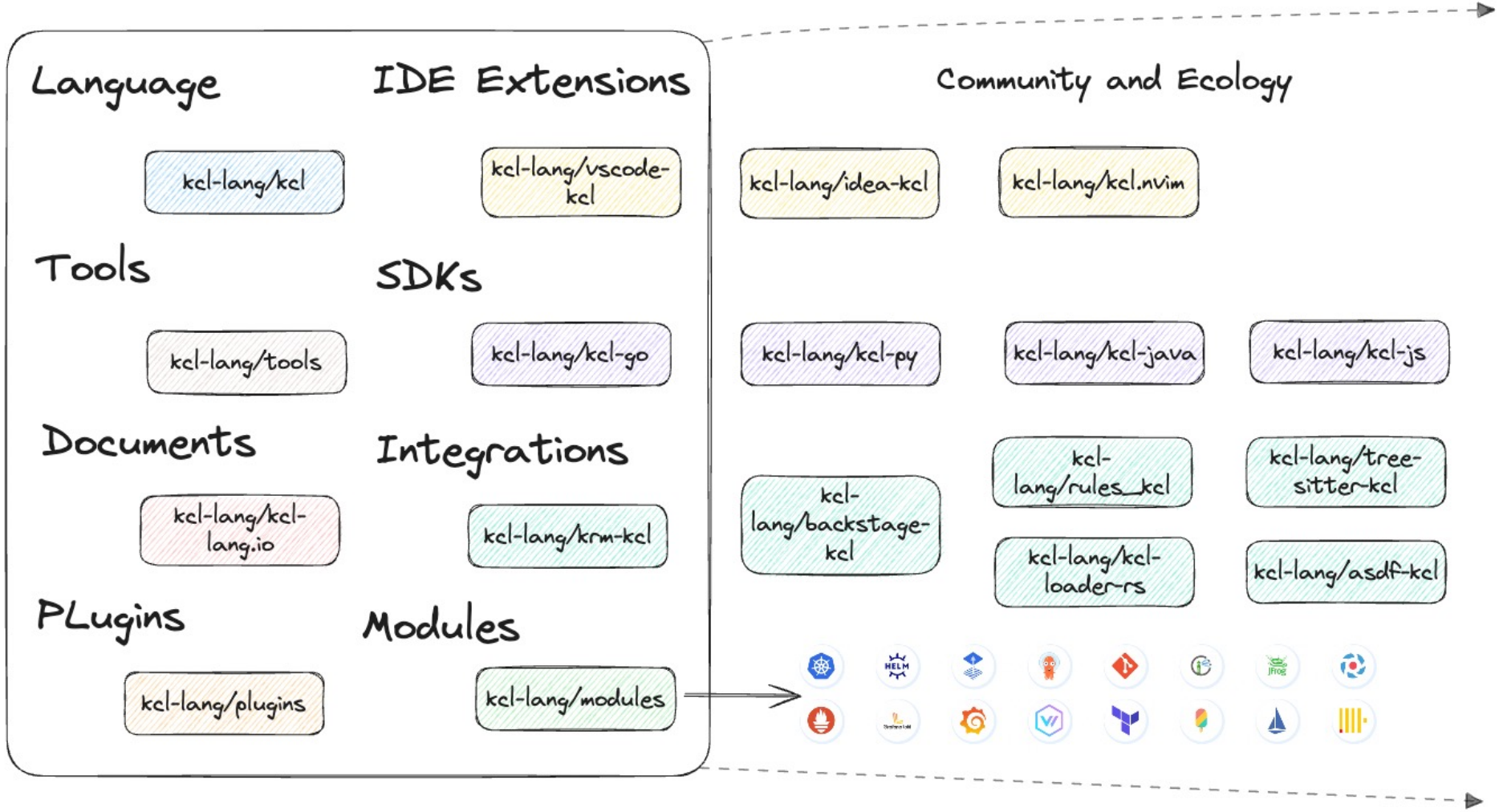
Community over Code

面向开源，我们坚持体验和生态优先，Rust 很好地满足了诉求，希望有越来越多的开发者参与进来

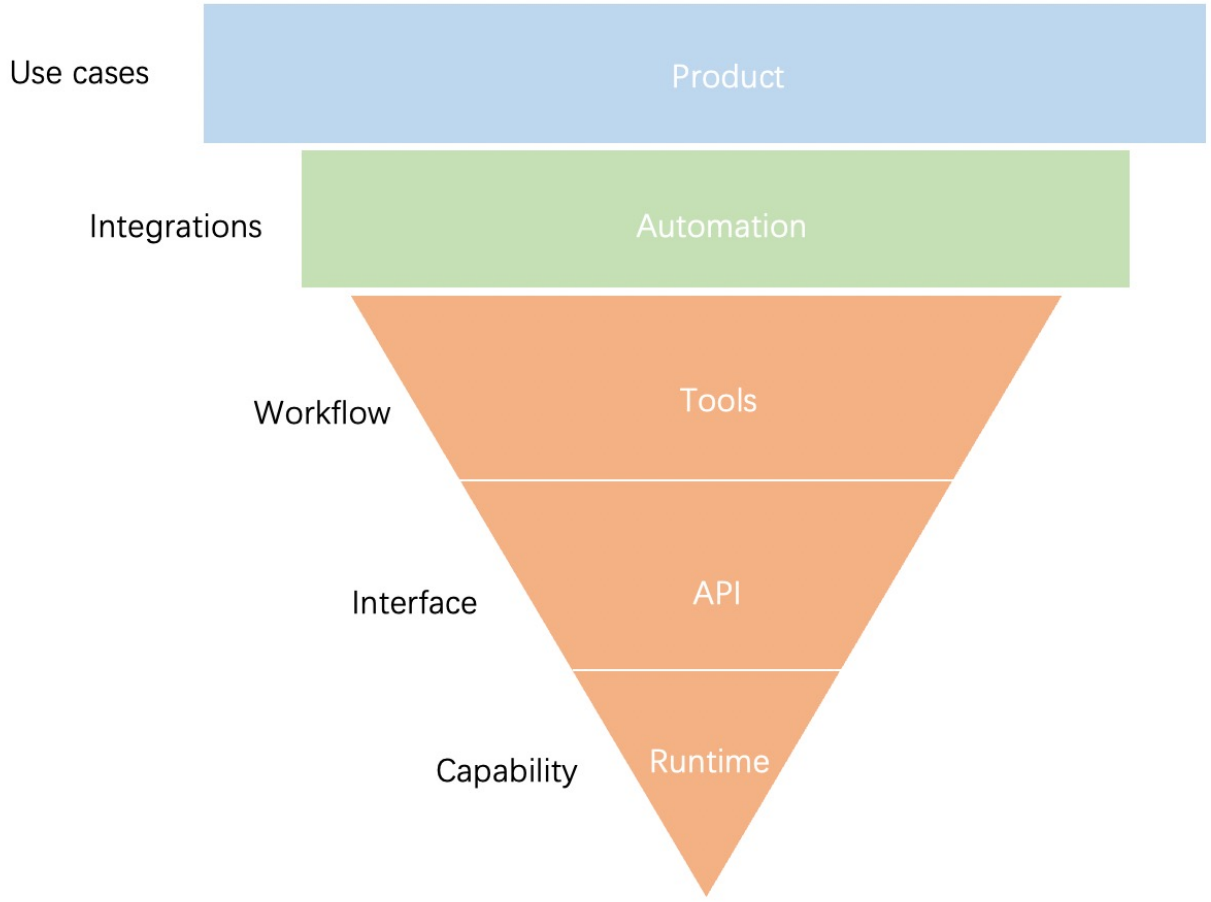
https://github.com/kcl-lang/kcl/issues/882

附录

项目组织


Language
kcl-lang/kcl
IDE Extensions
kcl-lang/vscode-kcl
Tools
kcl-lang/tools
SDKs
kcl-lang/kcl-go
Documents
kcl-lang/kcl-lang.io
Integrations
kcl-lang/krm-kcl
PLugins
kcl-lang/plugins
Modules
kcl-lang/modules
Community and Ecology
kcl-lang/idea-kcl
kcl-lang/kcl.nvim
kcl-lang/kcl-py
kcl-lang/kcl-java
kcl-lang/kcl-js
kcl-lang/backstage-kcl
kcl-lang/rules_kcl
kcl-lang/tree-sitter-kcl
kcl-lang/kcl-loader-rs
kcl-lang/asdf-kcl

模块组织



IaC & PaC Based KCL
GitOps & GitClickOps Based KCL
Dynamic Configuration & Package Management for Clouds and Kubernetes

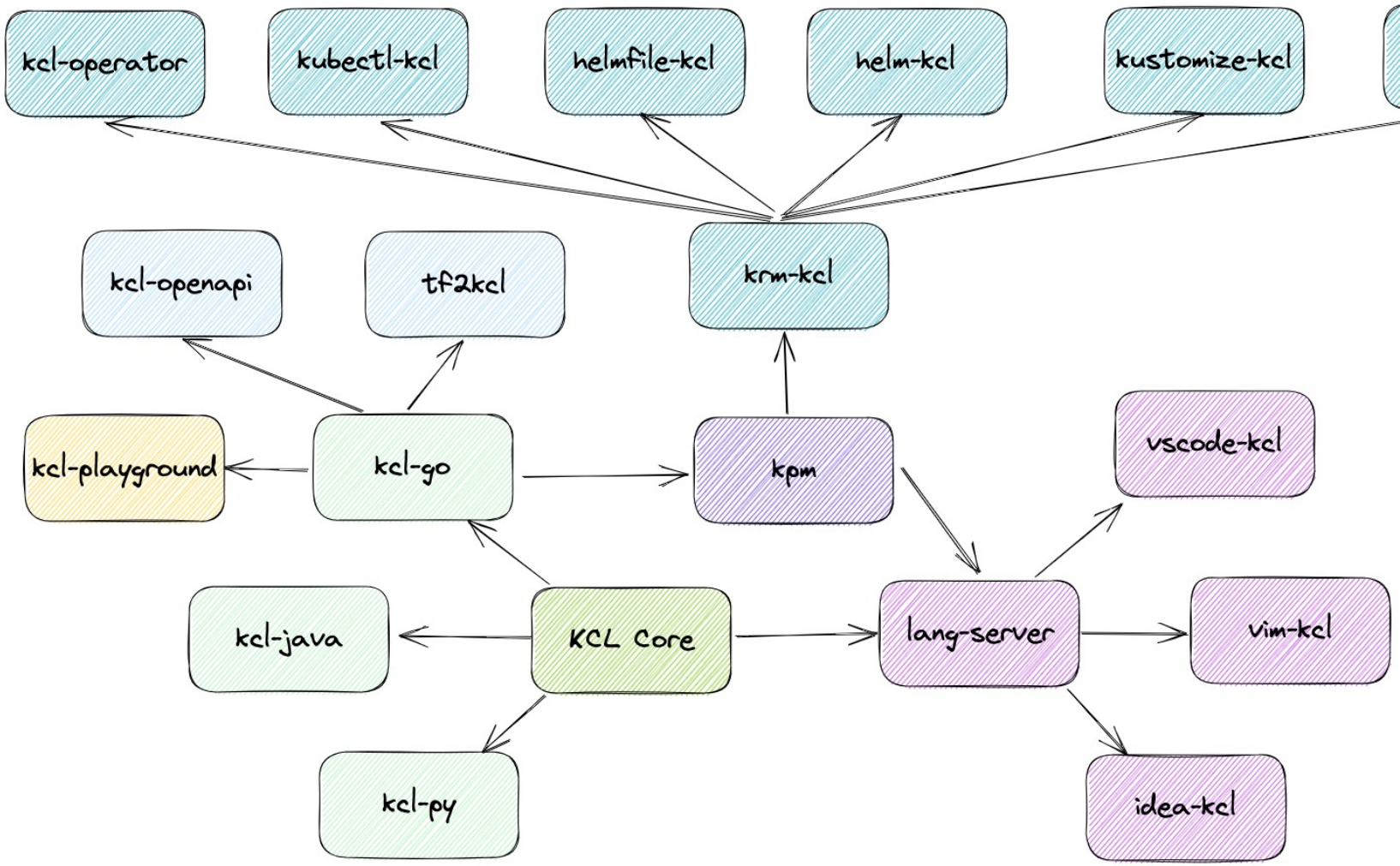
Workflow Integrations (CI & CD Tools & DevOps Tools): KusionStack,
API Composition Tool Integrations: Crossplane, Kustomize, Helm, ···
Hub & Registry Integration: Artifact Hub, Docker Hub, Harbor
Data & Schema Integration: Open API, Terraform, Kubernetes
Config UI: Backstage Plugin, Form Component Integrations

Tooling: run, mod, registry, vet, plugin, deps, doc, fmt, import, export, test, debug, fix, lint
IDEs: UI Extension & Client, Language Server: Definition, Reference, Diagnostic, Completion, ···
SDKs: Rust, Go, Python, Java, ···

API: API Server, Lang APIs, Plugin APIs, Tooling APIs, LSP APIs
Lang UI: Syntax, AST, Data Types.
Features: query, override, entry, top level arguments, error handling, ···
Plugin: Multilingual or WASM function plugins

Sematic: Type system, sematic rules.
Performance: Parse, Resolve, Compile, Evaluation
Cross-platform: macOS, Linux, Windows, WASM
Runtime Capabilities: IR, Profiling, Debuging, System Modules, Memory management, Exception handling
Compiler Data Structure and Algorithms

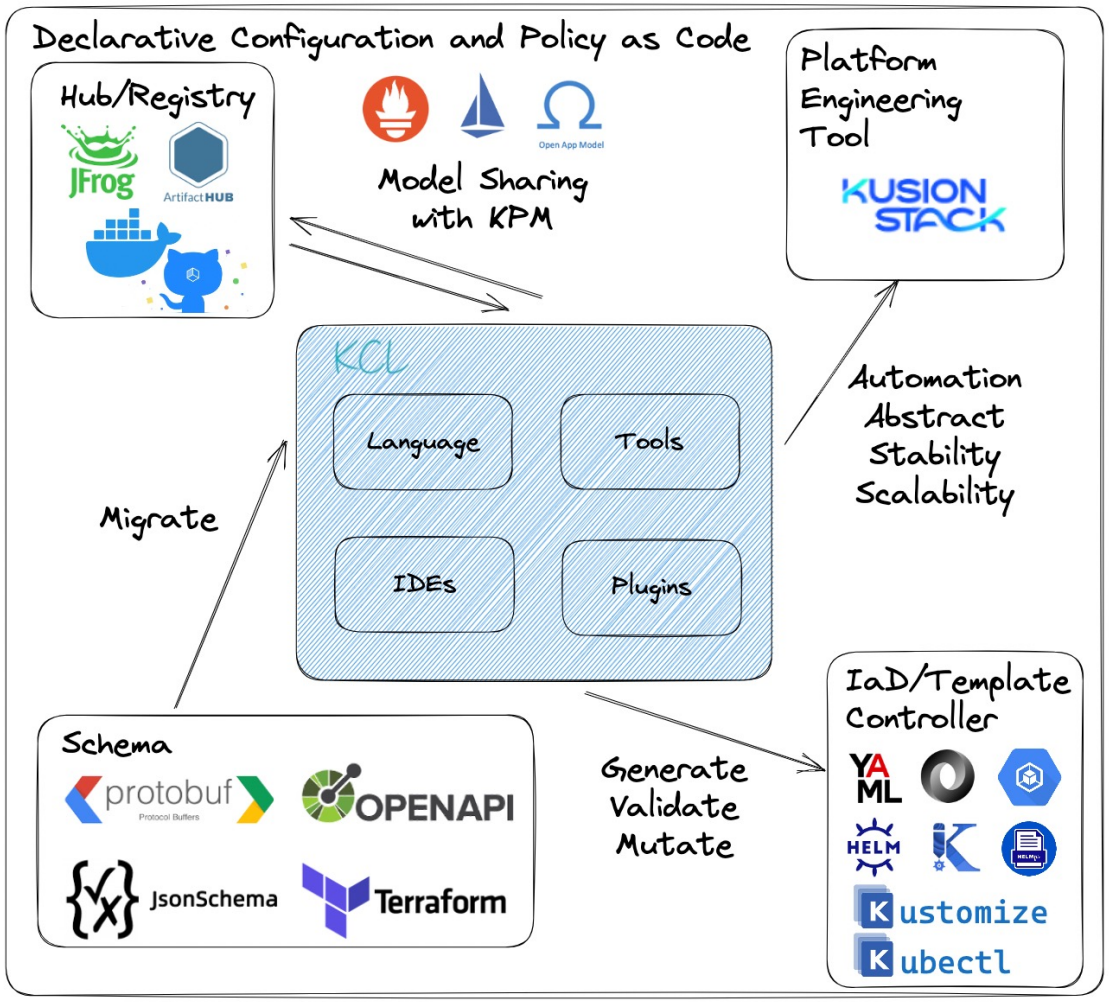
组件依赖关系



How to (For App/Platform dev & SRE)

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

生态集成



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

社区产品和定位

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