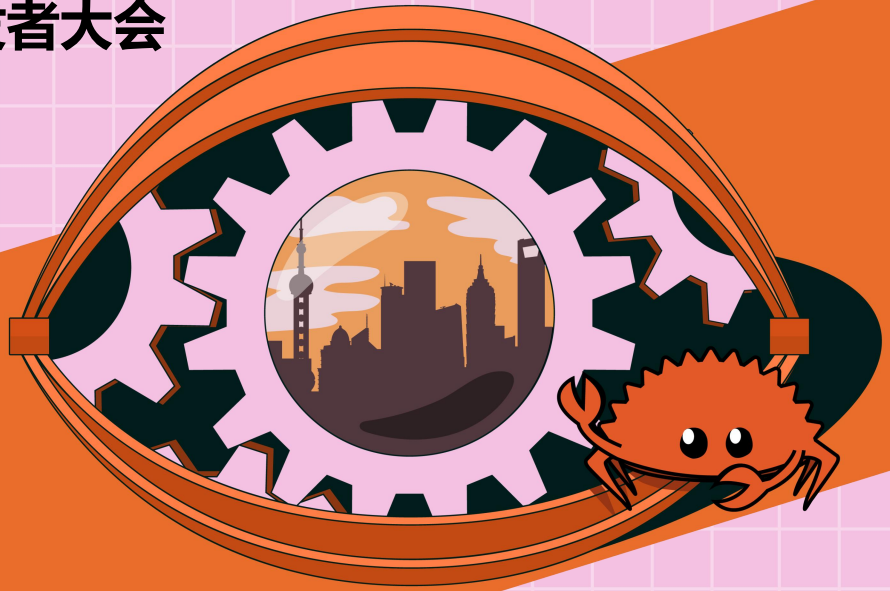
RUST CHINA CONF 2023

第三届中国Rust开发者大会



6.17-6.18 @Shanghai

KCL: Rust 在编译器领域的实践与探索

张正
蚂蚁集团

Agenda

01 KusionStack 与 KCL

02 用 Rust 重写 KCL

03 Rust 重写后的收益

04 更多的探索

01
KusionStack 与 KCL

KusionStack是什么?

KusionStack架构

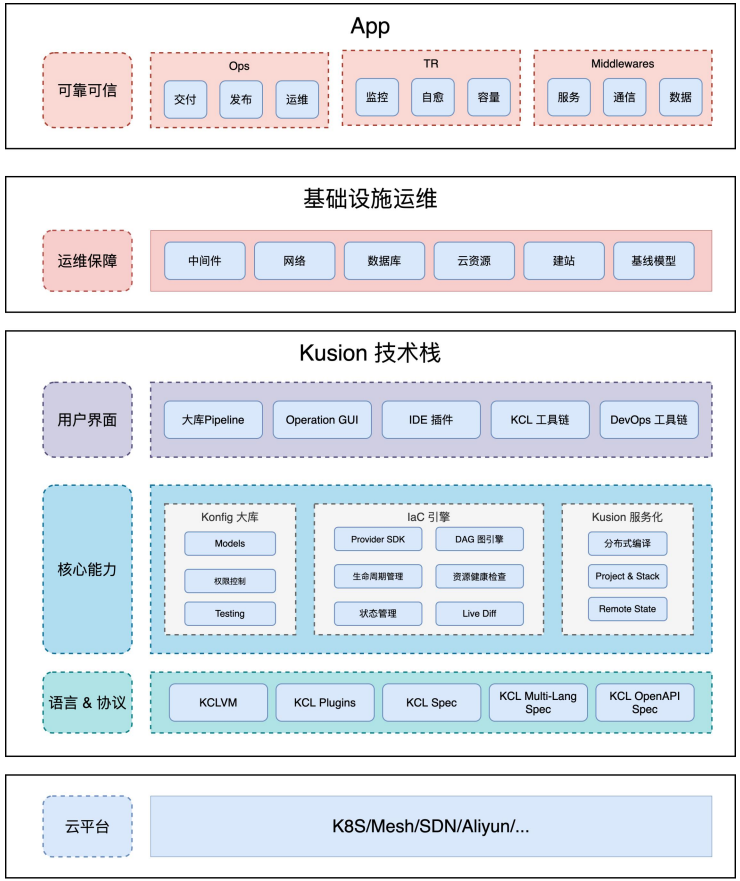
KCL

KusionStack 是什么?

KusionStack 是开源的云原生可编程技术栈!

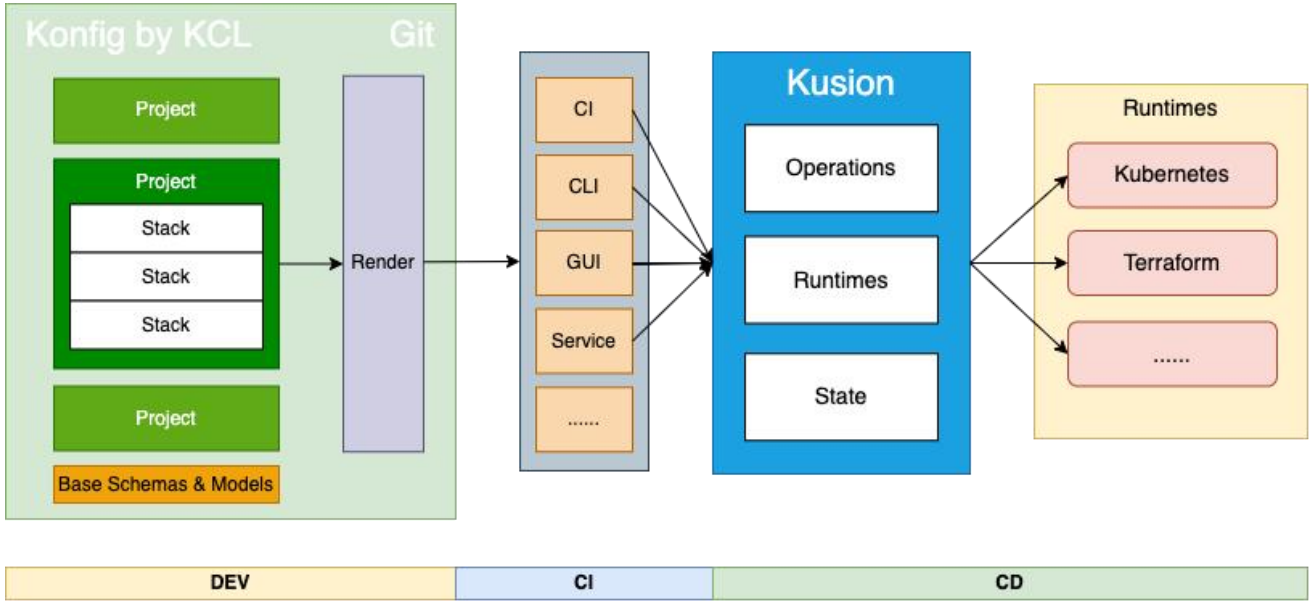
1. 围绕现代应用程序交付以及使用 OCI 镜像对配置和策略进行**编码**和**统一**
2. **组织**应用程序资源，并在整个交付过程中通过**身份**确保安全
3.为 **Kubernetes** 和**云**精简应用交付工作流，并提供开发友好的体验

基于 Platform as Code （平台服务即代码）理念，研发者可以用**统一的组织和操作界面**定义应用交付生命周期，充分利用**Kubernetes和云的混合能力**，通过**端到端的交付工作流程**，真正实现**集中定义、随处交付。**



KusionStack 架构

- **KCL**：面向应用研发者的配置策略专用高级编程语言，及其协议组，工具链及 IDE 插件
- **Kusion**：运维引擎、工具链、服务层，IDE 工作空间及社区技术集成套件
- **Konfig**：应用配置及基础模型共享仓库，及面向 GitOps 工作流程（如 GitHub Actions）的自定义 CI 套件



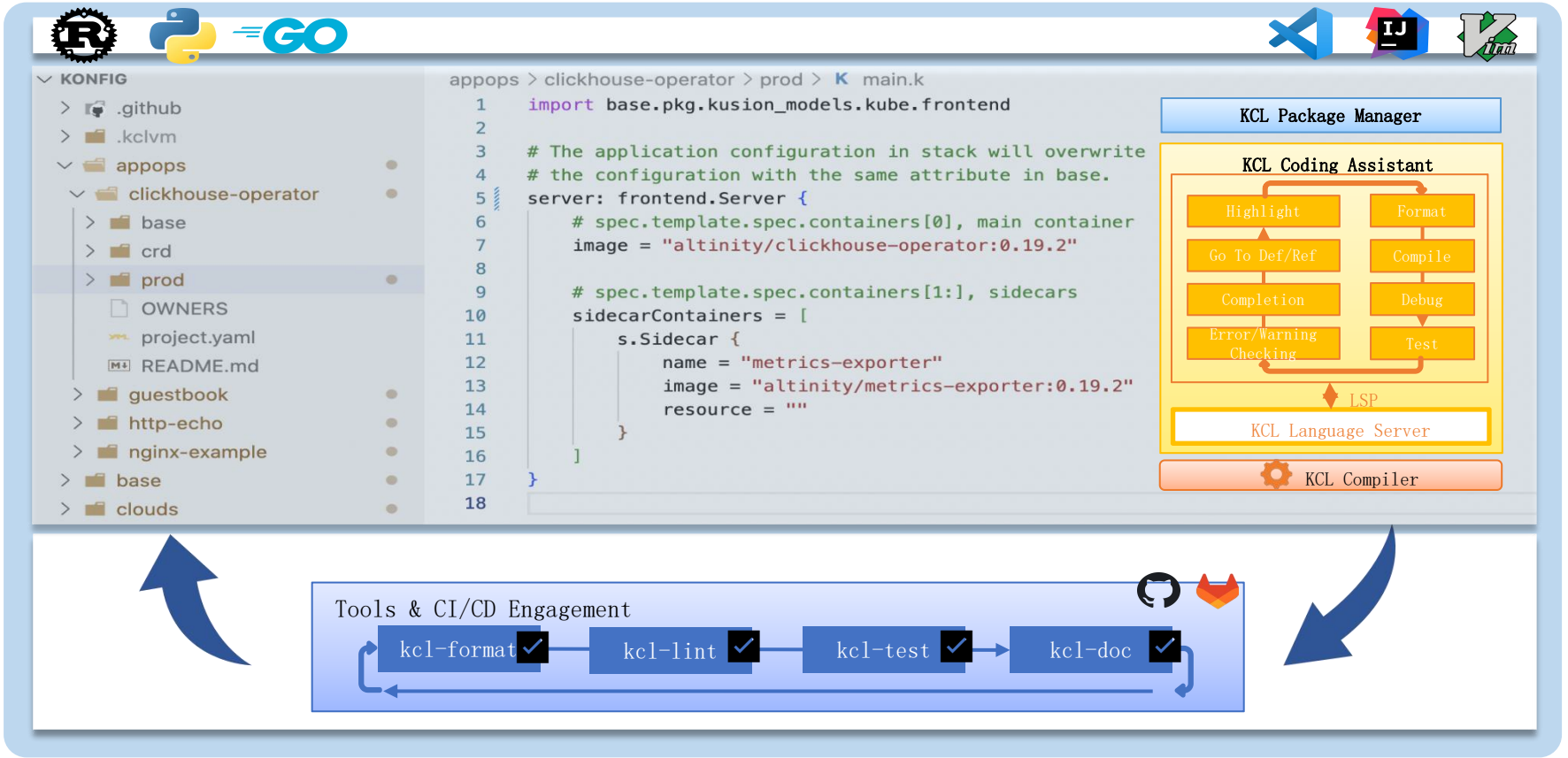
KusionStack 实践

1K/day	10K+/day	1 : 9	100K+
Pipelines	KCL Compilations	Plat : Dev	Commits
600+	5.7K+	1.2M+	10M+
Contributors	Projects	KCL Codes	YAML

Adopted by

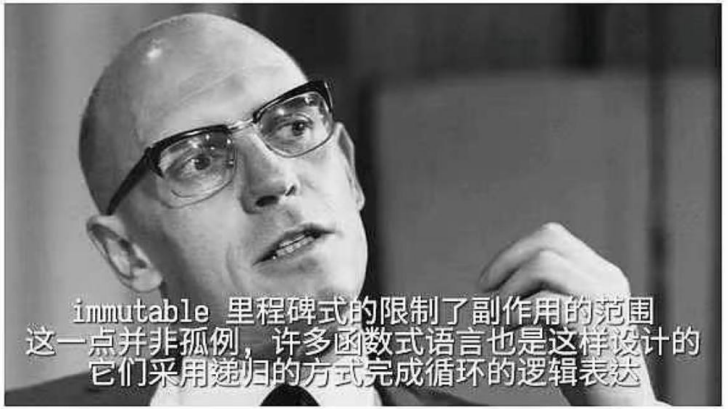
HUAWEI KYLIGENCE® 有赞

KCL


KONFIG
.github
.kclvm
appops
clickhouse-operator
base
crd
prod
OWNERS
project.yaml
README.md
guestbook
http-echo
nginx-example
base
clouds
appops > clickhouse-operator > prod > K main.k
1 import base.pkg.kusion_models.kube.frontend
2
3 # The application configuration in stack will overwrite
4 # the configuration with the same attribute in base.
5 server: frontend.Server {
6 # spec.template.spec.containers[0], main container
7 image = "altinity/clickhouse-operator:0.19.2"
8
9 # spec.template.spec.containers[1:], sidecars
10 sidecarContainers = [
11 s.Sidecar {
12 name = "metrics-exporter"
13 image = "altinity/metrics-exporter:0.19.2"
14 resource = ""
15 }
16]
17 }
18
KCL Package Manager
KCL Coding Assistant
Highlight
Format
Go To Def/Ref
Compile
Completion
Debug
Error/Warning Checking
Test
LSP
KCL Language Server
KCL Compiler
Tools & CI/CD Engagement
kcl-format
kcl-lint
kcl-test
kcl-doc

02
用 Rust 重写 KCL

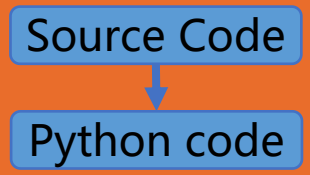
我从前跟别人谈论Rust



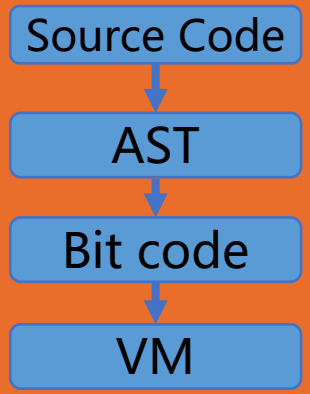
我现在跟别人谈论Rust

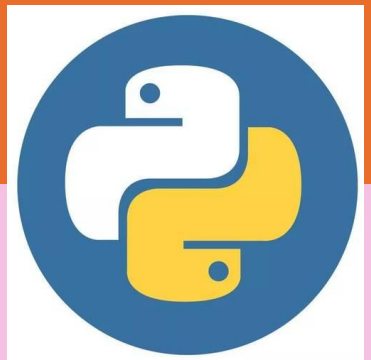
KCL 编译器架构升级

Python 代码翻译

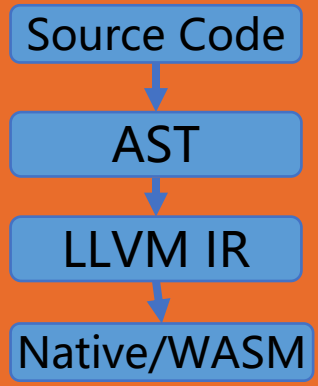


栈式虚拟机

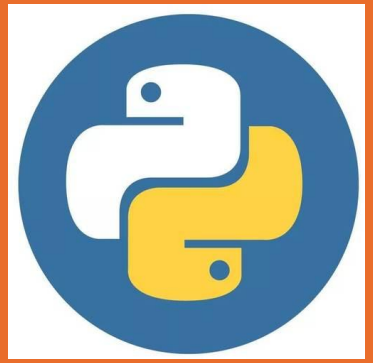




Rust 编译器



我们遇到了哪些问题?



Pros

简单易上手

生态丰富

研发效率高

Cons

性能问题

无法满足自动化系统需求

稳定性问题

None 空对象，属性不存在等运行时错误

为什么选择 Rust?

1. Go, Python, Rust 性能对比

	CPython	RustPython	GPython	VM(go)	VM(Rust)	VM(Python)	LLVM Native Code
简单Case a = 1	0.05s	0.125s	0.012s	0.01s	0.008s	0.6s	0.001s
for b = [i for i in rang(20000)]	0.065s	0.17s	0.055s	0.036s	0.011s	1s	0.01s
for(200000) b = [i for i in rang(200000)]	0.063s	0.187s	0.055s	0.044s	0.040s	1.99s	N/A
for b = [i for i in rang(2000000)]	0.177s	0.739s	0.304s	0.282s	0.283s	13.32s	N/A

> https://github.com/Peefy/StackMachine

为什么选择 Rust?

2. 越来越多的基础设施选择 Rust

3. 更好的性能和稳定性

4. 通过 FFI 暴露 C API 供多语言使用和扩展、方便集成

5. WASM 支持友好

6. 智能合约语言?

03
重写的收益

稳定性和性能提升

IDE：用户体验提升

稳定性和性能的巨大提升

01 **稳定性提升**

源于 Rust 强大的编译检查和错误处理方式， 更少的 Bug

02 **66 %**

端到端编译执行性能提升了 66%

03 **20 & 40**

前端解析器性能提升 20倍
中端语义分析器性能提升40倍

04 **50 %**

语言编译器编译过程平均内存使用量变为原来 Python 版本的一半

Case1: 单文件编译

> https://github.com/KusionStack/kcl#showcase

```
apiVersion = "apps/v1"
kind = "Deployment"
metadata = {
    name = "nginx"
    labels.app = "nginx"
}
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 }]
        }
    ]
}
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

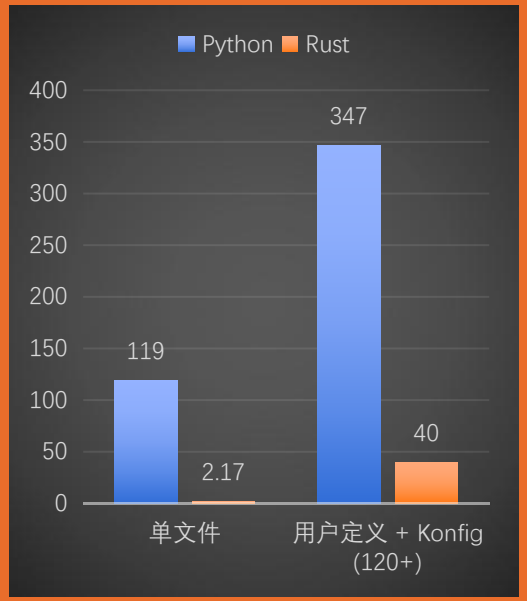
Case2: Konfig模型 + 用户定义

> https://github.com/KusionStack/konfig/blob/main/base/examples/native/nginx_deployment/nginx_deployment.k

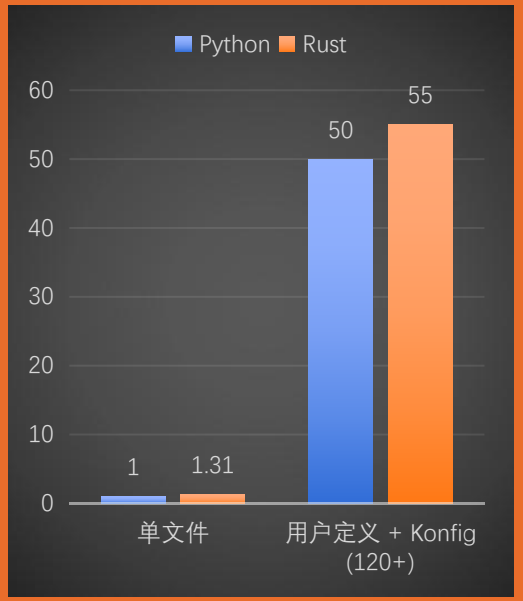
```
import base.pkg.kusion_kubernetes.api.apps.v1 as apps

demo = apps.Deployment {
    metadata.name = "nginx-deployment"
    spec = {
        replicas = 3
        selector.matchLabels = {
            app = "nginx"
        }
        template.metadata.labels = {
            app = "nginx"
        }
        template.spec.containers = [
            {
                name = "nginx"
                image = "nginx:1.14.2"
                ports = [
                    {containerPort = 80}
                ]
            }
        ]
    }
}
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

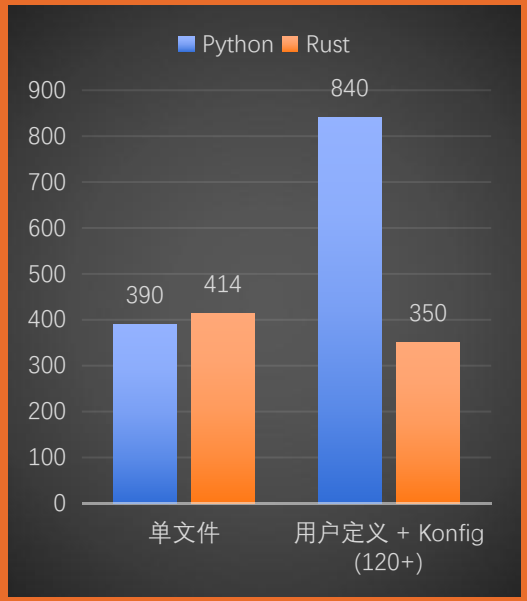
稳定性和性能的巨大提升



Parser



Resolver



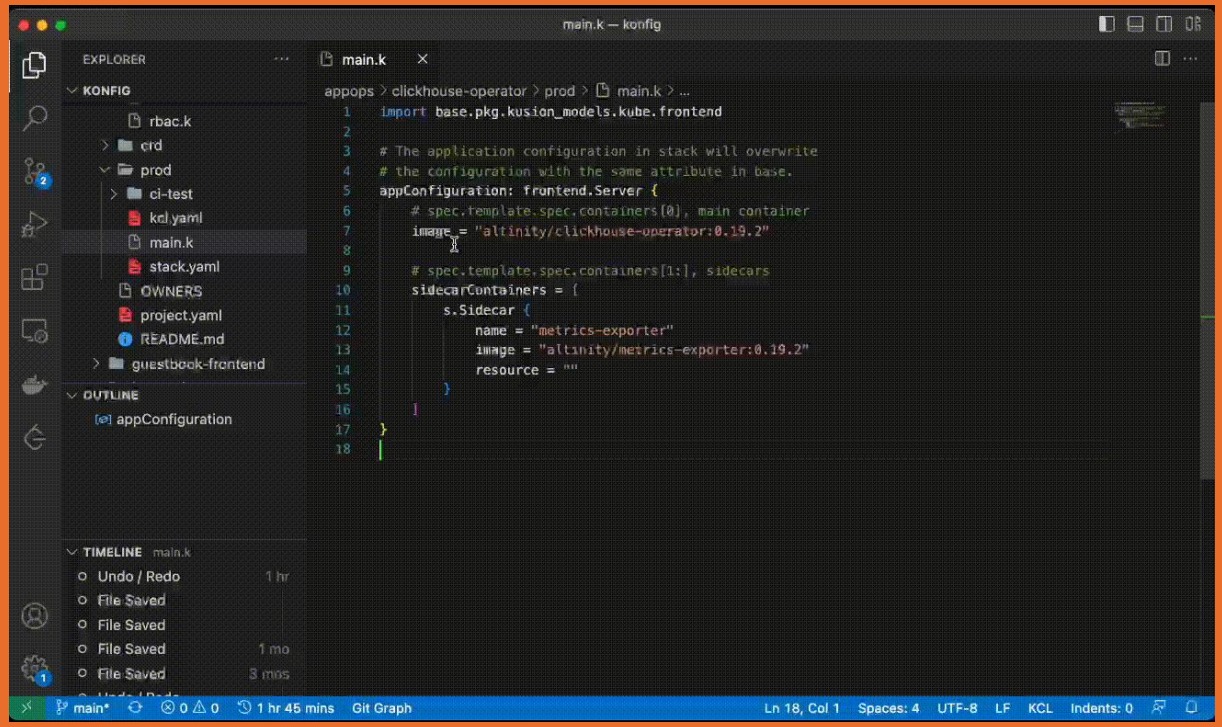
e2e

IDE: 用户体验的提升

端到端响应时间：

6s -> 100 ms

基于编译器前中端数十倍的性能提示，Rust 重写的 LSP 极大的提升了用户体验



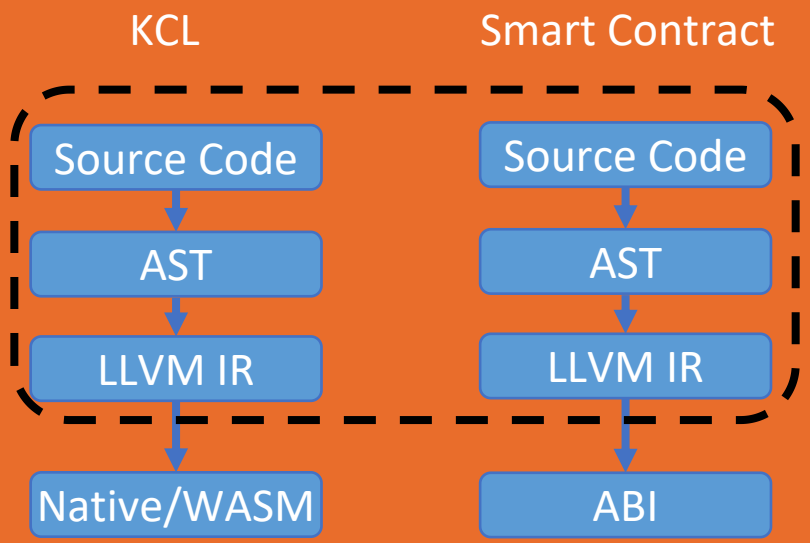
04
更多的探索
or
挖坑?

CompilerBase:
通用编译器组件

IDE 友好的编译器架构

RustCodeBook:
Rust源码解读

Compiler Base: 更通用的编译器组件



- Error
- Span & SourceMap
- Session
- Paraller
- ……

IDE 友好的编译器架构

1. 错误恢复: 不完整代码的编译

- 代码补全
- 错误代码的语义分析

2. 增量编译

- 大规模场景下的编译优化和 IDE 性能提升
- 编译粒度： 项目 -> 文件 -> 函数/定义

3. 结构化语义模型(Structured Semantic Model)

- Using the tree as a store for semantic info is convenient in traditional compilers, but doesn't work nicely in the IDE.

- A "structured semantic model" is basically an object-oriented representation of modules, functions and types which appear in the source code.This representation is fully "resolved": all expressions have types, all references are bound to declarations, etc.

RustCodeBook: Rust 源码解读

https://github.com/awesome-kusion/rust-code-book

欢迎加入我们

Web Site

- https://kusionstack.io/
- https://kcl-lang.io/

Github

- https://github.com/KusionStack/kusion
- https://github.com/KusionStack/kcl
- https://github.com/KusionStack/konfig

Twitter

- @KusionStack

微信

钉钉

Thank you!