KCL 配置策略语言

徐鹏飞 | 可信原生技术部

Agenda

01 背景

02 语言设计

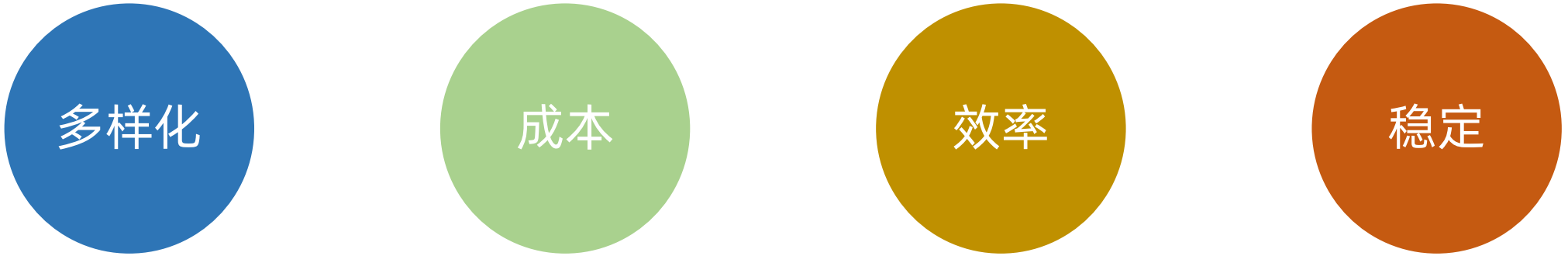
03 实现原理

04 未来展望

背景

01

背景 | 规模化配置策略编写



运维类研发统一为声明式 KCL 代码编写

背景 | 高度开放的云原生配置策略实践



头部公司已经大规模验证并实践过 IaC/PaC

语言设计

02

KCL 核心特性

- ✓ 有限语言能力
- ✓ 声明式为主
  - ✓ 记录
  - ✓ 函数
- ✓ 记录类型、数据分离
- ✓ 仅支持必要语言功能

- ✓ 强不可变性
- ✓ 静态类型
  - ✓ 类型推导
  - ✓ 类型检查
- ✓ 约束校验
- ✓ 可测试

- ✓ 多模式配置合并
  - ✓ 多环境
  - ✓ 多租户
- ✓ Schema + Mixin
- ✓ 模块化、包

- ✓ 自动化集成
  - ✓ CRUD
- ✓ 内置函数+系统库
- ✓ Plugin 扩展
- ✓ OpenAPI Model
- ✓ IaD 亲和

- ✓ Rust
- ✓ LLVM 优化器
- ✓ Native + WASM

为运维业务协同研发设计的专用语言

语言设计 | 简单为王-声明式语法

KUSION STACK

```
import base.pkg.kusion_models.kube.frontend

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

- 声明式语法简化配置 模型+实例 编写

语言设计 | 简单为王-声明式语法

```
schema Fib:
    n1: int = n - 1
    n2: int = n1 - 1
    n: int
    value: int

    if n <= 1:
        value = 1
    elif n == 2:
        value = 1
    else:
        value = Fib {n: n1}.value + Fib {n: n2}.value

fib8 = Fib {n: 8}.value  # 21
```

- 支持代码逻辑顺序无关编写，降低编写心智
- 默认值、配置值、逻辑判断、...

语言设计 | 稳定-静态类型

KUSION STACK

```
schema Deployment:
    apiVersion: str = "apps/v1"
    kind: str = 123  # 类型错误

schema ContainerPort:
    name?: str  # 可选属性
    protocol: "TCP" | "UDP" | "SCTP" = "TCP"
    containerPort: int
```

- 通过静态类型系统确保配置正确
- 保证 Schema 属性强制非空，避免配置遗漏

语言设计 | 稳定-配置约束校验

```
schema App:
    domainType: "Standard" | "Customized" | "Global"
    containerPort: int
    volumes: [Volume]
    services: [Service]

    check:
        1 <= containerPort <= 65535, "containerPort must be between 1 and 65535"
        all service in services {
            service.clusterIP == "None" if service.type == "ClusterIP"
        }
        all volume in volumes {
            volume.mountPath not in ["/", "/boot", "/home", "dev", "/etc", "root"]
        }
```

- 支持自定义约束校验规则

语言设计 | 稳定-测试工具

```
schema Person:
    name: str = "kcl"
    age: int = 1
```

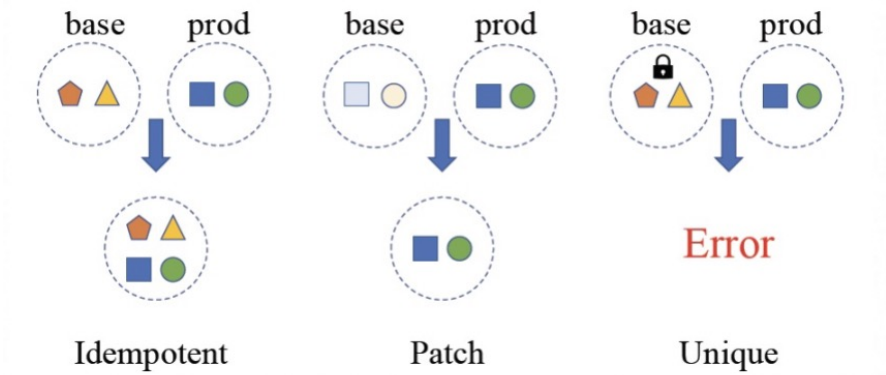
```
schema TestPerson:
    a = Person {}
    assert a.name == 'kcl'

schema TestPerson_age:
    a = Person {}
    assert a.age == 1
```

```
kclvm path: /bin/kclvm
ok /KCLVM/samples [44.018277ms]
```

- 内置 kcl-test 对单元测试、集成测试和插件测试提供友好支持

语言设计 | 协同编写



- 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
    }
}
```

等效配置代码

```
# 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
    }
}
```

- 多团队多模块并发协同维护配置数据

语言设计 | 协同编写 - Mixin 声明式组装

```
schema ServerBackend[inputConfig: server.Server]:
    """ServerBackend converts the user-written front-end model `Server` into a
    collection of kubernetes resources and places the resource collection into
    the `kubernetes` attribute.
    """
    mixin [
        # Resource builder mixin
        mixins.NamespaceMixin,
        mixins.ConfigMapMixin,
        mixins.SecretMixin,
        mixins.ServiceMixin,
        mixins.IngressMixin,
        mixins.ServiceAccountMixin,

        # Monitor mixin
        pod_monitor_mixin.PodMonitorMixin,
        mixins.OutputTypeMixin
    ]
```

- 通过 Mixin 复用扩展逻辑

语言设计 | 自动化

KUSION STACK

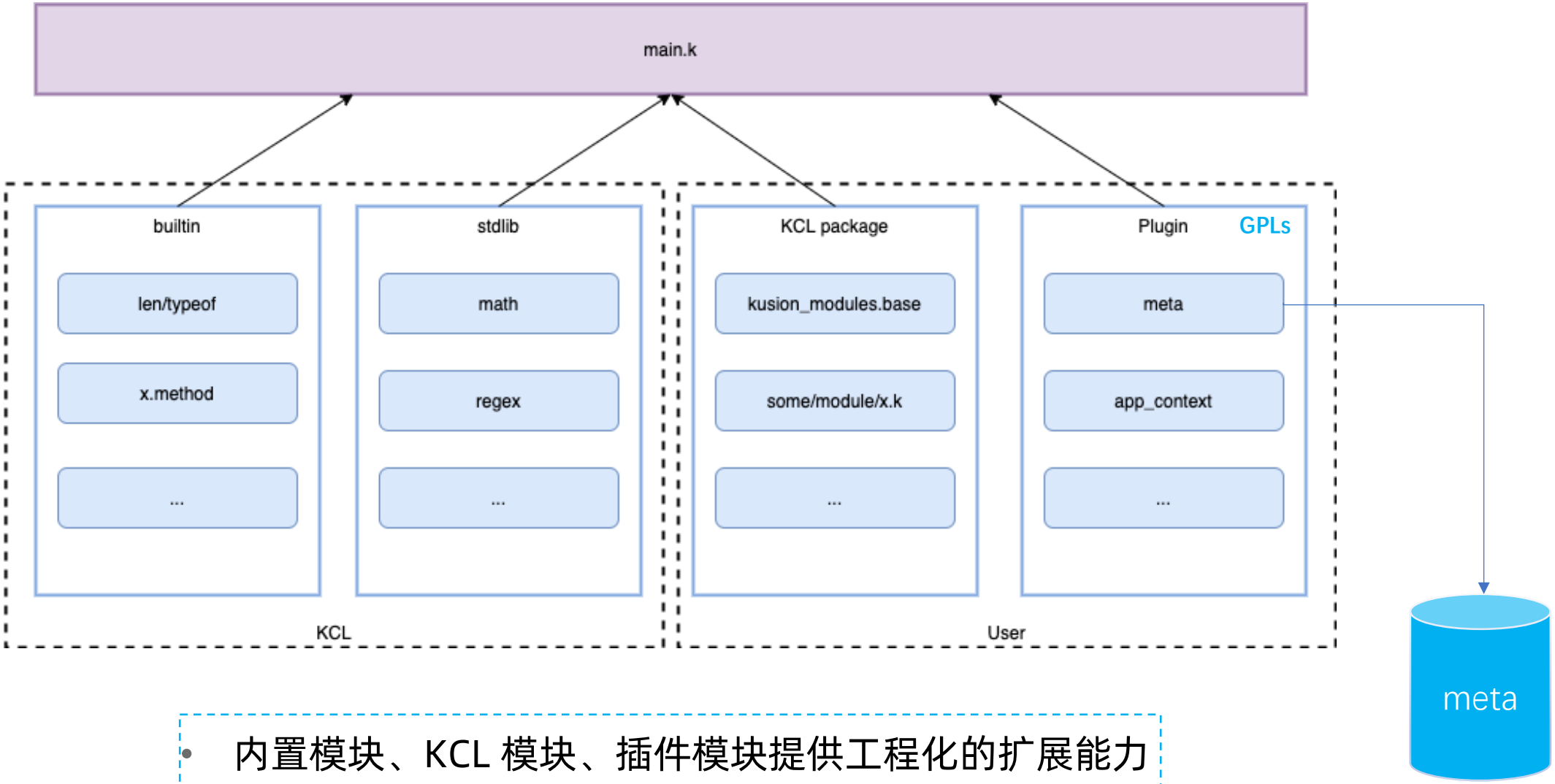
kcl -O appConfiguration.image='"nginx:1.7.9"'

```
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
```

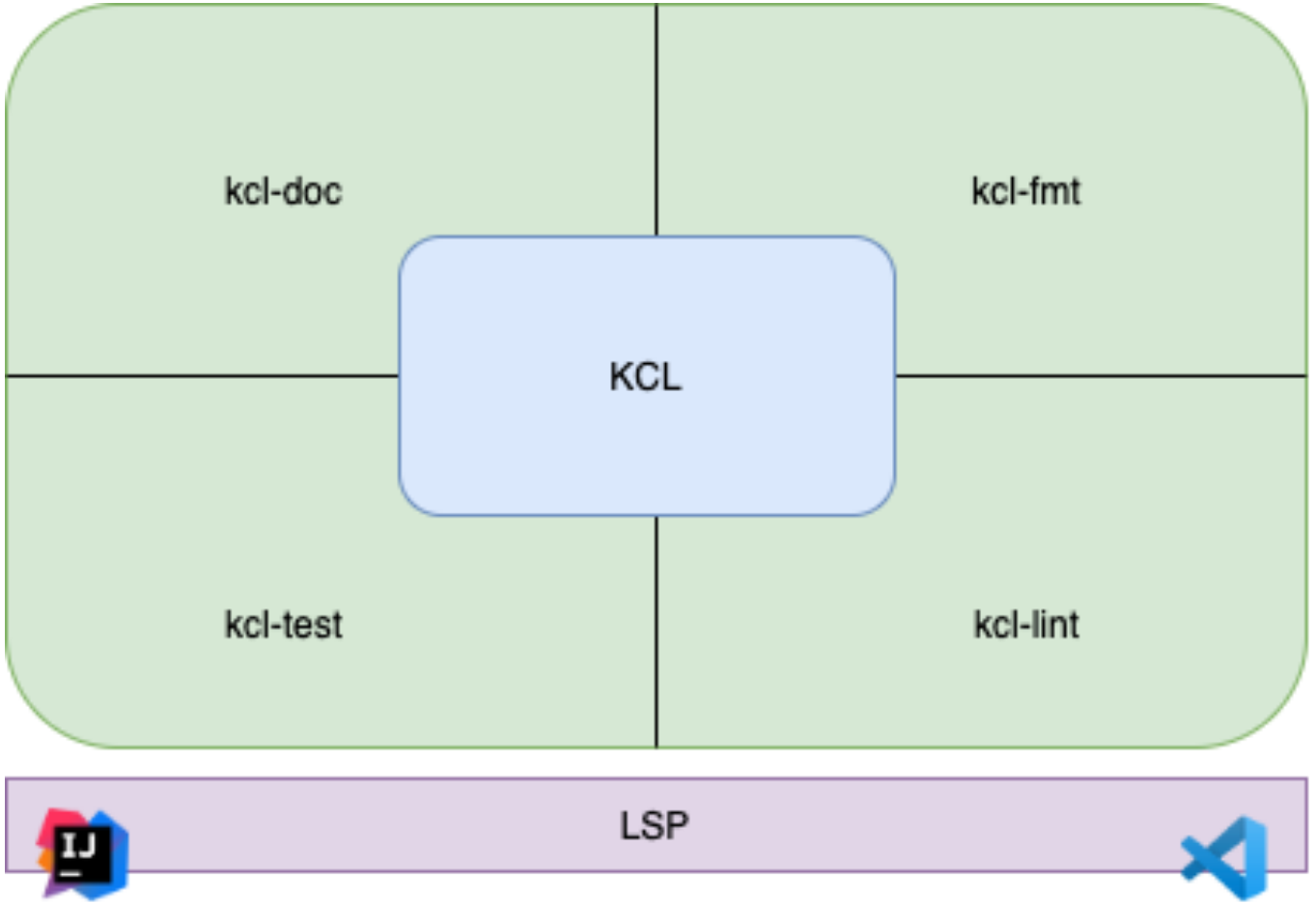
- 支持 CLI/API 等方式进行配置 CRUD & 自动化集成

语言设计 | 工程化-内置模块、用户模块、插件模块

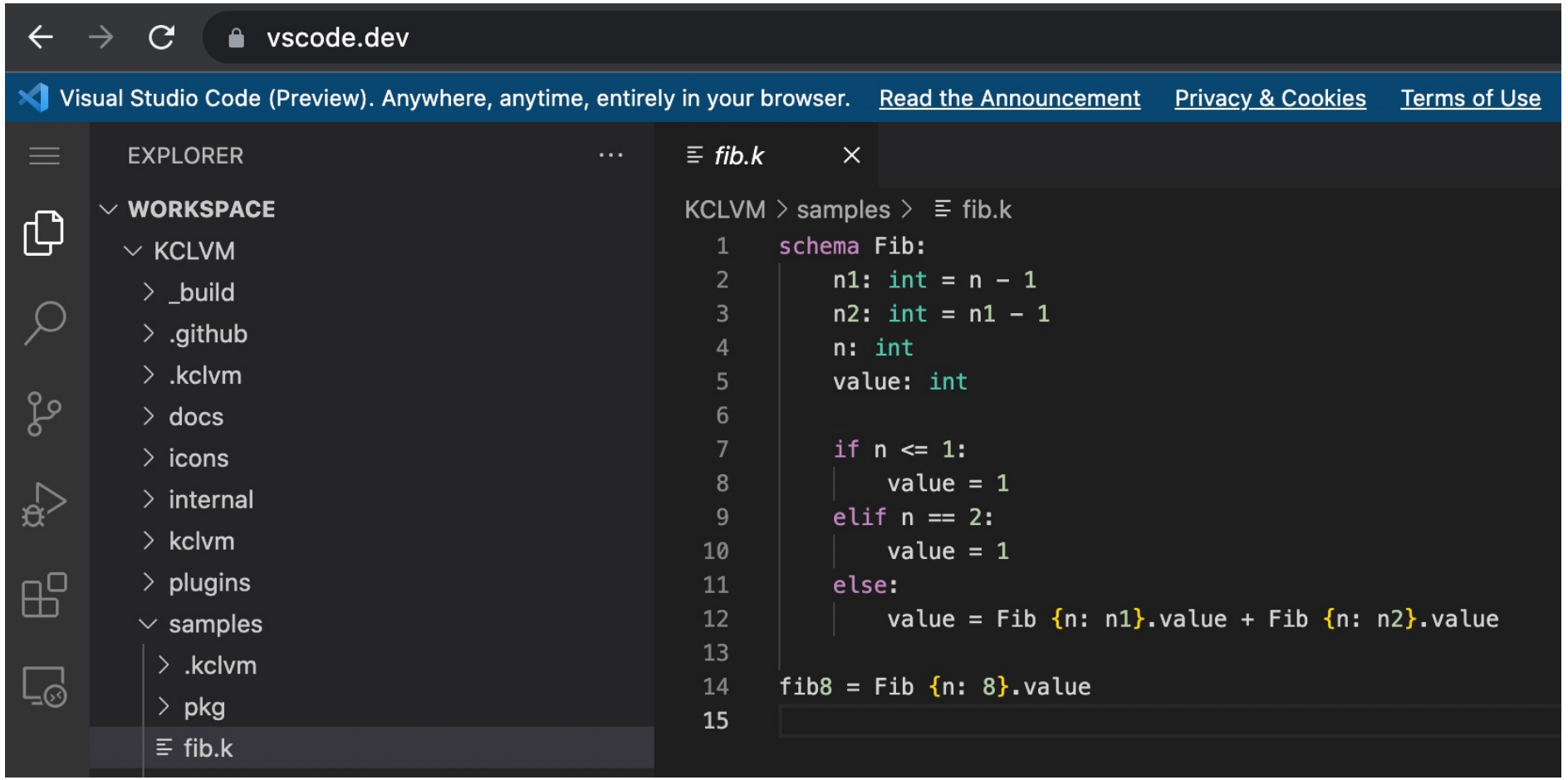


- 内置模块、KCL 模块、插件模块提供工程化的扩展能力

语言设计 | 工程化-完善的外围工具



语言设计 | 工程化-本地/云 IDE 插件

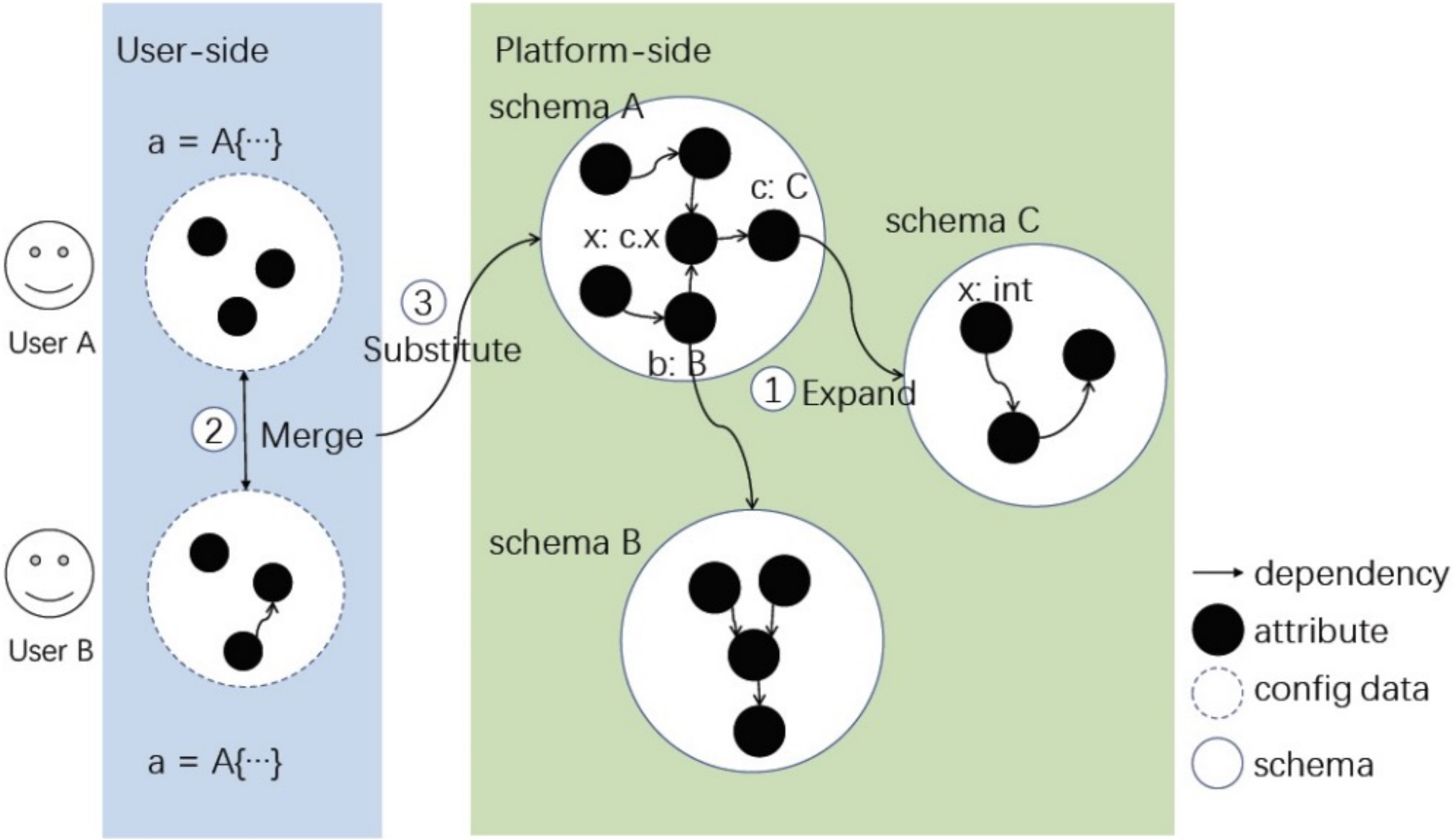


- https://vscode.dev

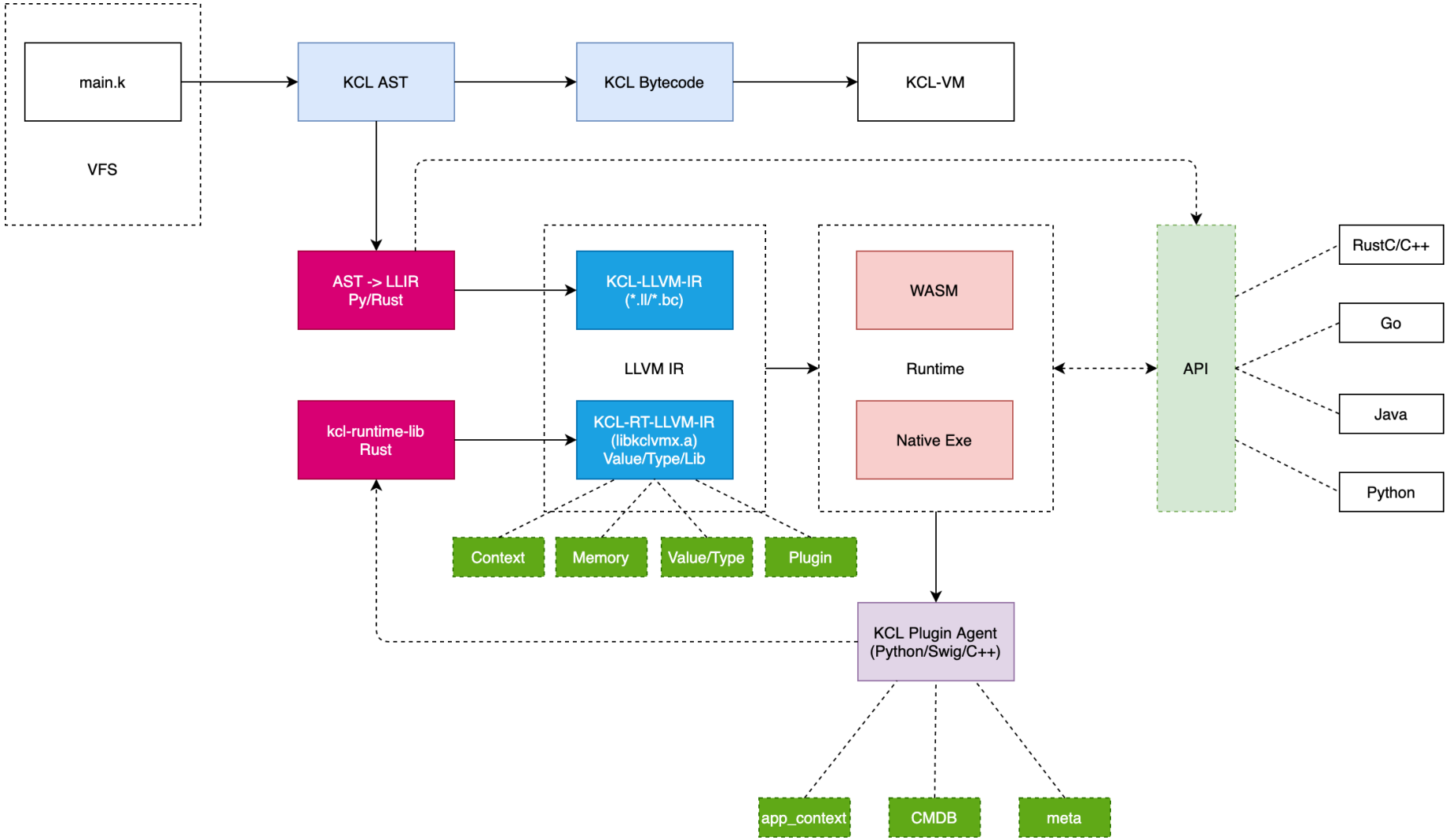
实现原理

03

实现原理 | 配置图合并


User-side
a = A{···}
User A
2
Merge
User B
a = A{···}
3
Substitute
Platform-side
schema A
c: C
x: c.x
b: B
1
Expand
schema C
x: int
schema B
dependency
attribute
config data
schema

实现原理 | 编译器架构


main.k
VFS
KCL AST
KCL Bytecode
KCL-VM
AST -> LLIR
Py/Rust
KCL-LLVM-IR
(*.ll/*.bc)
LLVM IR
WASM
Runtime
Native Exe
API
RustC/C++
Go
Java
Python
kcl-runtime-lib
Rust
KCL-RT-LLVM-IR
(libkclvmx.a)
Value/Type/Lib
Context
Memory
Value/Type
Plugin
KCL Plugin Agent
(Python/Swig/C++)
app_context
CMDB
meta
KCL Spec
KCL Multi-
Language Spec
KCL OpenAPI
Spec

未来展望

04

未来展望

KUSION STACK

语言改进

- ✓ 特性简化
- ✓ 稳定性提升
- ✓ 约束策略支持
- ✓ 语法、语义扩展
- ✓ 性能
- ✓ ...

生态扩展

- ✓ 包管理 & 代码分享
- ✓ 多种语言 binding
- ✓ 多语言 API
- ✓ 工具 & 多 IDE 支持
- ✓ ...

目标领域增强

- ✓ 多种运行时
- ✓ 多种后端
  - ✓ BPF
  - ✓ WASM
  - ✓ ...
- ✓ JIT 解释执行

KCLVM Roadmap Discussion Issue: https://github.com/KusionStack/KCLVM/issues/29

Q & A

欢迎大家用钉钉扫码
或钉钉搜索：42753001
加入 KusionStack 官方交流群

欢迎大家用微信扫码
加入 KusionStack 官方微信群

KUSION STACK

THANKS