SAVE*

Cloud-based distributed CI/CD platform for testing and benchmarking of dev-tools

***S**tatic **A**nalyzer's **V**erification and **E**valuation

01

PROBLEM

Why do we need new narrow-focused test framework?

02

SAVE-CLOUD

User scenarios for your CI/CD process

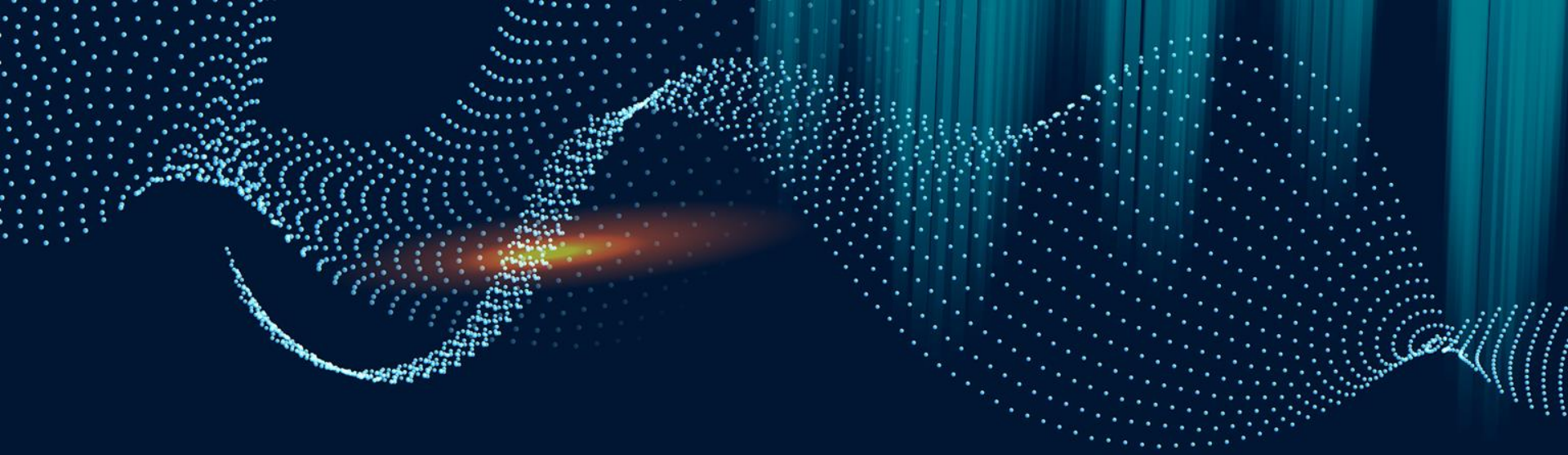
03

CLI-FRAMEWORK

Core framework, plugins and more

“Software never was perfect and won’t get perfect. But is that a license to create garbage?”

— Boris Beizer, author, “Software Testing Techniques“



01 PROBLEM

Why do we need new narrow-focused test framework?

Initial steps before the development of any new static analyzers/compilers/translators/e.t.c.

POC

Create basic things, so you will get something able to run

Collecting initial test-snippets

Understanding the Scope

Preparation for testing

Writing **yet another** test-framework for your purposes

Writing the code and hundreds of tests

Active Development

*https://github.com/cqfn/diKTat

What is the functionality of these frameworks?

What do these framework really test in the area of static analysis or compilers?

GENERALIZED TEST-SCENARIOS

1 FIND THE DIFFERENCE

1. **Execute** your tool that should be tested
2. **Use** some test file for the execution
3. **Compare** stdout/resulted file with some other expected test file

2 CHECK THE OUTPUT

1. **Execute** your tool that should be tested
2. **Use** some special test file (with metadata) for the execution
3. **Compare** the output (exact match of warnings/stderror/stdout to the metadata)

Example for static analyzers

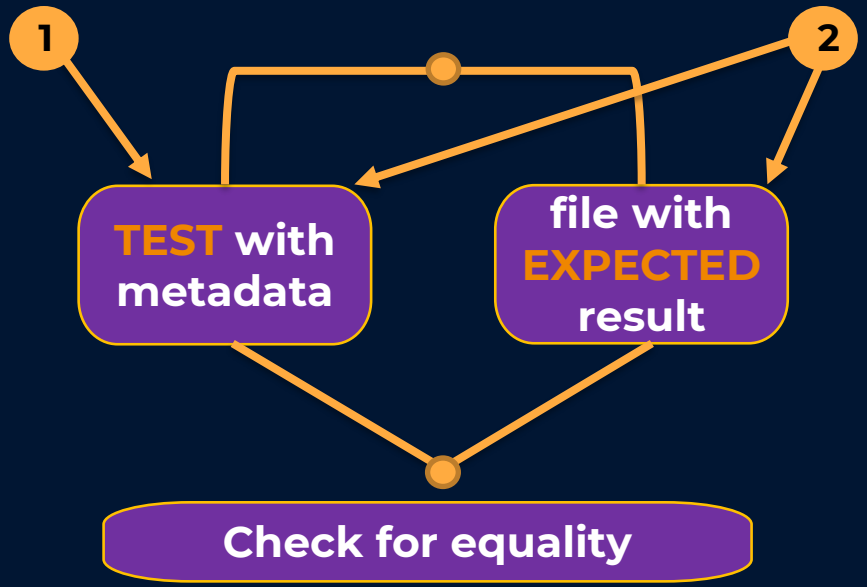
Imagine you have some **linter** with auto-fix functionality, like diktat/ktlint/yapf/idea/e.t.c.

You would like to check:

1. How it can fix the code (in-place, in a separate file, stdout)
2. Check how it detects errors/warnings in your code

Even test resources for 1 and 2 **could be the same**

Example for static analyzers



Example for compilers

Imagine you have some **compiler, translator or interpreter**

You would like to check:

1. What kind of code it generates (IR/machine code/e.t.c)
2. Which errors and warnings it generates on your code (parsing errors, compiler warnings, e.t.c)
3. How the app is executed (result code, produced file, result of the execution of the code)

3 is a superposition of scenarios 1 and 2

Where else these scenarios can be useful?

Static analyzers	Code formatters	Compilers and interpreters
Serializarion libraries	Translators	Code generators, low-code/no-code

Available tools

Opensource

The are **NO** specific frameworks for testing these groups of tools, each time developers re-invent a **NEW** tool

LIT/GCC test

LLVM LIT – compiler-specific Python framework.
Custom plugins are not supported, logic is mainly hardcoded for C/C++

JDK run-test framework

Mess of different groups of tests with custom drivers **without** a common core logic (JTReg, Gtest, Micro)

What else is missing in the area of static analysis?

Benchmarking!

Did we have anything succesfull in this area?

SPEC.ORG

- Standard Performance Evaluation Corporation (SPEC)
- Evaluates **performance** and energy efficiency for the newest generation of computing systems
- Creates **suites**, reviews and publishes submitted results

MISRA

- Software development guidelines for the **C/C++**
- Facilitates code safety, security, portability and reliability
- Misra-C example test suites

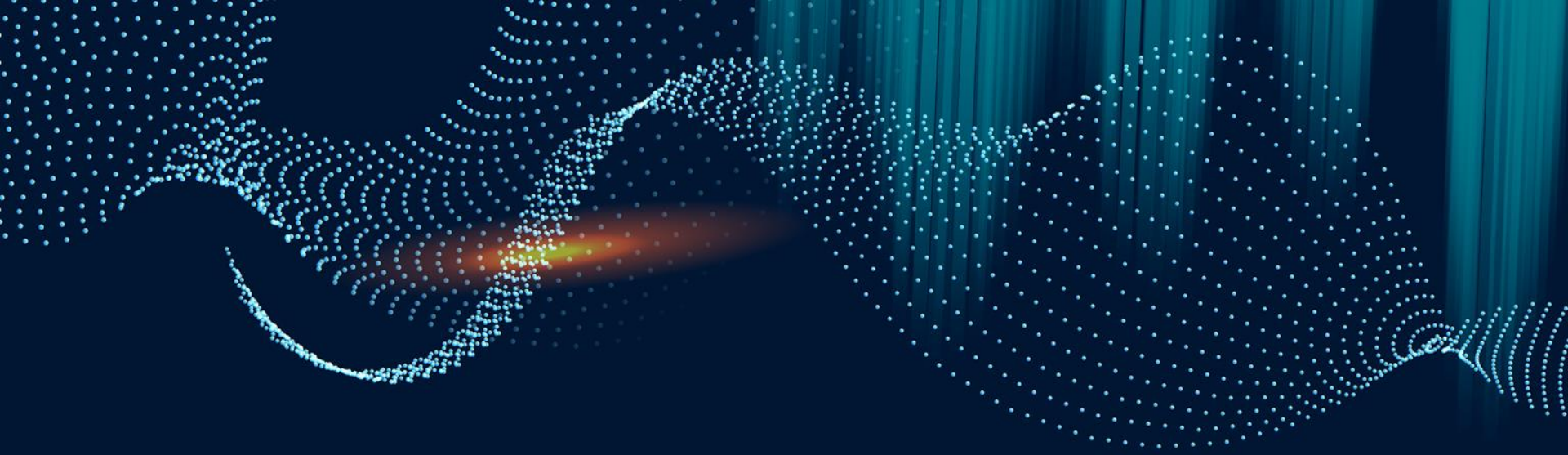
No specific benchmarks/suites for other programming languages.
We can't evaluate static analyzers properly.

Performance and resources

Test suites for such tools can be very large as they should cover thousands of combinations of samples. We see a demand for parallisation technologies in this domain

350,000+

This is how many funtional tests are there in **GCC**. **LLVM** and **Clang** have mostly same huge test packages. Industrial compilers, like **ICC,** can have **1 mln+** tests



02 SAVE-CLOUD

CI/CD framework for distributed test runs

Save-cloud

Application for a distributed parrallel run of tests in cloud

- Save-cloud downloads **your test resources** OR uses **it's own default** benchmarks(TBD) to run tests for your application
- **Historical results** of tests are stored in the database
- All **batching and scaling** is done automatically under the hood
- It's **own frontend** (TBD) and **monitoring system** for a better user-experience

IMPROTANCE

FAST

Save-cloud is used to create **multiple containers** with save images

EASY

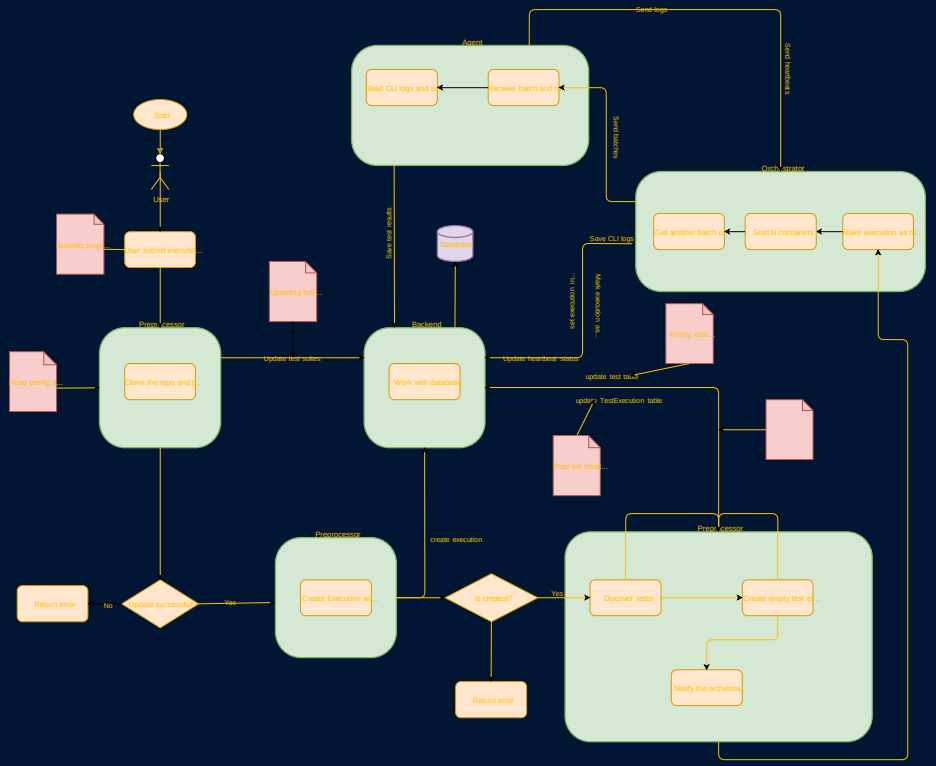
All components are **simple microservices** that communicate via REST-API

EXTENDABLE

Safe Docker images are used to scale the number of agents

CONTINUITY

Save-cloud stores all results of test-runs in the database, so you can compare **historical results** and **detect regressions**



Complex architecture

Save cloud is build using a cutting-edge enterprise technology stack

Full size diagram:
https://github.com/cqfn/save-cloud

Save-cloud main components

Orchestrator
Agent and processing management

Agent
Save (cli) running in the Ubuntu docker container

Backend
All related work with the database (processing of results)

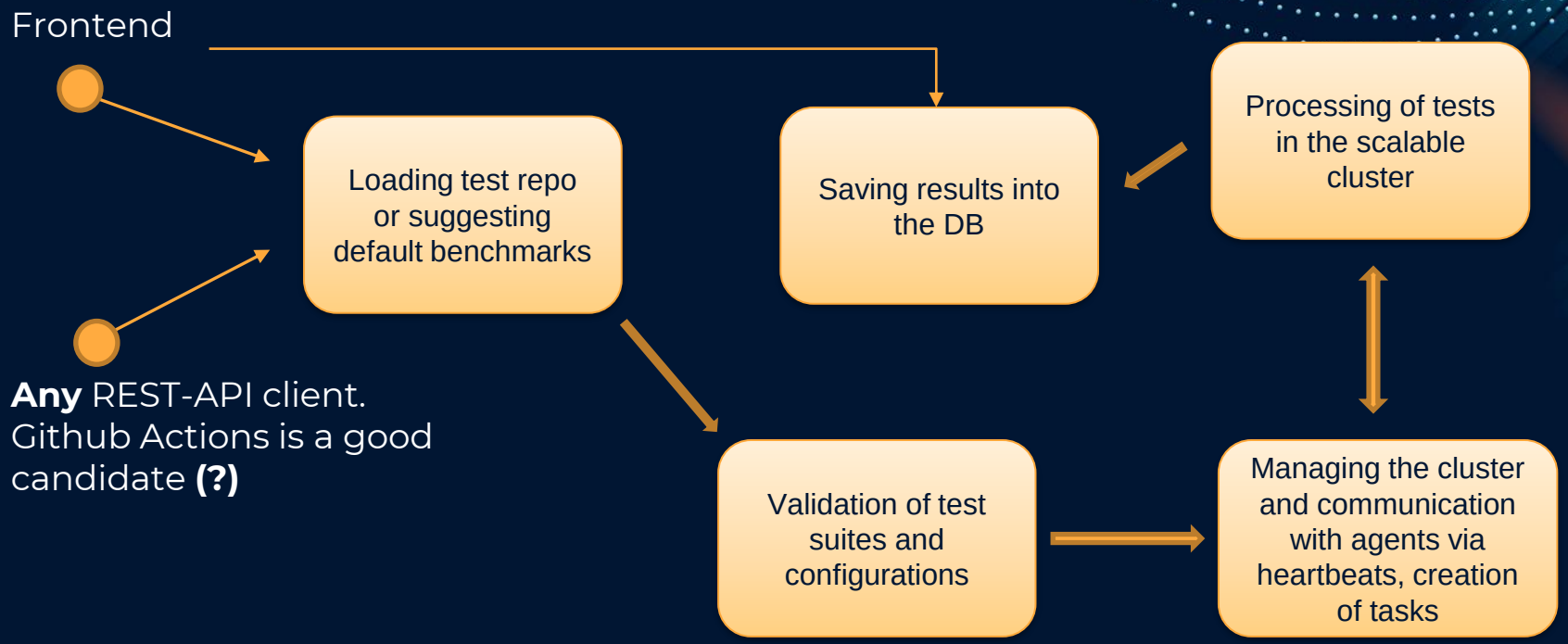
Preprocessor
Work with the test-repository (loading, checking, test-discovery)

All components are using **JVM and Kotlin**

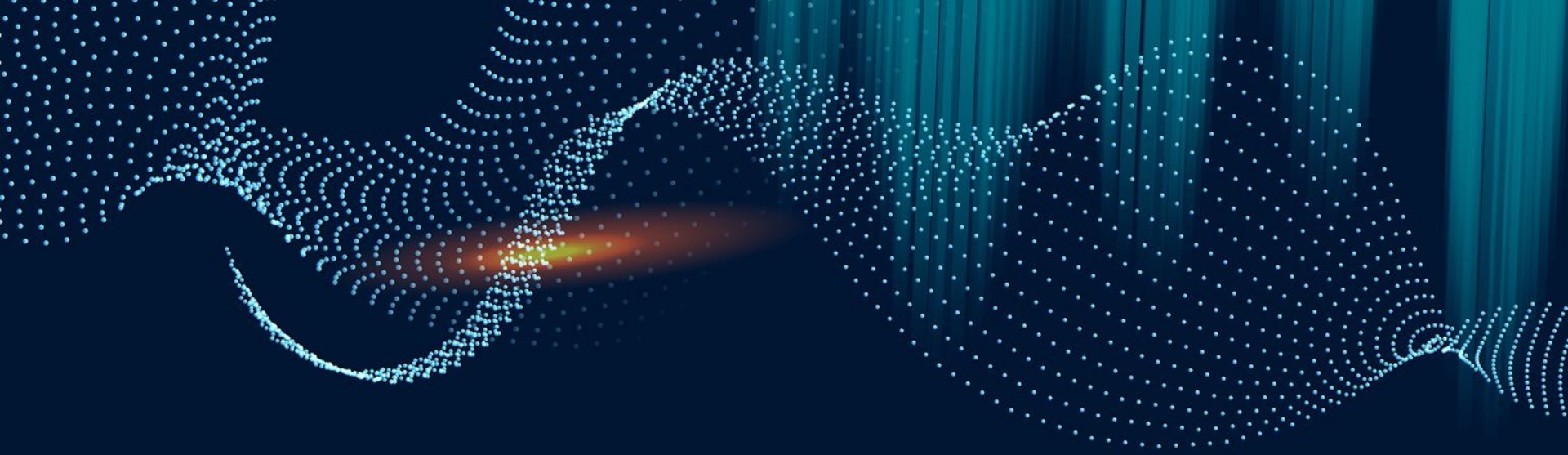
Technology stack

	Technology stack
Orchestrator	Docker, Prometheus, Graphana, testcontainers
Agent	Kotlin Native, KTOR (communication via heartbeats)
Preprocessor	Spring Boot 2.5, Spring Security
Backend	Spring Boot 2.5 WebFlux (Project Reactor stack), Hibernate/JPA, Mysql, Liquibase

SIMPLIFIED PROCESSING



Any REST-API client.
Github Actions is a good candidate **(?)**



03

SAVE

User-friendly test-framework (cli)
for system programmers

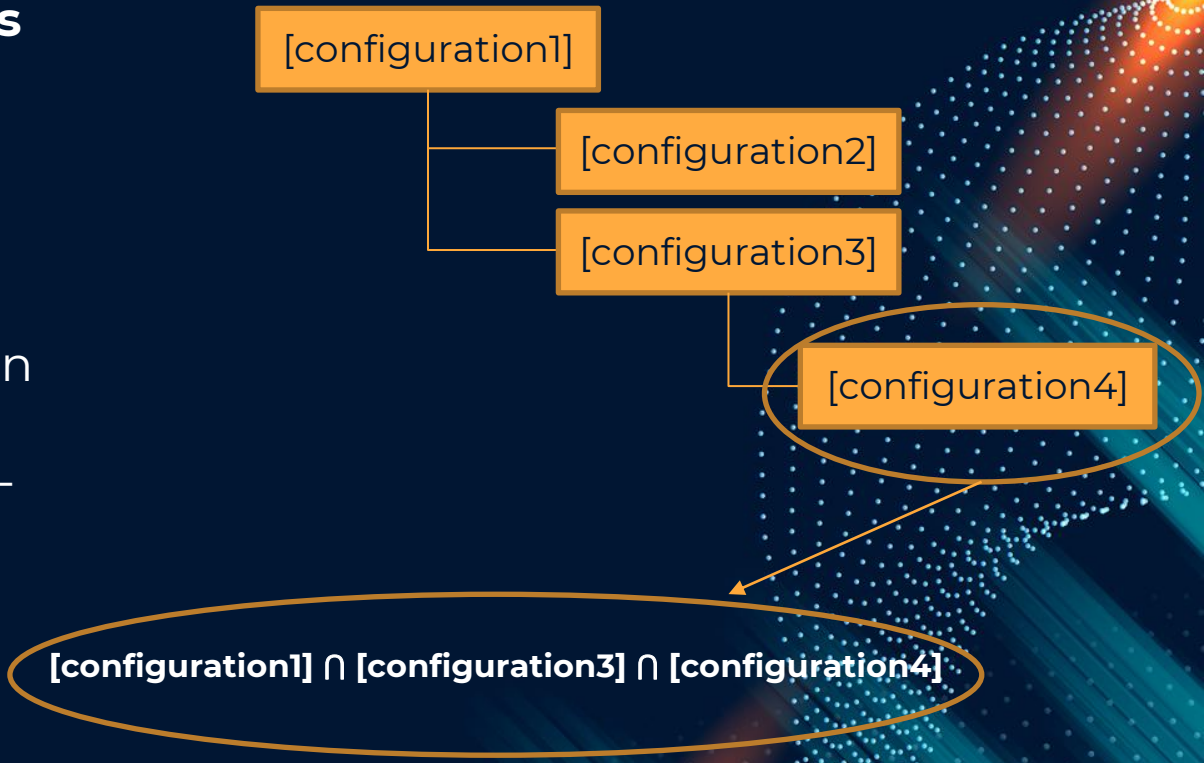
Save-cli

Framework for running tests for dev-tools. Focused on the problems of testing static analyzers/ compilers and other language-related tools and libs

- **Native** application (linuxX64, macosX64, winX64)
- Can also be run via **JVM** (slower than Native)
- Supports **hierarchical** easy-to-read **TOML** configuration
- Extendable: supports **plugins** that can be written on Kotlin
- Supports different types of reporting

Configuration

- Save **recursively detects** all test scenarios in your project
- You don't need to have boring duplicated configurations as you can simply put common configuration to a super-config of your test suite



Toml configuration

- Easy-to-read **TOML** language with separation of sections (tables) is a **perfect candidate** for being selected as a format for test-configs in save.

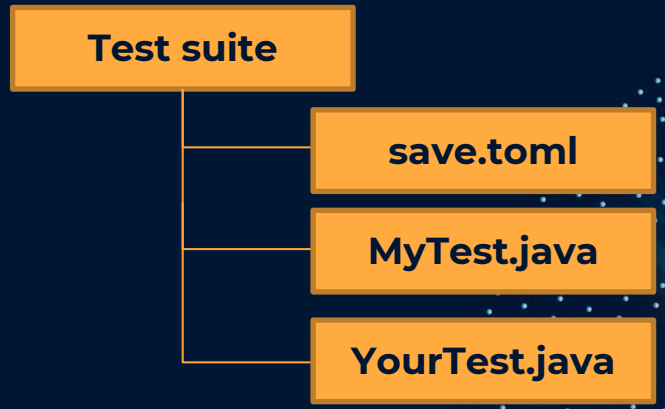
- We prepared our own deserialization library for it on Kotlin Native: https://github.com/akuleshov7/ktoml

```
[general]
tags = "tag1"
description = "My suite description"
suiteName = "DocsCheck"

[warn]
execCmd = "./detekt --build-upon-default-config -i"
warningsInputPattern = "// ;warn:(\\d+):(\\d+): (.*)"
warningsOutputPattern = "\\w+ - (\\d+)/(\\d+) - (.*)$"
lineCaptureGroup = 1
columnCaptureGroup = 2
messageCaptureGroup = 3
warningTextHasColumn = true
warningTextHasLine = true

[fix]
execCmd="./ktlint -R diktat-0.4.2.jar -F"
```

Detection of test resources

- Each Save Suite (directory with tests) should have a “**save.toml”** config. It will be applied to all resources in the same directory

- It depends on the plugin how to determine test resources. But **generally** it is detected using the suffix in the test name (My**Test**.c or My**Expected**.java)



Default plugins

1 [FIX] plugin

1. **Execute** your tool that should be tested on the test resource with **Test** suffix in the name
2. **Compare** the result with the resource with **Expected** suffix in the name

2 [WARN] plugin

1. **Execute** your tool that should be tested on ~~the~~ test resource with **Test** suffix in the name
2. **Map and Compare** the output with special metadata

```
// ;warn:1:7: Class name should be in an uppercase format
// ;warn:3:13: Method B() should follow camel-case convention
class a {
    // ;warn:2:13: Single symbol variables are not informative
    // ;warn:2:14: Trailing semicolon is redundant in Kotlin
    val b: String;
    fun B(): String {}
    fun setB(): String {}
}
```

Custom plugins

- Now we can support injection of custom plugins **only in JVM mode** (and not supporting in Native mode). We plan to support this via dynamic **.so** libraries

- But anyway there is a **common interface for Plugins** that can be extended directly in the framework

Reporter formats

- We have a **common interface** for reporters, so community can easily support their own format
- We support: **PLAIN**
- We expect to support: **JSON, TOML, YML**

WORK WITH COMMUNITY

We expect that opensource community will use SAVE as there are **no analogues** of such tool

We expect that community will help us in writing **plugins and benchmarks**

We expect to grow an organization for **certification** and comparison of static analyzers

THANKS!

See our SAVE project and give it a star