ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Análise de variantes com CodeQL

ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

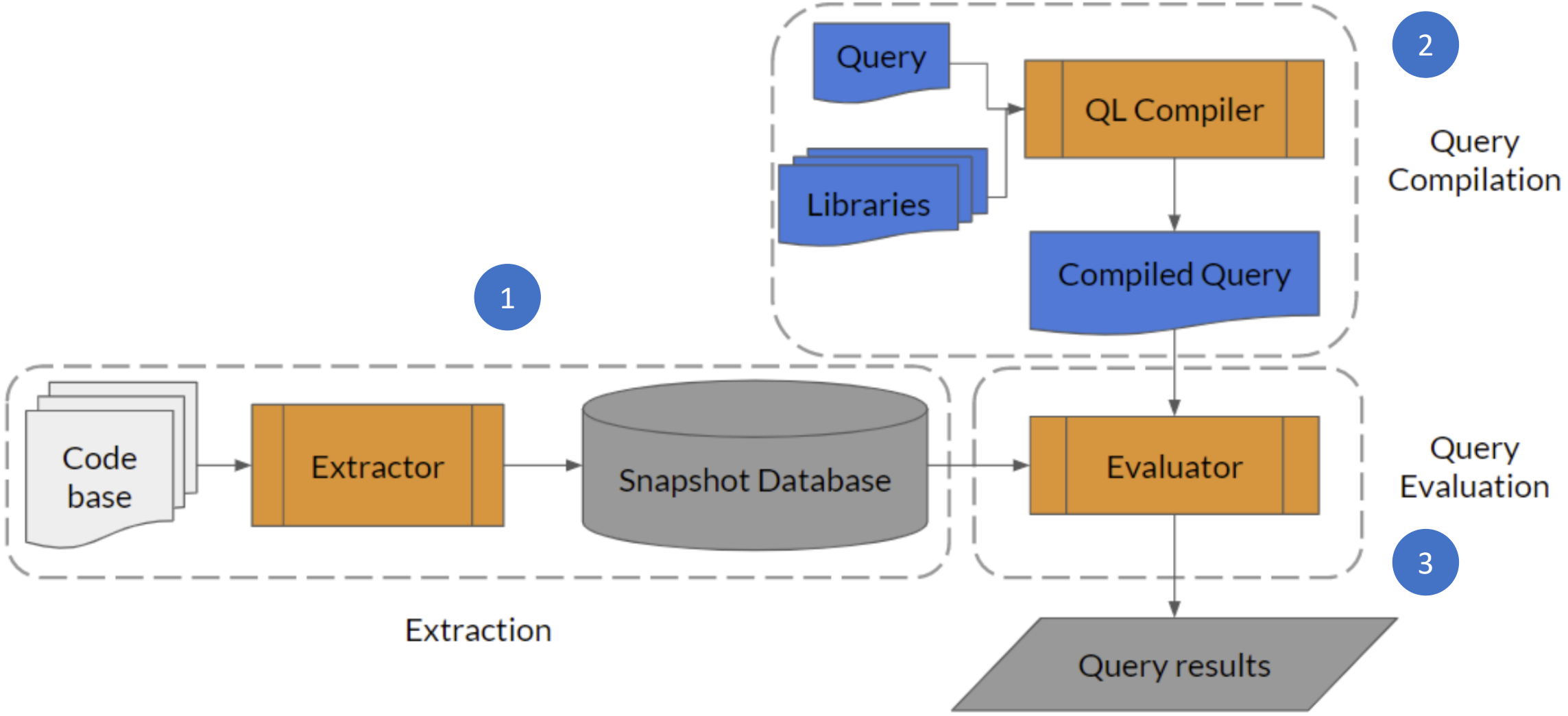
Análise de variantes

- A análise/auditoria manual de código procura por *bugs* que levem a vulnerabilidades de segurança
- A análise de variantes é o processo de usar como modelo uma vulnerabilidade conhecida e procurar problemas semelhantes no código:
  - Modelar o problema de segurança de maneira a poder ser aplicado a uma representação do programa
  - Varrer a base de código procurando instâncias do problema de segurança modelado
  - Adicionar o modelo a um repositório e usá-lo no processo de compilação da aplicação (integração contínua)
- O sistema CodeQL é uma linguagem e uma plataforma para automatizar a análise de variantes
  - Encontrar bugs, vulnerabilidades de segurança, realizar a análise de forma sistemática e com possibilidade de partilhar conhecimento

CodeQL

- Sistema desenvolvido pelo GitHub
- A linguagem QL (*Query Language*) é:
  - Uma linguagem lógica baseada na lógica de primeira ordem
  - Uma linguagem declarativa sem *side-effects*
  - Uma linguagem orientada a objetos
  - Um linguagem de interrogações sobre uma base de dados CodeQL apenas de leitura
  - Tem de base bibliotecas e padrões para análise de programas
- Uma plataforma
  - Motor de análise de interrogações
  - Ferramentas de linha de comando

Arquitetura do sistema CodeQL



Exemplo de um *bug*

```
int write(int[] buf, int size, int loc, int val) {
  if (loc >= size) {
    // return -1;
  }

  buf[loc] = val;

  return 0;
}
```

- A instrução de retorno foi comentada (ex: para efeitos de *debug*)
- A instrução `if` agora é código sem utilidade
- Sem a verificação explícita de limites o código lançará `ArrayIndexOutOfbounds`

Exemplo de interrogação sobre código Java

Reutilização de lógica sobre a linguagem em análise

```
import java
```

```
from IfStmt ifstmt, BlockStmt block
where
  block = ifstmt.getThen() and
  block.getNumStmt() = 0
select ifstmt, "This if-statement is redundant."
```

Interrogação que descreve o que se está a tentar encontrar

- Um ficheiro de interrogação tem a extensão `.ql` e contém uma cláusula de consulta e, opcionalmente, predicados, classes e módulos.

Predicados

```
import java


from IfStmt ifstmt, BlockStmt block
where
  block = ifstmt.getThen() and
  block.getNumStmt() = 0
select ifstmt,
       "This if-statement is redundant."
```

```
import java

predicate isEmpty(BlockStmt block) {
  block.getNumStmt() = 0
}

from IfStmt ifstmt
where isEmpty(ifstmt.getThen())
select ifstmt
```

- Um predicado permite colocar em evidência parte da interrogação

Classes

- As classes em QL estendem um ou mais tipos, representam um conjunto de valores e definem predicados

```
class OneTwoThree extends int {
  OneTwoThree() { this = 1 or this = 2 or this = 3 } // characteristic predicate

  // member predicate
  string getAString() { result = "One, two or three: " + this.toString() }

  // member predicate
  predicate isEven() {this = 2 }
}
```

- A definição do corpo da classe consiste num predicado de carcterística (opcional) e num ou mais predicados membros

Classes

```
import java

predicate isEmpty(BlockStmt block) {
  block.getNumStmt() = 0
}

from IfStmt ifstmt
where isEmpty(ifstmt.getThen())
select ifstmt
```

```
import java

class EmptyBlock extends BlockStmt {
  EmptyBlock() {
    this.getNumStmt() = 0
  }
}

from IfStmt ifstmt
where ifstmt.getThen() instanceof
      EmptyBlock
select ifstmt
```

- A classe EmptyBlock é um BlockStmt cujo número de instruções é zer0

Refinar sucessivamente o código

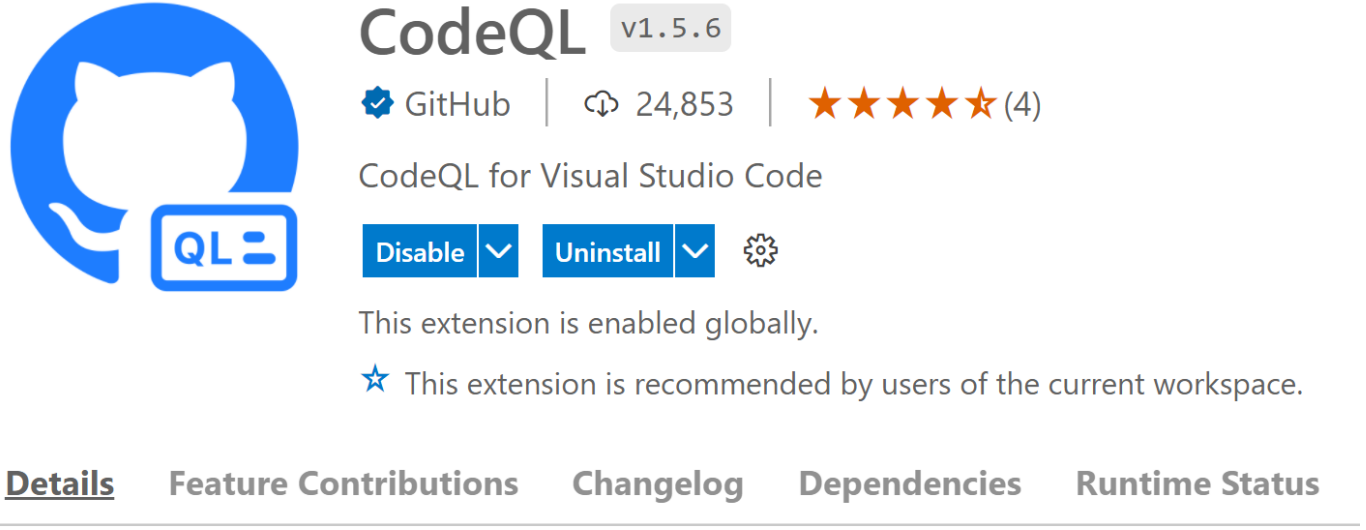
- Procura por instruções de `if` sem corpo no `then` e sem código no `else`

```
import java

class EmptyBlock extends Block {
  EmptyBlock() { this.getNumStmt() = 0 }
}

from IfStmt ifstmt
where
  ifstmt.getThen() instanceof EmptyBlock and not exists(ifstmt.getElse())
select ifstmt, "This if-statement is redundant."
```

Instalação do CodeQL e extensão no VSCode

- Command Line Interface:
  - https://github.com/github/codeql-cli-binaries/releases

- https://codeql.github.com/docs/codeql-for-visual-studio-code/setting-up-codeql-in-visual-studio-code/

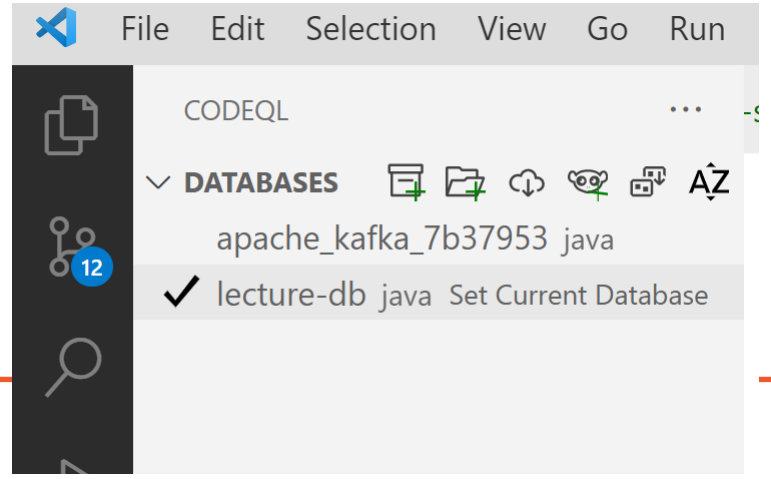


Utilização

- A instalação da extensão incluí uma base de dados para o projeto Apache Kafka e interrogações em algumas linguagens
  - https://kafka.apache.org/, um sistema para a distribuição de mensagens Pub/Sub, análise de dados em stream, …
- Criação de novas base de dados

```
$ codeql database create <db path> -l java
```

  - Comando assume que código fonte do projeto a analisar está na directoria atual
  - Tem de ser possível compilar o projeto com ferramenta de build automática (ex: Maven, Gradle)
- Gestão das base de dados na extensão do VSCode
- Extensão permite visualizar AST



Análise de fluxo em Java

- O módulo DataFlow define a classe Node que representa qualquer elemento pelo qual pode passar informação (expressões, parâmetros, etc.)
- O módulo TaintTracking faz análise de fluxo local com *tainting*

```
TaintTracking::localTaint(DataFlow::parameterNode(source), DataFlow::exprNode(sink))
```

- A classe Configuration faz análise de fluxo global

- Como realizar uma *query* que deteta a criação de ligações HTTP a partir de um valor obtido de uma variável de ambiente?

Resumo da interrogação

```
/**
 * Finds environment variable used to create an URL object
 */

import java

class GetenvSource extends DataFlow::ExprNode { ... }

class GetenvToURLConfiguration extends DataFlow::Configuration { ... }

from DataFlow::Node src, DataFlow::Node sink, GetenvToURLConfiguration config
where config.hasFlow(src, sink)
select src, "This environment variable constructs a URL $@.", sink, "here"
```

Classe usada para análise global entre *sink* e *source*

```
/**
 * Global taint analysis. Source is “GetenvSource”. Sink is a call to the constructor of
 * class java.net.URL. */
import semmle.code.java.dataflow.DataFlow

class GetenvToURLConfiguration extends DataFlow::Configuration {
    GetenvToURLConfiguration() { this = "GetenvToURLConfiguration“ }

    override predicate isSource(DataFlow::Node source) { source instanceof GetenvSource }

    override predicate isSink(DataFlow::Node sink) {
        exists(Call call |
               sink.asExpr() = call.getArgument(0) and
               call.getCallee().(Constructor).getDeclaringType()
                                             .hasQualifiedName("java.net", "URL")
        )
}   }
```

Classe que representa o método System.getenv()

```
class System {
    public static Map<String,String> getenv()
    //...
}
```

```
import java

class GetenvSource extends DataFlow::ExprNode {
    GetenvSource() {
        exists(Method m | m = this.asExpr().(MethodAccess).getMethod() |
            m.hasName("getenv") and
            m.getDeclaringType() instanceof TypeSystem
        )
    }
}
```

$$\exists\, m : m\ is\ a\ method, \ldots$$

https://codeql.github.com/docs/ql-language-reference/formulas/

Exemplo de CWE-78

```
class Test {
    public static void main(String[] args) throws Exception{
        // BAD: user input might include special characters such as ampersands
        {
            String latlonCoords = args[1];
            Runtime rt = Runtime.getRuntime();
            Process exec = rt.exec("cmd.exe /C latlon2utm.exe " + latlonCoords);
        }
        // GOOD: use an array of arguments instead of executing a string
        {
            String latlonCoords = args[1];
            Runtime rt = Runtime.getRuntime();
            Process exec = rt.exec(new String[] {
                    "c:\\path\to\latlon2utm.exe",
                    latlonCoords });
        }
    }
}
```

https://cwe.mitre.org/data/definitions/78.html

https://github.com/github/codeql/tree/main/java/ql/src/Security/CWE/CWE-078

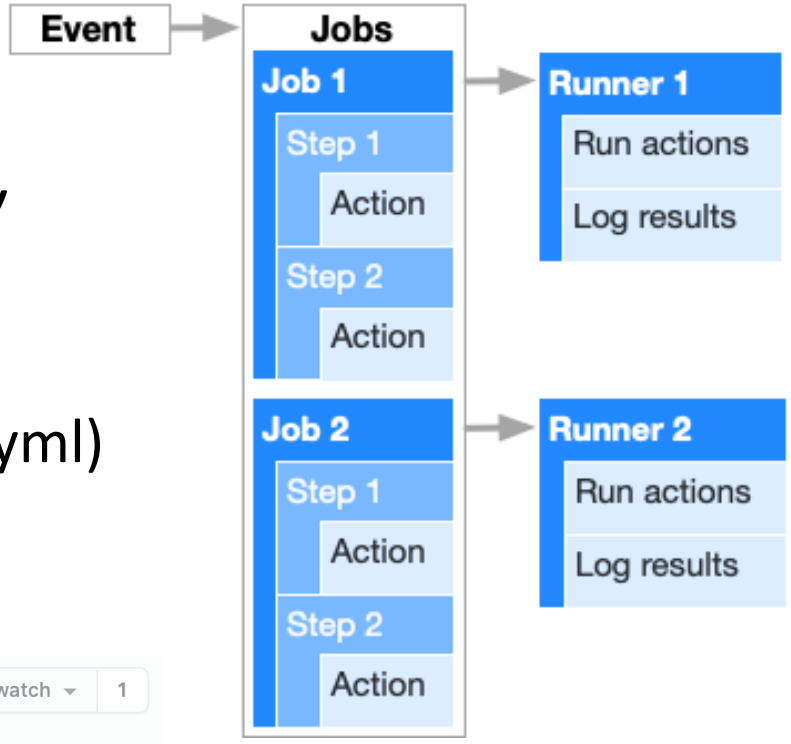
ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Automatização de análise com GitHub Actions

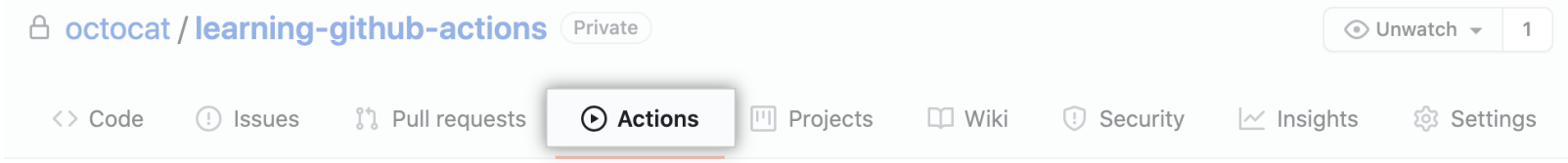
ISEL – Instituto Superior de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro, 1 | 1959-007 Lisboa

Github actions

- As operações realizadas sobre um repositório (push, pull, pull request, …) podem desencadear vários tipos de ações



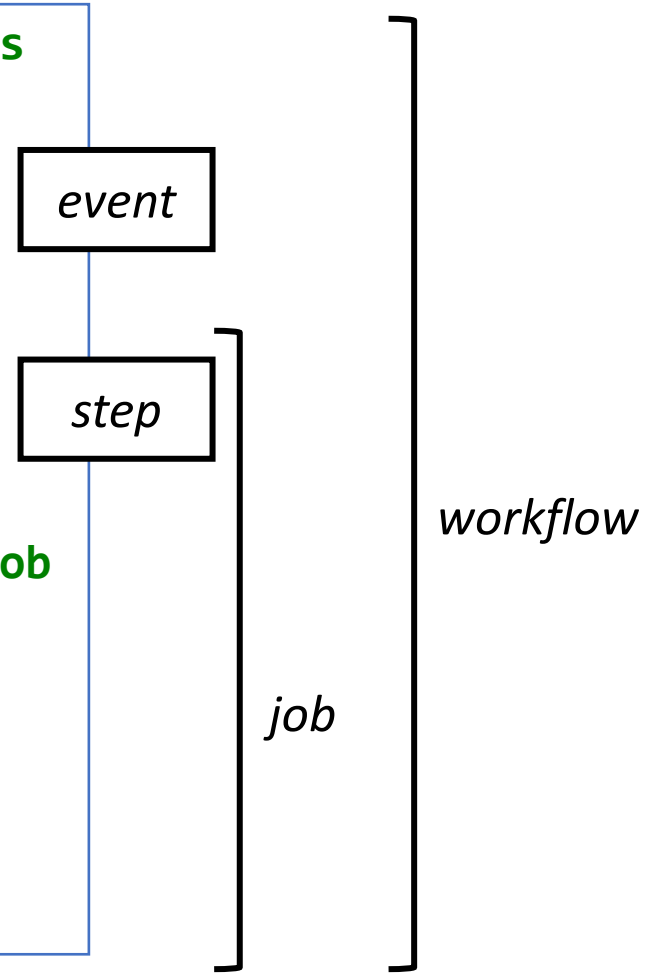
- As ações são agrupadas numa unidade designada *job* o qual faz parte de um *workflow*
  - Cada *job* executada em servidores hospedados pelo GitHub ou externos (*runners*)
  - Descrição das ações é feita em ficheiros de texto (.yml) armazenados no repositório em análise
  - O estado das ações pode ser consultado no site



Github actions – estrutura base

.github/workflows/hello-world.yml

```yaml
# This is a basic workflow to help you get started with Actions
name: Demo workflow
# Controls when the workflow will run
on: [push]

jobs:
  hello-world-job:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest
    # Sequence of tasks that will be executed as part of the job
    steps:
    # Checks-out the repository
    - uses: actions/checkout@v2
    # Runs a single command using the runners shell
    - name: Run a one-line script
      run: echo Hello, world!
```



Github Action para o CodeQL

- Cobertura de vários CWE em Java
  - https://codeql.github.com/codeql-query-help/java-cwe/
  - Código fonte da Action em: https://github.com/github/codeql-action
- *Security -> Code Scanning Alerts -> Set up this Workflow*