# **How to integration project with Katapult**

## **Katapult**

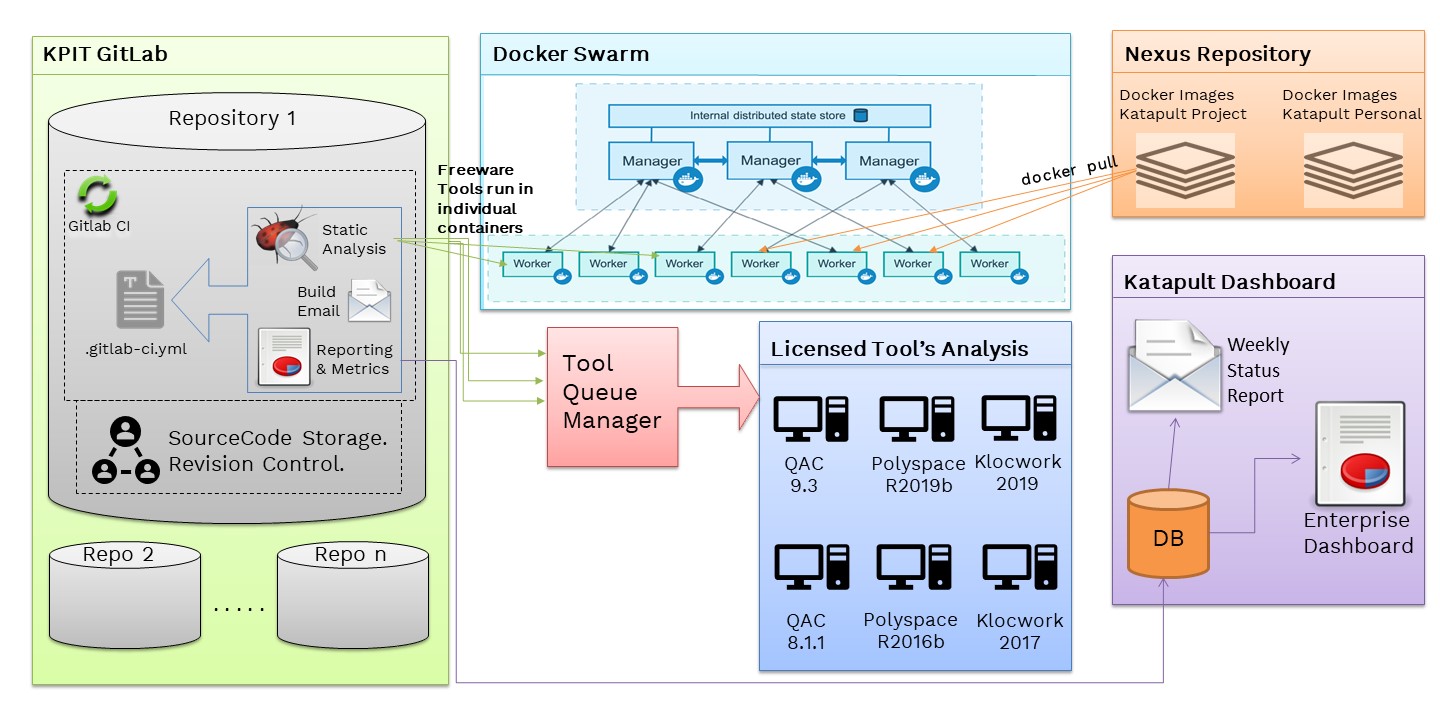
**Katapult - What is Katapult?**

In the context of growth scenario, complex programs and distributed teams, it has become important to provide an infrastructure that will give correct lead indicators and provide enough decision points to make correct judgement on project progress and potential issues that the project will have. The automated infrastructure also addresses scalability with the same infrastructure can now be implemented on many projects quickly without much human efforts.

katapult for Project as an infrastructure ensures we have a vehicle to bring improvements at the project level.

Katapult will help project team in:

* Improving Software Quality
* Increasing Productivity
* Monitoring early warning of broken/incompatible code
* Detecting and fixing integration problems continuously
* Reducing Risk by providing constant feedback on the project status



* As Katapult is fully automated, it will lay the foundation for a repeatable and consistent build process which will immediately notify the project team when there are build failing issues. Having this type of stability in the build process will help to improve the quality of the software indirectly.
* An automated and easily reproducible build process helps in reducing risk; in fact it can help to eliminate potential human errors during a build process.
* The development team will be notified immediately as soon as a broken build is detected which can also reduce delivery risk by improving team's productivity and keeping the project on-track.
* The goal of Katapult is to provide rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected as soon as possible.

There can be following problems in the development phase:

* + **There can be bugs in the code**
  + **Best practices are followed in isolation**
  + **No measurement on the code complexity**
  + **No lead indicators on the code**
  + **Quality check tools are used but not sustained**

Katapult can help to overcome all the above problems.

* Katapult will also help in reducing repetitive manual processes including code compilation, testing, inspection, deployment, and feedback.
* Katapult generates the following quality metrics to guide project management:
  + **Cyclomatic Complexity**
  + **Name of the most complex function and its complexity**
  + **Code Violations**
  + **Percent Duplication in the code**
  + **Code to comment ratio**
* Since Katapult is automated, it only needs to be configured once (and probably tweaked over time), which helps in reducing the time of development team members in few tasks, such as building, running unit tests and analyzing the code.
* The more time that the team can focus on producing quality code and spend less time manually running/ building the code, means the development team is able to produce value to the business
* KPIs supported in Katapult:
  + **Total LOC** or **# of Models**
  + **Code Violation** - Check for code compliance with industry standards (MISRA, CERT, ISO 26262 etc.)
  + **Cyclomatic Complexity** - Derived from McCabe's algorithm, it defines # of nested loops in a functions
  + **Percent Duplication** - No. of duplicated code blocks across the stack
  + **Code to Comment Ratio** - # of commented statement against executable statements
  + **Polyspace Red Checks** - Indicates faulty code causing a run-time error
  + **Polyspace Orange Checks** - Indicates unproven code
  + **Polyspace Grey Checks** - Indicates dead or unreachable code
  + **Unit Test Pass %**
  + **Code Coverage** - Line/Statement Coverage
  + **Software Testing Pass %** - Functional/Integration/System Testing
  + **Requirements Coverage %** - Requirements Proven

### **Technology, Tool and Metric Support**

### A matrix showcasing technologies supported in Katapult for Project and Individuals w.r.t. the tools and respective metrics provided by them

# 

## **Polyspace**

#### **Description**

* Polyspace is a static code analysis tool for large-scale analysis by abstract interpretation to detect, or prove the absence of, certain run-time errors in source code for the C, C++, and Ada programming languages.
* Polyspace tool also checks source code for adherence to appropriate code standards
* Polyspace also checks source code for adherence to MISRA C, MISRA CPP and other related code standards
* Polyspace examines the source code to determine where potential run-time errors such as arithmetic overflow, buffer overrun, division by zero, and others could occur
* Code standards or guidelines such as MISRA C attempt to address code quality, portability and reliability
* To run the Polyspace on your source code, the code must be Polyspace compliance (must compile on Polyspace machine)
* Reports
* Polyspace static analysis consists of following two sections:  
  Polyspace Bug Finder:
* Detect run-time errors, concurrency issues, security vulnerabilities, and other defects in C and C++ embedded software
* It checks compliance with coding rule standards such as MISRA C, MISRA C++, JSF++ and custom naming conventions
* It generates reports consisting of bugs found, code-rule violations, and code quality metrics, including cyclomatic complexity
* It identifies software bugs by performing static program analysis on source code
* It finds defects such as numerical computation, programming, memory, and other errors
* It also produces software metrics such as Comment density of a source file, Cyclomatic complexity, Number of lines, parameters, call levels, etc. in a function, Identified run-time errors in the software
* Polyspace Code Prover:
* It is a sound static analysis tool that proves the absence of overflow, divide-by-zero, out-of-bounds array access, and certain other run-time errors in C and C++ source code
* It produces results without requiring program execution, code instrumentation, or test cases
* It uses semantic analysis and abstract interpretation based on formal methods to verify software interprocedural, control, and data flow behavior
* Verify handwritten or generated code
* Each operation is color-coded to indicate whether it is free of run-time errors, proven to fail, unreachable, or unproven
* It also displays range information for variables and function return values, and can prove which variables exceed specified range limits
* Results can be published to a dashboard to track quality metrics and ensure conformance with software quality objectives
* It annotates source code with a color-coding scheme to indicate the status of each element in the code:
* Green indicates reliable code
* Red indicates faulty code causing a run-time error
* Gray indicates dead or unreachable code
* Orange indicates unproven code
* It uses formal methods-based static code analysis to verify program execution at the language level
* It checks each code instruction by taking into account all possible values of every variable at every point in the code, providing a formal diagnostic for each operation in the code under both normal and abnormal usage conditions
* ace is a static code analysis tool for large-scale analysis by abstract interpretation to detect, or prove the absence of, certain run-time errors in source code for the C, C++, and Ada programming languages.
* Polyspace tool also checks source code for adherence to appropriate code standards
* Polyspace also checks source code for adherence to MISRA C, MISRA CPP and other related code standards
* Polyspace examines the source code to determine where potential run-time errors such as arithmetic overflow, buffer overrun, division by zero, and others could occur
* Code standards or guidelines such as MISRA C attempt to address code quality, portability and reliability
* To run the Polyspace on your source code, the code must be Polyspace compliance (must compile on Polyspace machine)

#### **Reports**

Polyspace static analysis consists of following two sections:  
Polyspace Bug Finder:

* Detect run-time errors, concurrency issues, security vulnerabilities, and other defects in C and C++ embedded software
* It checks compliance with coding rule standards such as MISRA C, MISRA C++, JSF++ and custom naming conventions
* It generates reports consisting of bugs found, code-rule violations, and code quality metrics, including cyclomatic complexity
* It identifies software bugs by performing static program analysis on source code
* It finds defects such as numerical computation, programming, memory, and other errors
* It also produces software metrics such as Comment density of a source file, Cyclomatic complexity, Number of lines, parameters, call levels, etc. in a function, Identified run-time errors in the software

**Polyspace Code Prover**:

* It is a sound static analysis tool that proves the absence of overflow, divide-by-zero, out-of-bounds array access, and certain other run-time errors in C and C++ source code
* It produces results without requiring program execution, code instrumentation, or test cases
* It uses semantic analysis and abstract interpretation based on formal methods to verify software interprocedural, control, and data flow behavior
* Verify handwritten or generated code
* Each operation is color-coded to indicate whether it is free of run-time errors, proven to fail, unreachable, or unproven
* It also displays range information for variables and function return values, and can prove which variables exceed specified range limits
* Results can be published to a dashboard to track quality metrics and ensure conformance with software quality objectives
* It annotates source code with a color-coding scheme to indicate the status of each element in the code:
  + **Green** indicates reliable code
  + **Red** indicates faulty code causing a run-time error
  + **Gray** indicates dead or unreachable code
  + **Orange** indicates unproven code
* It uses formal methods-based static code analysis to verify program execution at the language level
* It checks each code instruction by taking into account all possible values of every variable at every point in the code, providing a formal diagnostic for each operation in the code under both normal and abnormal usage conditions

## **QAC**[**¶**](https://kquest.kpit.com/kquest/projects/katapult/wiki/QAC#QAC)

#### **Description**

* QAC is the static analysis solution for the C language, providing a comprehensive suite of features to help to enforce a wide range of coding standards, and to find bugs in new and legacy code.
* C language proprietary parser, integrated with a sophisticated dataflow engine
* Identifies coding defects at the earlier possible stage in the development cycle
* Dataflow analysis to catch bugs
* Demonstrate compliance to coding rule sets and coding standards, including MISRA C
* Very precise diagnostic information (and extensive help text) for developers to identify defects, analyze the root causes and implement fixes
* Automated code inspection provides instant, object, repeatable code audits
* QAC license is node lock license. So, it cannot be accessed remotely. Even the floating licenses of QAC cannot be accessed remotely
* For more details of QAC refer the: <http://www.programmingresearch.com/static-analysis-software/qac-qacpp-static-analyzers/>
* For FAQs on QAC visit <https://hive.kpit.com/cba/index.php/QAC>

#### **QAC Reports**

**Code Review Report**

* The Code Review Report summarizes metrics and messages from files, functions and classes
* It can also display some code visualizations: includes, calls, relations and function structure
* It gives a broad overview of the code and can be used, for example, in code review meetings

**Compliance Report**

* This report shows how well a project complies with the Coding Standard - the Message Personality
* A Summary is shown on the cover page giving details of the size of the project and violations of the Coding Standard
* Two compliance indices are shown to give an indication of the adherence to the Coding Standard
* Section one shows a compliance matrix similar to the output produced by the Project Warning Summary, showing files and the number of messages in each levels
* Section two highlights the Message Groups with the highest number of messages, while section three shows the worst files. These sections can be used to decide which areas of the code need the greatest attention to make the code more compliant
* Section four shows the compliance index for each file in the project

**Quality Report**

* The Quality Report presents a quality model for files, functions and for C++ classes
* The model is based on Principal Component Analysis of metrics (including message counts and message densities for files and functions). The metrics used in the analysis can be selected. This includes selection of user defined metrics created by secondary analysis, giving the option of creating a weighted metric formula
* A summary of code size, number of items analyzed and number of metrics used is displayed on the title page
* The main body of the report is divided in two sections, described below. If no metrics have been selected for a section it will be omitted

**1. What is the process/ stages involved in Katapult integration for project?**

|  |  |  |
| --- | --- | --- |
| **Stage Name** | **Description** | **Accountability** |
| Project Details | Module creation on Katapult dashboard | Project Team |
| Integration | Integrate Katapult job with project's CI | Katapult Team |
| Reporting | Ensure builds are running and reports are generated | Katapult Team |
| Tracking | Track daily reports and take action to resolve them | Project Team |

## How to check Polyspace report?

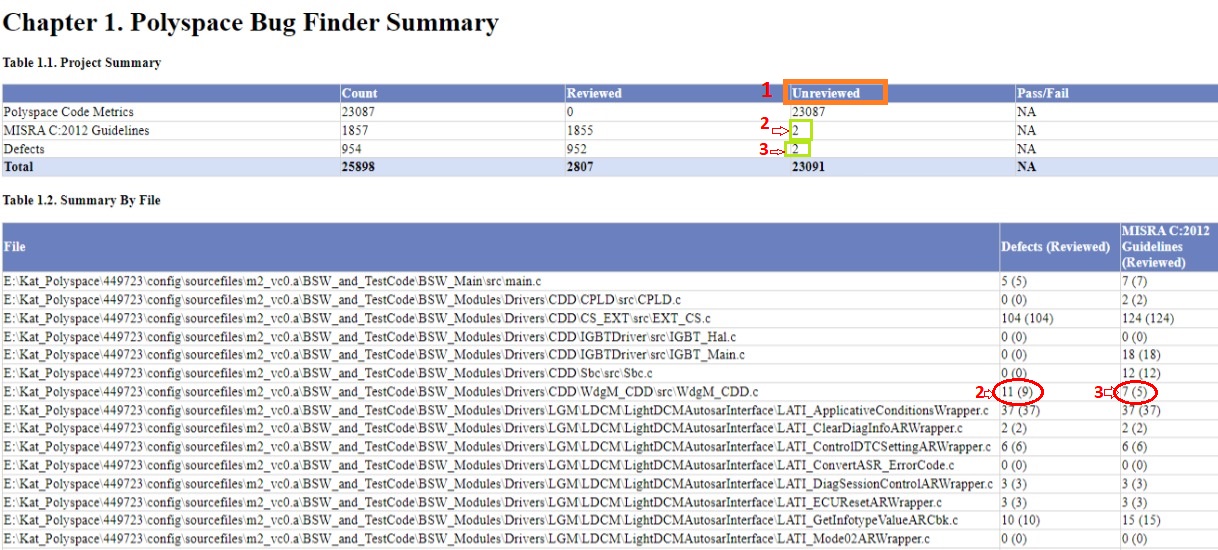
First download polyspace report form stack pipeline.

Go to stack >> CI/CD >> Pipeline>> download latest polyspace report

After download polyspace extract polyspace report…

Open Bugfinder.html file

Look at chapter 1



1. Unreviewed
   1. Mishra c-2012 guidelines -
   2. Defects

If Misra and Defects are 0,0 then no problems in polyspace.

If any other number in unreview column, then find out that file who break the rule. And send