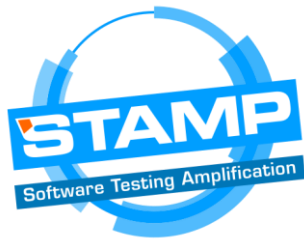




| | |
|-------------------|---|
| Title: | WP2 – D2.4 – Consolidated tool for the configuration amplification, selection and execution |
| Date: | September 30, 2019 |
| Writer: | SINTEF (F. Chauvel, B. Morin, E. Garcia-Ceja) X-Wiki (V. Massol) Engineering (D. Gagliardi) Tellu (S. Morka) |
| Reviewers: | TU Delft (A. Zaidman) INRIA (C. Landry) KTH (B. Baudry) |

Table Of Content

| | | |
|----------|---|-----------|
| 1 | EXECUTIVE SUMMARY | 3 |
| 2 | REVISION HISTORY | 3 |
| 3 | INTRODUCTION | 3 |
| 3.1 | Relation to WP 2 Tasks | 4 |
| 4 | REFERENCES | 4 |
| 5 | ACRONYMS | 5 |
| 6 | CONFIGURATION AMPLIFICATION | 5 |
| 6.1 | Service Orchestrations | 7 |
| 6.2 | Software Stacks | 8 |
| 6.3 | Component Configuration | 9 |
| 6.4 | Realization Actions | 10 |
| 7 | CONFIGURATION SELECTION | 11 |
| 8 | CONFIGURATION EXECUTION | 11 |
| 8.1 | Docker Execution Engine | 11 |
| 8.2 | TestContainers Execution Engine | 13 |
| 8.2.1 | Motivations | 13 |
| 8.2.2 | Architecture | 13 |
| 8.2.3 | Usage | 14 |
| 8.3 | Functional Tests using JUnit | 15 |
| 8.4 | Performance Tests using JMeter | 15 |
| 8.4.1 | Test performance tool choice | 16 |
| 8.4.2 | Integrating JMeter in CAMP | 17 |
| 8.4.3 | Executing JMeter in CAMP | 19 |
| 8.4.4 | A working example to execute JMeter in CAMP | 20 |
| 9 | EXPERIMENTS | 23 |
| 9.1 | XWiki case | 23 |



| | | |
|--------------|--------------------------------------|-----------|
| 9.1.1 | <i>Command line execution</i> | 24 |
| 9.1.2 | <i>IDE execution</i> | 25 |
| 9.1.3 | <i>CI Integration</i> | 25 |
| 9.1.4 | <i>Tested Configurations</i> | 26 |
| 9.1.5 | <i>Migrated Tests</i> | 27 |
| 9.1.6 | <i>Results</i> | 28 |
| 9.2 | The TellU Case-study | 28 |
| 10 | CONCLUSIONS | 32 |

1 Executive Summary

This deliverable D2.4¹ presents CAMP, STAMP's tool for configuration amplification developed by partners involved in WP2. Being a consolidated version of D2.2 and D2.3, this deliverable D2.4 focuses on the enhancements and new features of CAMP for the period M20 – M34, still providing the necessary background information to make D2.4 standalone. Being a demonstrator deliverable, D2.4 adopts a pragmatic approach targeting the users of CAMP.

2 Revision History

| Date | Version | Author | Comments |
|--------------|---------|---|--|
| 20-Aug-2019 | 0.0.1 | Franck Chauvel (SINTEF) | Outline of the deliverable |
| 22-Aug-2019 | | Vincent Massol (X-Wiki) | Section 9 |
| 6-Sept-2019 | | Franck Chauvel, Brice Morin, Enrique Garcia-Ceja (SINTEF) | Sections 3, 6 and 7 |
| 11-Sept-2019 | | Daniele Gagliardi (Engineering) | Section 8 |
| 16-Sept-2019 | | Sverre Morka (Tellu) | Section 9 |
| 24-Sept-2019 | | Brice Morin (SINTEF) | Section 10 + fixes after reviewers' feedback |
| 26-Sept-2019 | | Franck Chauvel (SINTEF) | Fixes after reviewers' feedback |

3 Introduction

Configuration tests, assess the interaction between our code and 3rd party component [1] such as file systems, databases, libraries or services to name a few. Besides the system under test (SUT), integration tests also include its environment, that is all these 3rd party components the SUT interact with, directly or indirectly. They differ from unit tests, which exercise small fragments of code in isolation from this environment.



Configuration tests are however not a silver bullet: they take more time to write and run, and are more brittle than unit tests. Configuring the whole environment, including network, database servers (initialized with users, databases, tables, and possibly data), remote service stubs and the likes remains tedious and error prone. Provisioning it, even when automated with scripts, may take minutes for every run. Worse, integration tests may fail because of changes outside our project: library versions no longer available/compatible or changes in 3rd party services application programming interfaces (API) or behavior.

¹ Latest public version: [STAMP D2.4.pdf](#)



As a result, integration tests remain only used for specific parts of the system, either because the technologies at play make it easy, or because they are likely to bring “return on investment”.

CAMP is a technology-agnostic tool to amplify existing integration tests. Given tests, a template environment as well as a description of the desired variations of the environment, CAMP generates, deploys, executes and collects test reports for all possible environment variants, or a selected sub-set. Variations may alter service orchestrations, software stacks, local dependencies to 3rd party libraries as well as any configuration files. CAMP uses modern constraint-solving techniques to compute all possible environments. As for the test execution, CAMP relies on the Docker platform to quickly provision, execute and decommission the many environment variants.

CAMP is available on Github² under an MIT open-source license. During the M20 – 34, CAMP has been strongly revised and improved, based on the feedback from users. The changes are detailed in Sections 7 – 9. During this period:

- 11 contributors have contributed about 450 commits targeting the CAMP tool itself, documentation or experiments conducted with CAMP.
- 34 issues reported by users were closed
- 21 releases were made available to users for rapid feedback cycles

As of August 21, 2019, CAMP (core tool + tests, excluding all the rest) is about 7,500 LoC mostly written in Python. For comparison, CAMP was about 1,200 LoC written in Python at the beginning of M20.

3.1 Relation to WP 2 Tasks

| Task | Title | Action and related sections |
|-----------------|--|---|
| Task 2.1 | Survey state of practice for configuration specification and automatic configuration planning. | None. See D 2.1 |
| Task 2.2 | Abstract Configuration Model | We refined and improved the model as we show in Section 6 in order to address configuration in service orchestration, software stacks and local component configurations.. |
| Task 2.3 | Automatic Configuration Generation | We improved the CAMP tool according to the feedback of the case study providers. The configuration relies on the Z3 solver as explained in Section 6. |
| Task 2.4 | Configuration assessment and selection | As shown in Section 7, we integrated existing techniques from combinatorial testing to reduce the number of configuration actually tested. CAMP can generate covering arrays of configurations |
| Task 2.5 | Configuration execution and instrumentation | As shown in sections 8.3 and 8.4, we extend CAMP so it can run tests over multiple platform and collect reports that comply with either JUnit4 (functional test) or JMeter (performance tests). |

4 References

- [1] H. K. N. Leung and L. White, "A study of integration testing and software regression at the integration level," in *Proceedings. Conference on Software Maintenance 1990*, 1990, pp. 290-301.
- [2] F. Chauvel, B. Morin, and E. Garcia-Ceja, "Amplifying Integration Tests with CAMP," presented at the The 30th International Symposium on Software Reliability Engineering (ISSRE 2019), Berlin, 2019.

² <https://github.com/STAMP-project/camp>

- [3] C. Atkinson and T. Kühne, "Rearchitecting the UML infrastructure," *ACM Trans. Model. Comput. Simul.*, vol. 12, pp. 290-321, 2002.
- [4] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, pp. 418-421, 2004.

5 Acronyms

| | |
|------|------------------------------------|
| API | Application Programming Interface |
| CI | Continuous Integration |
| EC | European Commission |
| FTP | File Transfer Protocol |
| IDE | Integrated Development Environment |
| JDBC | Java Data Base Connectivity |
| JDK | Java Development Kit |
| JMS | Java Messaging Service |
| JRE | Java Runtime Environment |
| JSON | Java Script Object Notation |
| JTL | JMeter Text Logs |
| HTML | Hyper Text Markup Language |
| HTTP | HyperText transfer protocol |
| LoC | Lines of code |
| MOF | Meta Object Facilities |
| OCL | Object Constraint Language |
| OS | Operating System |
| SUT | System under test |
| UI | User Interface |
| YAML | Yet Another Markup Language |
| XML | eXtensible Markup Language |

6 Configuration Amplification

As shown in Figure 1, CAMP accepts two main inputs: a test template and a variation model. The *test template* is a directory that contains the complete system under test (SUT), which can be deployed on the Docker technology. It includes source code, configuration and deployment scripts, and any piece of information one needs to run the system under test.

The *variation model* is a YAML file that describes selected environment variations and how to modify the template accordingly. Developers manually write them, based on their knowledge of the SUT and of the most likely (or sensitive) environments.

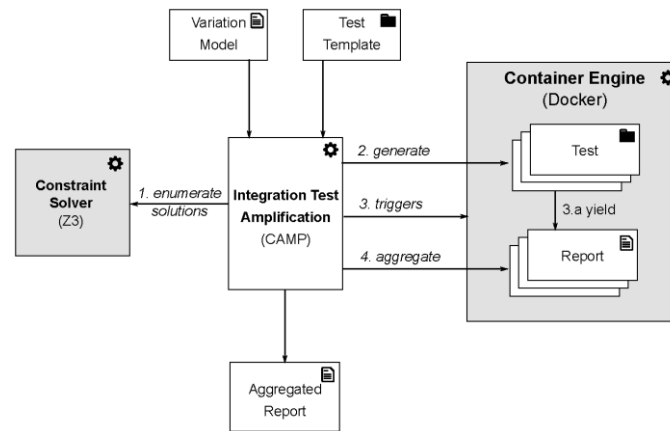


Figure 1 Overview of the CAMP architecture

Equipped with these two pieces of knowledge CAMP first converts the variation model into a constraint satisfaction problem and then uses a solver to obtain all possible configurations. Then, CAMP realizes each solution the solver yielded: it copies the template and then modifies it in order to obtain new “deployable” environments. CAMP then triggers the container engine to run the tests, collects test reports and produces an aggregated report explaining what tests failed in which configurations.

We define an integration test as a test that assesses the interaction of the SUT with at least one of its 3rd party dependencies, be it the file system, the underlying application server, a database or a remote service. For example, Figure 2 illustrates small scale integration tests on a simple web-service. A simple integration test, deployed on a separate node, emits a first request to create a new user, and a second one to retrieve this very user's details, checking for consistency issues. The hashed boxes show all the components that make the environment of this test: the MySQL³ database server and its underlying operating system, the Tomcat servlet container⁴ and the underlying OS and Java runtime environment (JRE), as well as the JRE and OS that support the execution of test itself.

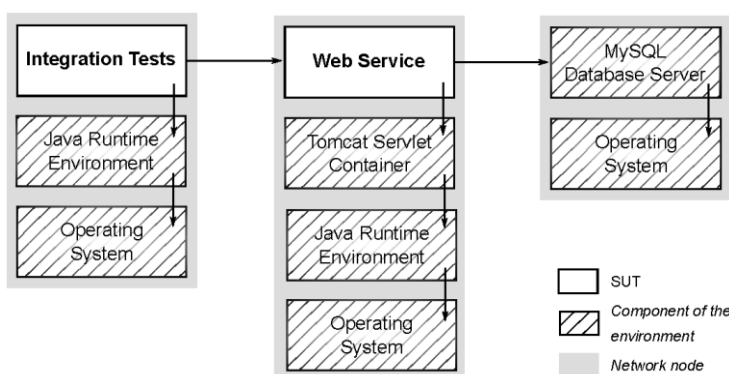


Figure 2 An integration test for a simple Java web service and the many components of its environment (hashed boxes)

This test however only verifies the behaviour of our service in a single specific environment. First, we could modify the configuration of the components in this environment. We could for instance allocate more threads for Tomcat, more or less heap memory for the JRE, etc. Next, we could use different versions of these components, say version 6, 7 or 8 of the Tomcat servlet container. The same holds for

the JRE, the MySQL server and the OS. Finally, we could even use alternative products that offer the same services/functionality:

- Glassfish instead of Tomcat,
- MariaDB⁵ instead of MySQL, or

³ <https://mysql.com>

⁴ <http://tomcat.apache.org/>

⁵ <https://mariadb.org/>



- The OpenJDK⁶ instead of the standard Oracle JDK.

These variations quickly lead to more than 200 possible environments, for which we need to build confidence about the ability of the system to run in those environments. When we deploy remote services, we will eventually have to test alternative environment that account for new bug fixes and security patches. When our services are deployed on our customers' premises, integration tests account for their most likely execution environments. The same holds for libraries and desktop/mobile applications whose execution environment may greatly vary from one installation to the next.

To amplify a test, CAMP generates many variations of the environment where CAMP deploys the SUT, but still executes the same test. Provided we know what this environment consists of, what are its “moving parts” and how these parts may vary, we can then explore the space of possible environments, and in turn, generate alternative “integration tests”. In our web service example, we can generate a new test by switching from one version of MySQL to another (say from MySQL 5.6 to MySQL 8.0). Any change to the SUT's environment yields a new integration test.

We have identified three dimensions along which one can modify the environment in which an integration test executes:

- Changes in the service orchestration;
- changes in the software stack underlying any service;
- changes in the configuration of any component of any software stack.

We describe how we capture these possible changes in the variation model in the following subsections.

6.1 Service Orchestrations

At the coarsest level, software systems include multiple remote services that are deployed on separate network nodes. Each service has a specific API, that is, the operations it offers as well as the data structures it consumes and produces.

Our first variation is therefore to modify how we chain our services together. We can replace a service by any other that complies to the same API, possibly inducing further changes downstream. In our web service example (see Figure 2), we could replace MySQL by MariaDB. This would replace the whole network node where the database service is running and not only the database component running on top of the OS.

⁶ <https://openjdk.java.net/>

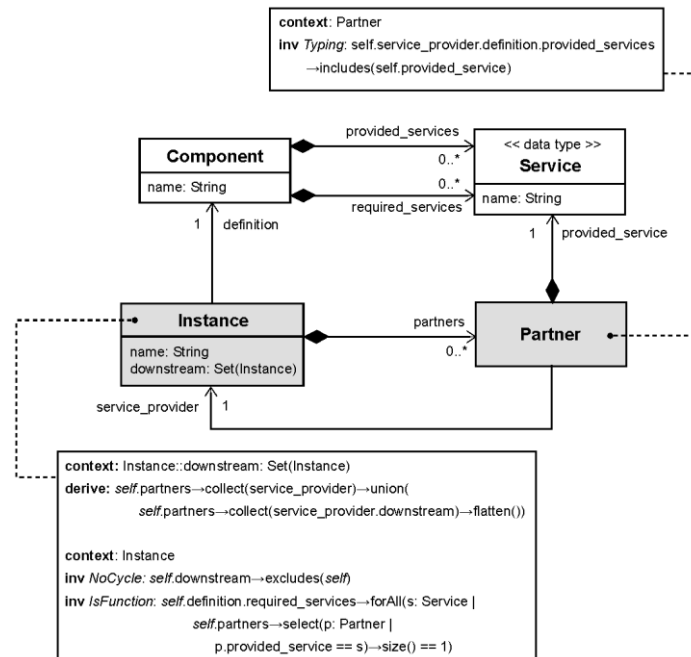


Figure 3 CAMP's model of service orchestrations (MOF class diagram with OCL constraints) reused from [2]

Figure 3 shows how our variation model captures changes in service orchestrations. The SUT environment is made of components, which may offer and require services. We abstracted away services' APIs and we assume that two services whose names match are compatible. From there, we can describe a particular service orchestration as a graph of component instances (grey classes in Figure 3). The Typing invariant restricts connections to instances that provide and require the same service. The *IsFunction* invariant avoids pending services: Each instance has exactly one specific partner defined for every service its definition requires. The *NoCycle* invariant rules out cycles in the graph of dependencies between instances.

6.2 Software Stacks

Services, unless they are compiled into a statically-linked binary, do not run in a vacuum, they run on top of dedicated software stacks that include local dependencies such as application servers, local libraries or 3rd party tools that must be available on the same node. In our variation model, we see these dependencies as other kinds of components that may vary as well.

In our web service example (see Figure 2), we could replace for instance the Tomcat servlet container by a different version or by an alternative, say by Glassfish for example. This would affect the software stack on top of which our web service runs but would preserve the rest of the stack (JRE and OS).

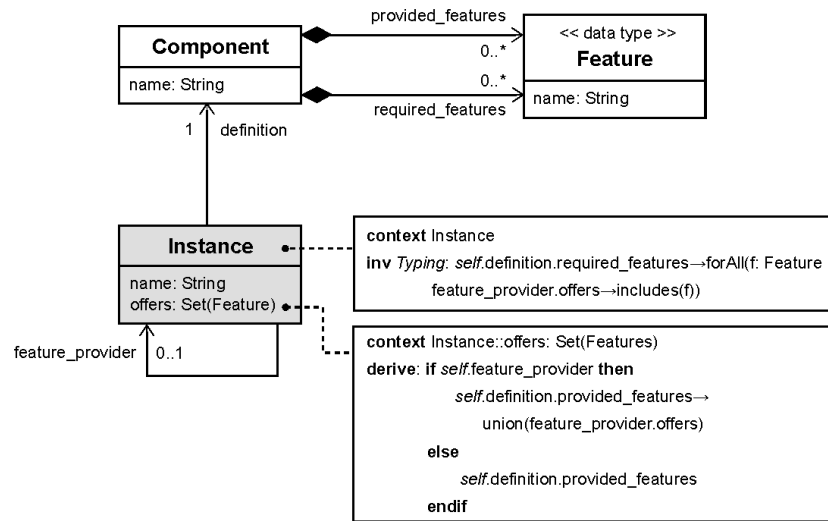


Figure 4 CAMP's model of software stacks (MOF class diagram, with OCL constraints). The Component and Instance classes refer to those shown in Figure 3 (borrowed from [2]).

We model these software stacks using features, as shown in Figure 4. A component (possibly a service, but not necessarily) may offer or demand features. To obtain a valid configuration, one must hence bind every instance whose definition demands specific features to another instance that offers them. The derived property named “offers” defines these offered features the transitive closure of all the features offered by the underlying feature providers, further down in the stack. The Typing constraint ensures components lay on top of stacks that collectively offer all the features they require.

6.3 Component Configuration

At the finest level, the SUT environment may vary because of specific components' configuration options. These may be parameters in configuration files, such as number of threads allocated, database connection details or network configuration, to name a few.

In our web service example (see Figure 2), we would for instance alter the configuration of the Tomcat servlet container and set a different capacity for its thread pool. We could as well modify the number of database connections, the amount of heap memory allocated to the underlying JRE, the garbage collection algorithm, etc.

As shown in Figure 4, we model these variations by adding variables to components. Each variable (whose name must be unique within a component) specifies the values it accepts, either as an enumeration or as a range. We equipped instances with a configuration that maps every variable specified in their definition (i.e., the related component) to its actual value. The *isFunction* invariant ensures every instance has a value defined for every variable its definition carries. The *IsValid* invariant makes sure this value belongs to the domain of the associated variable.

Figure 5 CAMP's model of components' detailed configuration (MOF class diagram with OCL constraints). The Component and Instance classes refer to those already shown in Figure 3 and Figure 4 (borrowed from [2]).

6.4 Realization Actions

Once the solver has produced all possible environments, CAMP realizes them and derives actual configuration files from the given SUT template.

To do so, the user must have specified in the variation model actions associated with decisions made by the constraint solver. In the current version, shown in Figure 6, only variables can be equipped with such actions.

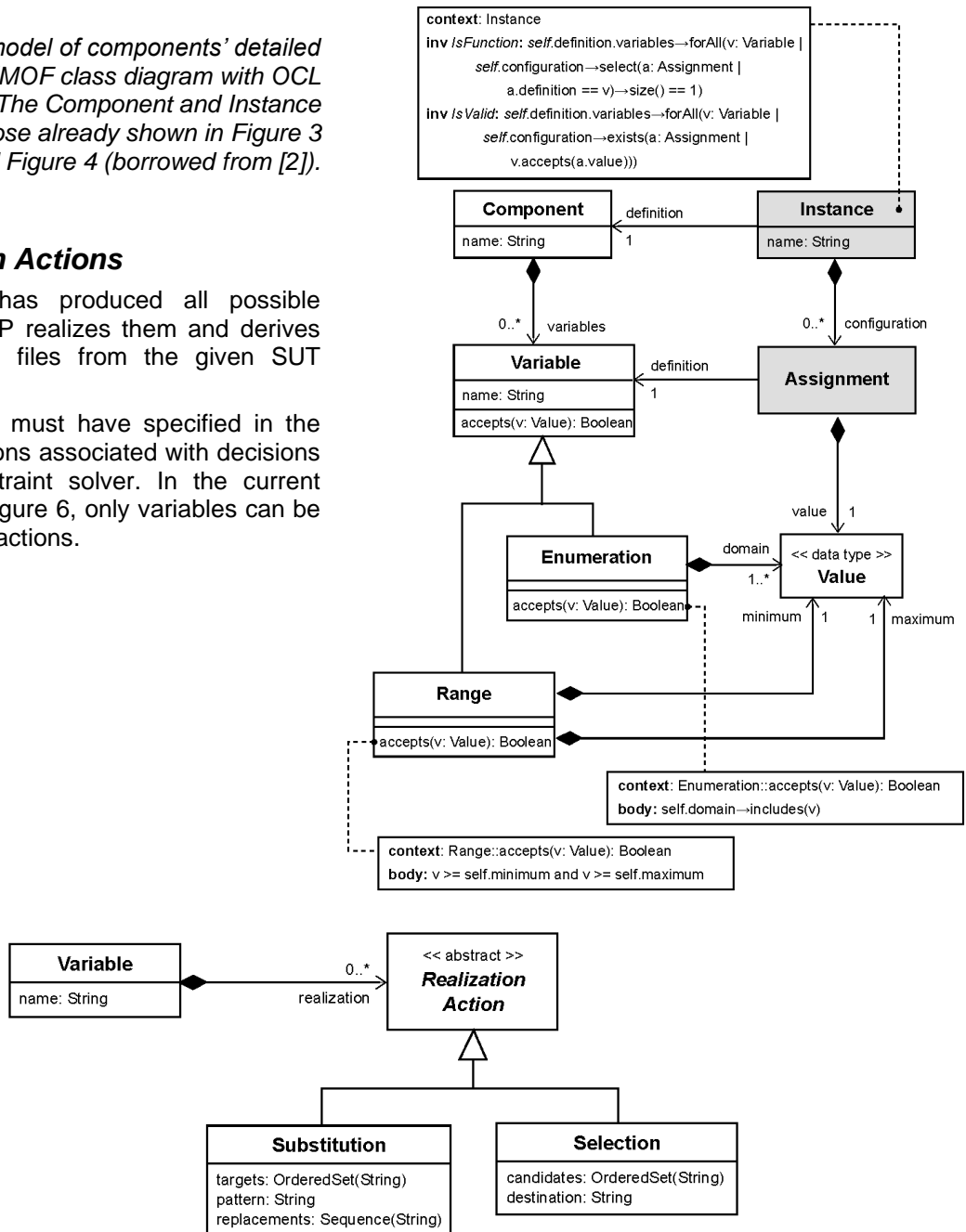


Figure 6 Realization actions to enable the derivation of new environments from the SUT template (borrowed from [2])

We so far support two actions, namely text substitution and resource selection. A text substitution allows CAMP to replace all occurrences of a pattern in a file with a replacement that varies according to the value selected for the variable. A resource selection allows CAMP to choose among alternative files or directories.

In most modern systems, configuration variables are typically described into semi-structured text files, such as YAML, JSON or XML, which makes string substitutions and resource selection effective means to amplify those files.

7 Configuration Selection

Given this variation model that captures all possible variations in the service orchestration, in the software stacks and in the components' configuration (see Figure 3, Figure 4 and Figure 5), CAMP then generates a constraint satisfaction problem and uses a solver to compute all possible environments.

Building all possible environments boils down to finding all possible graphs of instances (objects instantiated from grey classes in Figure 3, Figure 4 and Figure 5) that satisfy all the constraints listed above. Given the "definitions" specified by the user in the variation model (i.e., the white classes in Figure 3, Figure 4 and Figure 5), the constraint solver (cf. Figure 1) attempts to complete the underlying object model by instantiating the grey classes until all constraints are satisfied. The white and grey classes implement a type-instance pattern [3].

In most cases, a few variation points yield a combinatorial explosion of the number of possible environments. Returning to the Java web service example, testing multiple versions of MySQL (say versions 5.6, 5.7 and 8.0), multiple versions of Tomcat (say 7, 8.0, 8.5 and 9.0) as well as multiple versions of the underlying Java runtime environment (JRE, say versions 6, 7 and 8) already results in 36 configurations, though there remain many other parameters of interest, such as thread pool size, database connections pool size, maximum memory allocated to the JRE, etc.

To reduce the number of tests and speed up the feedback to the developers, CAMP applies existing combinatorial testing techniques. CAMP can compute covering arrays [4] and output a smaller number of configurations, where each single option is selected at least once. In the previous example, four configurations suffice to test at least once every version of every component (i.e., MySQL, Tomcat and JRE).

To synthesize covering arrays, CAMP continuously modifies the problem given to the solver. Each time the solver finds a solution, CAMP adjusts the given problem to ensure that the next solution does belong to the covering array. In our Java example, once the solver has found a possible solution (say using MySQL 5.6 and Tomcat 7), CAMP will add a new constraint to the original model in order to rule out any other configurations using both MySQL 5.6 and Tomcat 7.

8 Configuration Execution

8.1 Docker Execution Engine

As shown in Figure 1, CAMP leverages the Docker⁷ container technologies to execute test in multiple environments. Given the variation model introduced in Figure 3, Figure 4 and Figure 5, as well as a deployment template, CAMP copies and then modifies this template to yield new deployable environments.

The deployment template relies on two main ideas: encapsulating services and their underlying stacks into separate container and deploying service orchestrations as networked containers. The Docker platform directly supports these two ideas. On the one hand, Dockerfiles capture how to install and configure the software packages and processes that will be running inside a given container. CAMP thus requires every component of a stack come with a separate Dockerfile. The structure of stacks easily lends itself to Dockerfiles. Each Dockerfile refers to a starting Docker image (specified by the FROM statement) and, when executed, it yields a new Docker image. To modify a software stack initially described in the SUT template, CAMP modifies these FROM statements to impose the alternative stack structures chosen by the constraint solver. In our previous Java example (see Figure 2), Tomcat is running on top of a JRE, running on top of the OS. On the other hand, the docker-compose tool lets us describe how services relate to containers and how they are networked together (e.g., configuring network ports, host names, credentials). In our Java example, docker-compose captures the connection of the test program to the running service, exposing the service on network with the host name used by the client test program.

To execute the tests, CAMP does three things. It first spins off the containers, volumes and network required by the service orchestration. CAMP then triggers the test suites, and eventually collects and aggregates the test reports to show which tests have failed on what configuration. CAMP therefore needs extra information to know what command triggers the test, and how and where to read the test reports. The snippet below shows an excerpt of the variation model (in YAML) that specify this:

⁷ See <https://www.docker.com/>



components:

```
tests:
  provides_services: [ IntegrationTests ]
  requires_services: [ Greetings ]
  implementation:
    docker:
      file: tests/Dockerfile
  tests:
    command: mvn -B test
    reports:
      format: junit
      location: target/surefire-reports
      pattern: .xml
```

As we can see, we extended the description of the "tests" components, with all the details needed for CAMP to execute the test. These include the command to run "mvn -B test", which requests Maven to run the tests. The following "reports" entry explains the format of the test reports (JUnit XML), the location of the test reports (i.e., target/surefire-reports) the default location on Maven projects, and the file extension to filter out other format of reports (e.g., text files). We show below an example of test summary, also CAMP generates a detailed report including the test name, output and stack trace for each configuration. Note that we stripped out some lines of the output for the sake of brevity.

```
$ camp execute -d .
CAMP v0.3.3 (MIT)
Copyright (C) 2017 -- 2019 SINTEF Digital
```

```
Loaded './camp.yaml'.
Loading configurations from './out' ...
```

```
- Executing ./out/config_1
  1. Building images ...
    $ bash build_images.sh (from './out/config_1/images')
  2. Starting Services ...
    $ docker-compose up -d (from './out/config_1')
  3. Running tests ...
    $ docker-compose run tests mvn -B test (from './out/config_1')
  4. Collecting reports ...
    $ docker ps --all --quiet --filter name=config_1_tests_run_1 (from
'./out/config_1')
    $ docker cp f68e14d053a6:/tests/target/surefire-reports ./test-reports
(from './out/config_1')
    Reading TEST-org.samples.GreetingServiceTest.xml
  5. Stopping Services ...
    $ docker-compose down (from './out/config_1')

- Executing ./out/config_2
- Executing ./out/config_3
```

Test SUMMARY:

| Configuration | RUN | PASS | FAIL | ERROR |
|----------------|-----|------|------|-------|
| ./out/config_1 | 1 | 1 | 0 | 0 |

| | | | | |
|----------------|---|---|---|---|
| ./out/config_2 | 1 | 1 | 0 | 0 |
| ./out/config_3 | 1 | 1 | 0 | 0 |
| <hr/> | | | | |
| TOTAL | 3 | 3 | 0 | 0 |

8.2 TestContainers Execution Engine

8.2.1 Motivations

The Docker Execution Engine operates as a black box for executing tests on lots of generated configurations on a server machine. However, there are situations when we would like to be able to execute configuration tests on developers' machines:

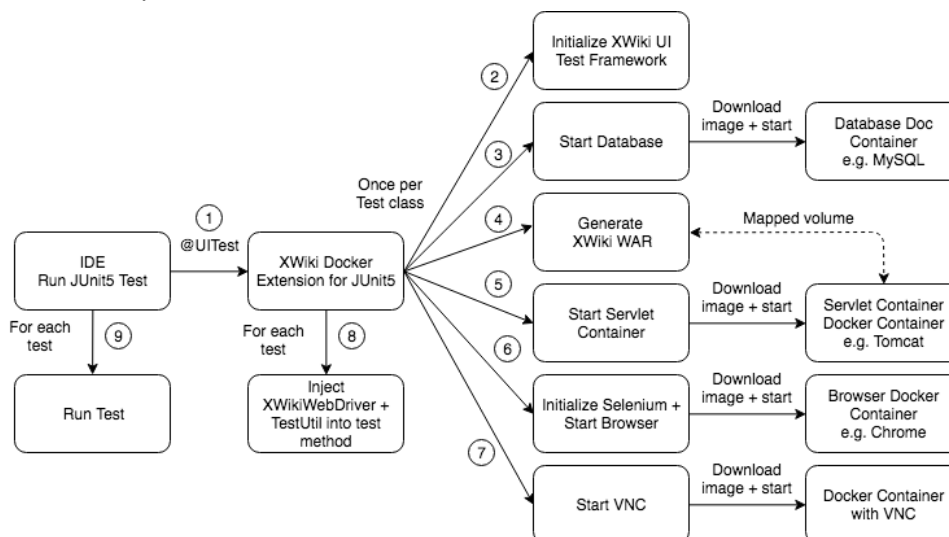
- The developer wants to test what he's developing on various configurations before he can push the new code to ensure that it's working and of good quality;
- A user has reported an issue with a given configuration and the developer needs to reproduce locally and debug the issue;
- The project would like to integrate the execution of configuration tests inside the existing build and CI system in place, as standard JUnit tests, executed by the build (Maven for example).

8.2.2 Architecture

A solution to implement the requirements is to use the TestContainers library which is a library that you use on top of JUnit 4 or JUnit 5. TestContainers is a library to drive Docker by easily allowing to start and stop containers. It performs a lot of interesting background tasks such as waiting for the containers to be started, cleaning up dangling Docker containers, taking screenshots and videos of test executions, supporting Docker in Docker, allowing creation of Docker images on the fly, and more.

It should be noted that the CAMP TestContainers Execution Engine is not a framework/library per se since its usage depends widely of the project using it. However, by taking one usage of it, it is relatively easy to adapt to another use case and some common parts can be reused such as the scaffolding for the JUnit 5 Extension.

As an example, here is the architecture for the XWiki use case.



Explanations:

1. The `@UITest` annotation is just a way to tell JUnit 5 to use our custom Extension when the test starts. It can also be used to provide the configuration for the test. This is useful when executing the test in one's own IDE to test a specific configuration. For example:

```
@UITest(database = Database.MYSQL, databaseTag = "5", servletEngine = ServletEngine.TOMCAT,
```



```
servletEngineTag  
    browser  
    public  
    ...  
= class Browser.CHROMEMenuIT "8"
```

In this example the MenuIT test will be executed on the latest MySQL 5.x, latest Tomcat 8.x and latest Chrome versions. For more information on the possible parameters, please see the reference documentation. Note that in general no parameter should be passed since this is controlled from the build itself by passing Java System Properties.

2. XWiki is using a UI testing framework based on Selenium and step 2 is about initializing it.
3. Start the database that will be used by XWiki for the test
4. Generate a custom XWiki distribution for the test. This distribution is built based on the current Maven POM (pom.xml file). It is a minimum distribution, containing only what is needed for the test to execute. It has several benefits:

- Test the feature in isolation from other features
- Make sure that all module pom.xml files contain the proper dependencies. XWiki is a generic framework for building collaborative web applications and the XWiki users can use it in a wild variety of ways (it is a framework) so the XWiki development team needs to be sure that each module is tested and declares its proper dependencies.
- Makes the distribution lighter and faster to start and execute (although this is slightly balanced by the time it takes to generate the custom distribution).

5. Start the Servlet Container (XWiki is a webapp).
6. Start the Selenium Container provided by TestContainers. The XWiki tests use Selenium/WebDriver to run UI tests.
7. Start the VNC container provided by TestContainers. This container allows connecting to an executing test and view the test in action. It also allows taking screenshots and videos of the test execution.
8. Step 8 is specific to XWiki and uses the JUnit 5 feature of being able to inject test parameters. We inject some utility classes that the test can use to perform its actions.
9. The test is then executed and then stopped.

8.2.3 Usage

Note: The only requirements for running TestContainers-based tests is to have Docker installed locally and to have the user under which you run your IDE and the Maven build be able to use the docker executable.

The usage is very simple since the test is a standard JUnit 5 test and thus can be executed by any JUnit test runner (your IDE, Maven Surefire plugin, etc.).

Here is an example test that gives a flavor of how a test is written.

```
@UITest  
public class MenuIT  
{  
    @Test  
    @Order(1)  
    public void verifyMenuInApplicationsIndex(TestUtils setup)  
    {  
        // Log in as superadmin  
        setup.loginAsSuperAdmin();  
  
        ApplicationIndexHomePage applicationIndexHomePage =  
        ApplicationIndexHomePage.gotoPage();  
  
        assertTrue(applicationIndexHomePage.containsApplication("Menu");  
        ViewPage vp = applicationIndexHomePage.clickApplication("Menu");  
  
        // Verify we're on the right page!  
        assertEquals(MenuHomePage.getSpace(), vp.getMetadataValue("space"));  
        assertEquals(MenuHomePage.getPage(), vp.getMetadataValue("page"));  
    }  
}
```




```
// Now log out to verify that the Menu entry is not displayed for guest users
setup.forceGuestUser();
// Navigate again to the Application Index page to perform the verification
applicationIndexHomePage = ApplicationIndexHomePage.gotoPage();
assertFalse(applicationIndexHomePage.containsApplication("Menu"));
}
}
```

8.3 Functional Tests using JUnit

The most direct way to leverage configuration tests is to run existing functional tests in multiple environments. From a practical standpoint, most of functional tests rely on frameworks such as JUnit⁸, with ease the automation of test runs and report generation. Although not officially recognized by any standardization body, JUnit generates test reports in an XML format whose schema has become the *de facto* standard reused by others testing frameworks. Most CI servers accept it to display graphical summaries about what tests failed.

CAMP also parses this JUnit test report format and therefore collects and aggregates reports from functional tests. Provided the testing framework in use does output test reports in this XML JUnit format, CAMP will collect every test report generated in every configurations and will provide the user with an aggregated view of what test failed in what configuration, including stack traces and system outputs to ease debugging.

As sketched in Section 8.1, one of the component listed in the variation model must include an entry that specifies the command to run the tests, as well as the expected format and location of the test report. In the following snippet for instance, uses the Python green test runner to trigger the tests and will collect XML all files produced in the same directory. At the time of writing, this JUnit format permitted applying CAMP on Java/Maven projects, on Python projects relying on green or pytest, as well as on JavaScript projects.

```
tests:
  command: green --junit-report test_report.xml integration
  reports:
    location: "./"
    pattern: "*.xml"
    format: junit
```

8.4 Performance Tests using JMeter

One of the most interesting usages of configuration testing is the possibility to assess several different configurations, in order to identify performance issues with a specific configuration, or to identify the best configuration from the performance point of view. CAMP generate and realize features together achieve the goal of automatically generating new test configurations, making them available to being executed externally in manual or automated way. Automatic configuration testing is necessarily bound to the specific tool used to automate planned tests, and this means that if you want CAMP to automatically execute these configurations, you have to provide it with a plugin able to.

The CAMP execution module is designed to develop new modules to support specific tools: what it needs is to develop an extension able to read test results and format them accordingly to the CAMP test result template. This template is quite generic, in the sense that it provides developers with a simple container of test results, and each test result contains in turn the test identifier, the result, and, in case of errors, an error code/message/description.

⁸ JUnit was one of the first unit testing framework available and spawn many "clones" in other technologies such as CppUnit for C++, NUnit for C#, unittest for Python, etc.

8.4.1 Test performance tool choice

In the field of performance testing we decided to support Apache JMeter, an open source tool written in Java that can be used to analyze and measure the performance of a variety of applications and services, with a focus on web applications.

JMeter can be used to test also FTP, JDBC, JMS and other widely adopted protocols.

Test cases are written in XML language, but JMeter provides a tester with a nice GUI to ease the process of defining test scripts. Usually, for complex navigation, JMeter can be configured as a proxy, in order to record the navigation done with a web browser. Then the recorded script can be parameterized in order to simulate a variable load of different virtual users interacting with the application under test. Scripts can be enriched with assertion and pre- and post- action for each HTTP call, in order to emulate as realistically as possible the expected load.

Once the script has been parameterized, the execution is done in headless (non-GUI) mode, in order to optimize JMeter HW and SW resources usage, otherwise the test client could become a further bottleneck when measuring performances. When JMeter is executed in headless mode, it generates a JSON file containing information about each single call: the called URL, response time, bytes sent, bytes received, http response codes and possible http error messages. With this JSON file, JMeter also builds a html report containing several statistics:

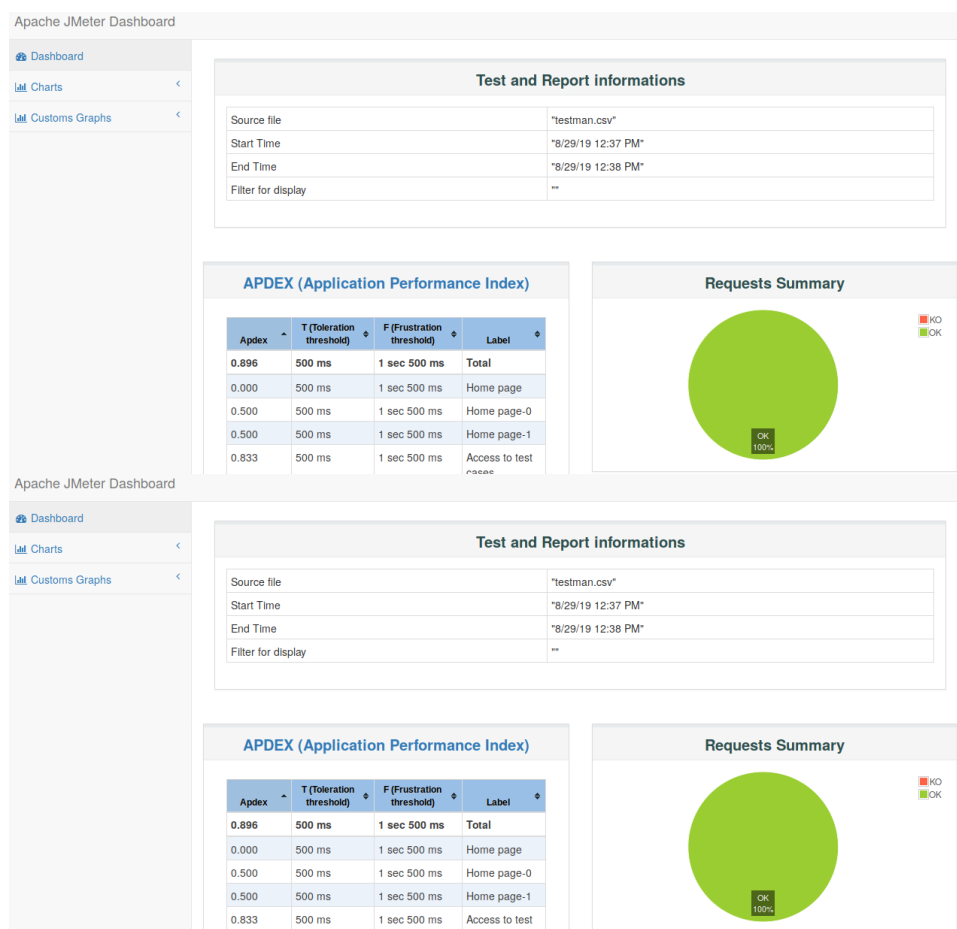


Figure 7 JMeter report performance dashboard

The main page shows a dashboard containing several statistics about error percentage, test duration, etc. On the left side, several charts are available to inspect performance information on each single HTTP call or in an aggregated form:



Figure 8 Distribution of response times across different thresholds

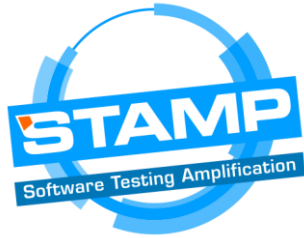
This report is very useful to analyze performance related to a specific configuration under test, but of course it cannot offer information on multiple configurations.

8.4.2 Integrating JMeter in CAMP

Integration of JMeter required to use JMeter result files and use them to build the CAMP performance test result report.

The structure of result JSON file is the following:

```
{
  "Home page-0" : {
    "transaction" : "Home page-0",
```



```
"sampleCount" : 1,  
"errorCount" : 0,  
"errorPct" : 0.0,  
"meanResTime" : 982.0,  
"minResTime" : 982.0,  
"maxResTime" : 982.0,  
"pct1ResTime" : 982.0,  
"pct2ResTime" : 982.0,  
"pct3ResTime" : 982.0,  
"throughput" : 1.0183299389002036,  
"receivedKBytesPerSec" : 0.2744717413441955,  
"sentKBytesPerSec" : 0.3470675280040733  
,  
"Test case creation page" : {  
  "transaction" : "Test case creation page",  
  "sampleCount" : 2,  
  "errorCount" : 0,  
  "errorPct" : 0.0,  
  "meanResTime" : 244.5,  
  "minResTime" : 24.0,  
  "maxResTime" : 465.0,  
  "pct1ResTime" : 465.0,  
  "pct2ResTime" : 465.0,  
  "pct3ResTime" : 465.0,  
  "throughput" : 0.11328877308258752,  
  "receivedKBytesPerSec" : 0.6749200783108644,  
  "sentKBytesPerSec" : 0.05188714314036479  
,  
  "Save new test case" : {  
    "transaction" : "Save new test case",  
    "sampleCount" : 1,  
    "errorCount" : 0,  
    "errorPct" : 0.0,  
    "meanResTime" : 169.0,  
    "minResTime" : 169.0,  
    "maxResTime" : 169.0,  
    "pct1ResTime" : 169.0,  
    "pct2ResTime" : 169.0,  
    "pct3ResTime" : 169.0,  
    "throughput" : 5.9171597633136095,  
    "receivedKBytesPerSec" : 32.24389792899408,  
    "sentKBytesPerSec" : 6.622133875739644
```



```
},  
"Home page-1" : {  
  "transaction" : "Home page-1",
```

For each distinct call, JMeter generates a JSON element containing several data: average, min, max response time, three different percentiles of response times (by default they are 90°, 95° and 99°, but it can be configured with different values) and error percentages.

We developed a Python class able to parse this structure and return a CAMP test report object, following this logic:

- if error percentage is 0 % (no errors—it means no 4xx or 5xx HTTP return codes), it means success;
- if error percentage is 100 % (no errors—it means no 2xx HTTP return codes, i.e. no successful responses), it means error;
- if error percentage is below 100 % (several errors—it means several 2xx HTTP return codes, and several 4xx or 5xx HTTP return codes), it means a failure.

8.4.3 Executing JMeter in CAMP

The CAMP execution result needs to be an aggregated report, but we also wanted to make available also each HTML report per single test configuration.

To execute performance test, we extended the CAMP execute feature.

The first step was to find a JMeter containerization solution. We selected the following image <https://hub.docker.com/r/justb4/jmeter>, available in DockerHub, which is actively maintained, and updated to the last JMeter release (currently: 5.1.1).

The CAMP model (defined in the `camp.yaml` file) contains the configuration to execute performance tests against the SUT.

The following excerpt of the CAMP model shows the performance test execution configuration:

components:

```
tests:  
  provides_services: [ PerformanceTests ]  
  requires_services: [ Testman ]  
  implementation:  
    docker:  
      file: tests/Dockerfile  
  tests:  
    command: -n -t perfctest_script/testman.jmx -l perfctest_script/testman-  
perfctest-report/testman.jtl -e -o perfctest_script/testman-perfctest-report  
    reports:  
      format: jmeter  
      location: perfctest_script/testman-perfctest-report  
      pattern: statistics.json
```

The implementation section specifies the Docker execution engine, providing the relative path to the Docker file containing the description of the Docker image to use to execute the performance tests.

The command to execute tests is the usual command to launch JMeter in headless mode:

```
-n -t perfctest_script/testman.jmx -l perfctest_script/testman-perfctest-  
report/testman.jtl -e -o
```

Now, a short description for each parameter follows:

- -n: headless mode (execute JMeter in "CLI mode")

- -t: name of JMX file containing the test plan
- -l: name of sample JTL file to log sample results
- -e: generate dashboard (HTML) report after load test
- -o: output folder where to store dashboard report

In the output folder also a `statistics.json` file is stored, containing all statistics for each sample. This file is used by CAMP JMeter integration to build the aggregate report (referenced in pattern parameter). In the same folder .jtl files are available to be used, for instance, with the Jenkins JMeter plugin (which is fed by .jtl reports)

8.4.4 A working example to execute JMeter in CAMP

In order to demonstrate the usage of JMeter integration in CAMP, a working example has been developed, and stored within CAMP repository, in this subfolder: <https://github.com/STAMP-project/camp/tree/master/samples/java-web>.

We developed a java Web Application (with RAD tool Spring Roo), implementing a simple test management solution, which exposes these features:

- Authorization password-based: user is asked to provide a userid and password to authenticate
- Record new test case
- List existing test cases
- Edit existing test cases
- Delete existing test case.

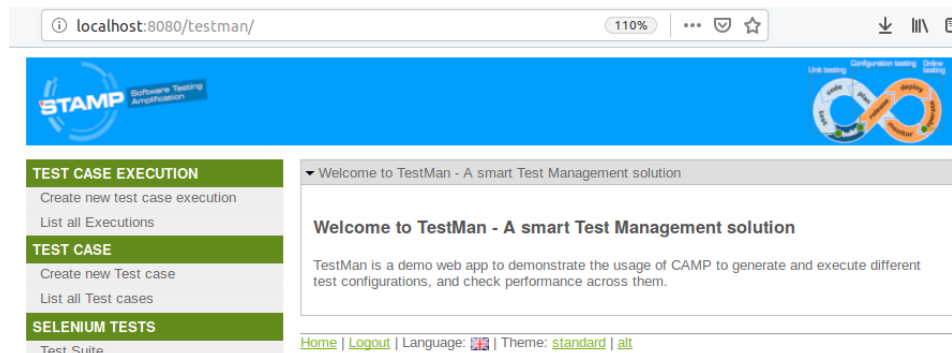


Figure 3: simple web-based test management tool.

We developed a JMeter test plan which authenticates to the application, executes several operations and then logs out from the application:

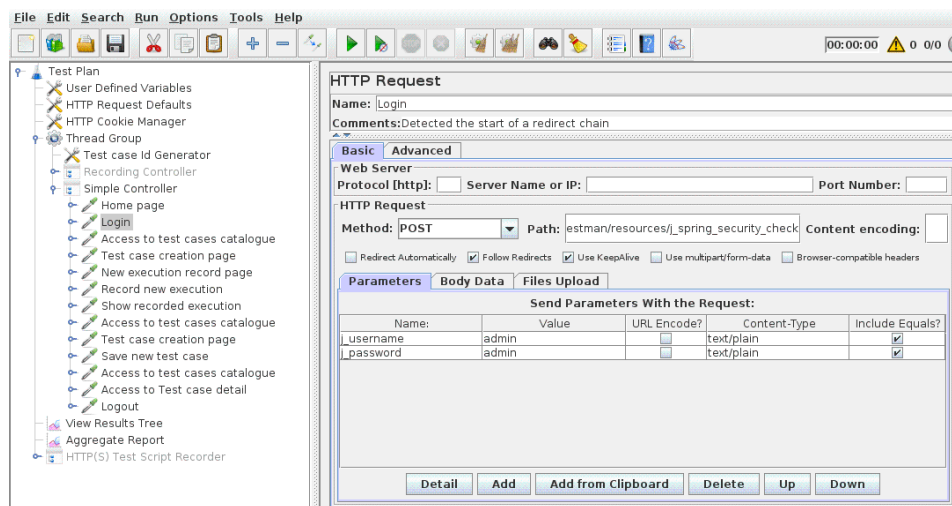


Figure 9 JMeter Test Plan



In this example, CAMP generates three different configurations, and then iterates over them, executing the JMeter test plan, and reporting in the end the aggregate report. What follows is a screenshot taken from the STAMP demo folder, where the example has been executed:

```
stamp@vmi2:~/stamp/camp/samples/java-web$ camp generate -d .
```

```
CAMP v0.6.0 (MIT)
```

```
Copyright (C) 2017 -- 2019 SINTEF Digital
```

```
Loaded './camp.yaml'.
```

```
/home/stamp/stamp/camp/camp/generate.py:214: YAMLLoadWarning: calling yaml.load() without Loader=...  
is deprecated, as the default Loader is unsafe. Please read https://msg.pyyaml.org/load for full  
details.
```

```
metamodel = load_yaml(data)
```

```
- Config. 1 in './out/config_1/configuration.yaml'.
```

```
Includes testman (v7), tests...
```

```
- Config. 2 in './out/config_2/configuration.yaml'.
```

```
Includes tests, testman (v8)...
```

```
- Config. 3 in './out/config_3/configuration.yaml'.
```

```
Includes testman (v9), tests...
```

```
That's all folks!
```

```
stamp@vmi2:~/stamp/camp/samples/java-web$ camp realize -d .
```

```
CAMP v0.6.0 (MIT)
```

```
Copyright (C) 2017 -- 2019 SINTEF Digital
```

```
Loaded './camp.yaml'.
```

```
Loading configurations from './out' ...
```

```
- Built configuration './out/config_3'.
```

```
- Built configuration './out/config_1'.
```

```
- Built configuration './out/config_2'.
```

```
That's all folks!
```

```
stamp@vmi2:~/stamp/camp/samples/java-web$ camp execute -d .
```

```
CAMP v0.6.0 (MIT)
```

```
Copyright (C) 2017 -- 2019 SINTEF Digital
```

```
Loaded './camp.yaml'.
```

```
Loading configurations from './out' ...
```

```
- Executing ./out/config_3
```

```
1. Building images ...
```

```
$ bash build_images.sh (from './out/config_3/images')
```

```
2. Starting Services ...
```

```
$ docker-compose up -d (from './out/config_3')
```

```
3. Running tests ...
```

```
$ docker-compose run tests -n -t perfctest_script/testman.jmx -l perfctest_script/testman.csv -e  
-o perfctest_script/testman-perfctest-report (from './out/config_3')
```

```
4. Collecting reports ...
```



```
$ docker ps --all --quiet --filter name=config_3_tests_run_ (from './out/config_3')
$ docker cp 022df1f79327:/tests/perftest_script/testman-perftest-report ./test-reports (from
'./out/config_3')
Reading statistics.json
5. Stopping Services ...
$ docker-compose down --volumes --rmi all (from './out/config_3')

- Executing ./out/config_1
  1. Building images ...
    $ bash build_images.sh (from './out/config_1/images')
  2. Starting Services ...
    $ docker-compose up -d (from './out/config_1')
  3. Running tests ...
    $ docker-compose run tests -n -t perftest_script/testman.jmx -l perftest_script/testman.csv -e
-o perftest_script/testman-perftest-report (from './out/config_1')
  4. Collecting reports ...
    $ docker ps --all --quiet --filter name=config_1_tests_run_ (from './out/config_1')
    $ docker cp cc001d4a4889:/tests/perftest_script/testman-perftest-report ./test-reports (from
'./out/config_1')
    Reading statistics.json
  5. Stopping Services ...
    $ docker-compose down --volumes --rmi all (from './out/config_1')

- Executing ./out/config_2
  1. Building images ...
    $ bash build_images.sh (from './out/config_2/images')
  2. Starting Services ...
    $ docker-compose up -d (from './out/config_2')
  3. Running tests ...
    $ docker-compose run tests -n -t perftest_script/testman.jmx -l perftest_script/testman.csv -e
-o perftest_script/testman-perftest-report (from './out/config_2')
  4. Collecting reports ...
    $ docker ps --all --quiet --filter name=config_2_tests_run_ (from './out/config_2')
    $ docker cp a6a972e1e7bc:/tests/perftest_script/testman-perftest-report ./test-reports (from
'./out/config_2')
    Reading statistics.json
  5. Stopping Services ...
    $ docker-compose down --volumes --rmi all (from './out/config_2')
```

Test SUMMARY:

| Configuration | RUN | PASS | FAIL | ERROR |
|----------------|-----|------|------|-------|
| ----- | | | | |
| ./out/config_3 | 22 | 22 | 0 | 0 |
| ./out/config_1 | 22 | 22 | 0 | 0 |
| ./out/config_2 | 22 | 22 | 0 | 0 |
| ----- | | | | |
| TOTAL | 66 | 66 | 0 | 0 |

That's all folks!

stamp@vmi2:~/stamp/camp/samples/java-web\$



Test Summary reports that 22 tests passed for each configuration: it means that all requests completed successfully for each sample (for each URL called by JMeter).

This example has been fully documented in the CAMP documentation: <https://stamp-project.github.io/camp/pages/execute.html>

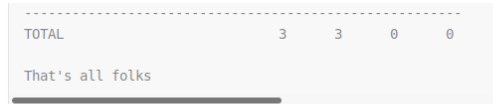
CAMP

Amplify your testing with more configurations!

[View the Project on GitHub](#)
 STAMP-project/camp

This project is maintained by STAMP-project

Hosted on GitHub Pages — Theme by [orderedlist](#)

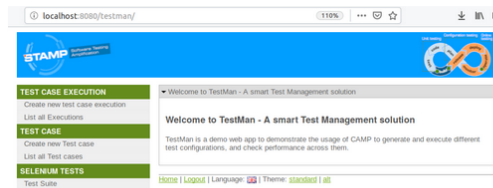


Example: Performance Test A Java WebApp

In this example you will see how to use CAMP to generate new configurations, to be performance tested with [Apache JMeter](#).

If you are unsure how to generate new configuration with CAMP, please check the `camp generate` and `camp realize` commands.

Let us consider a [Java WebApp](#) as a running example of CAMP execute. We called it "Testman": It's a simple Test Management tool web-based, with a form-based authentication, written using [Spring Roo](#):



Here is an overview of our project structure:

Figure 10 Online documentation on performance test with CAMP

9 Experiments

9.1 XWiki case

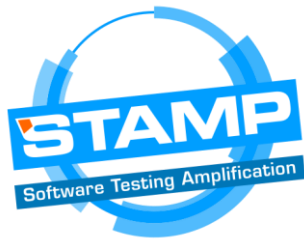
The XWiki project has a fixed number of configurations it supports:

- For [databases](#):

| Database Name | Version |
|---------------|--|
| HSQLDB | Latest |
| MySQL | Latest of <code>oldstable</code> Debian repository, latest of <code>stable</code> Debian repository, latest of <code>unstable</code> Debian repository |
| PostgreSQL | Latest of <code>oldstable</code> Debian repository, latest of <code>stable</code> Debian repository, latest of <code>unstable</code> Debian repository |
| Oracle | Latest of 12.x |

- For [Servlet Containers](#):

| Servlet Container Name | Version |
|------------------------------------|--|
| Tomcat | Latest of <code>oldstable</code> Debian repository for Tomcat 8 and latest of <code>stable</code> Debian repository, latest of <code>testing</code> Debian repository for Tomcat 9 |
| Jetty (XWiki Standalone packaging) | Latest of 9.x |



| | |
|----------------------------|--|
| Jetty (Official packaging) | Latest of <code>oldstable</code> Debian repository, latest of <code>stable</code> Debian repository, latest of <code>unstable</code> Debian repository for Jetty 9 |
|----------------------------|--|

- For [browsers](#):

| Browser Name | Version |
|--------------|-----------------------|
| IE | Latest stable IE11 |
| Edge | Latest stable Edge |
| Firefox | Latest stable Firefox |
| Chrome | Latest stable Chrome |
| Safari | Latest stable Safari |

Thus, it does not make sense to use the CAMP configuration generator to automatically generate lots of configuration since these configurations will not be in the supported list. Since the configuration list is fixed and not too large, a better approach is to describe them explicitly and test all of them. This is why the XWiki project has chosen to use the TestContainers Execution engine with a fixed list of configurations.

9.1.1 Command line execution

To run a functional test for a single configuration, navigate to the Maven module (e.g. `xwiki-platform-menu-test-docker`) and pass the configuration details as system properties. For example to execute the MenuIT test on MySQL 5.7.x, Tomcat 8.5.x, and latest Chrome you'd start Maven with the following command:

```
mvn clean install -Dxwiki.test.ui.database=mysql -Dxwiki.test.ui.databaseTag=5.7  
-Dxwiki.test.ui.servletEngine=tomcat -Dxwiki.test.ui.servletEngineTag=8.5 -  
Dxwiki.test.ui.browser=chrome
```

You will get the following logs in the console (excerpted for clarity):

```
[INFO] -----  
[INFO]  T E S T S  
[INFO] -----  
[INFO] Running org.xwiki.menu.test.ui.MenuIT  
10:14:56.999 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - XWiki is not started, starting all...  
10:14:57.001 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Starting database [MYSQL]...  
10:15:12.120 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Building custom XWiki WAR...  
...  
10:15:37.745 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Starting Servlet container  
[TOMCAT]...  
10:16:16.917 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Provision extensions for test...  
...  
10:18:05.054 [main] INFO  o.x.t.d.i.j.b.BrowserContainerExecutor - (*) Starting browser [CHROME]...  
...  
10:18:26.826 [main] INFO  o.x.t.d.i.j.b.BrowserContainerExecutor - VNC server address:  
[vnc://vnc:secret@localhost:32854]  
10:18:27.046 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Initialize Test Context...  
10:18:45.992 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Start VNC container...  
10:18:52.417 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Starting test  
[verifyMenuInApplicationsIndex]  
10:19:12.559 [main] INFO  o.x.t.d.i.j.XWikiDockerExtension - (*) Stopping test  
[verifyMenuInApplicationsIndex]  
10:19:12.863 [main] INFO  o.x.t.d.i.j.junit5.DockerTestUtils - Screenshot for test  
[verifyMenuInApplicationsIndex] saved at [(...)/xwiki-platform/xwiki-platform-core/xwiki-platform-  
menu/xwiki-platform-menu-test/xwiki-platform-menu-test-docker/./target/mysql-5.7-default-tomcat-8.5-  
chrome/screenshots/mysql-5.7-default-tomcat-8.5-chrome-org.xwiki.menu.test.ui.MenuIT-  
verifyMenuInApplicationsIndex.png].
```



```
10:19:13.136 [main] INFO o.x.t.d.i.j.XWikiDockerExtension - (*) VNC recording of test has been
saved to [./target/mysql-5.7-default-tomcat-8.5-chrome/screenshots/mysql-5.7-default-tomcat-8.5-
chrome-org.xwiki.menu.test.ui.MenuIT-verifyMenuInApplicationsIndex.flv]
```

```
...
```

```
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 313.518 s - in
org.xwiki.menu.test.ui.MenuIT
```

9.1.2 IDE execution

It is also possible to run a given configuration in the IDE and to put debugging breakpoints in the test or in the code under test. That makes it very easy to debug configuration-related issues.

You do this by specifying parameters to the `@UITest` annotation. For example:

```
@UITest(database = Database.MYSQL, databaseTag = "5.7", servletEngine = ServletEngine.TOMCAT,
servletEngineTag = "8.5",
browser = Browser.CHROME)
public class MenuIT
...
```

9.1.3 CI Integration

Section 9.2 described the TestContainers Execution engine for CAMP. It explained how to execute a test for a single configuration. Now we needed a place where to describe all the configurations and to loop over them and use Maven and the Maven Failsafe plugin to run JUnit on them. We chose to do this in Jenkins, using a Pipeline.

We also realized that it would take too long to execute all the configurations every time there is a commit on the XWiki SCM. Thus, we use the scheduling feature of Jenkins to schedule 3 type of jobs:

- `docker-latest` job type: executes all the test on the latest configuration for each element (latest MySQL, latest PostgreSQL, latest Tomcat, latest Firefox, etc). This job is executed once per day.
- `docker-all` job type: executes only a few selected tests (smoke tests) but on all supported configurations. This job is executed once every week.
- `docker-unsupported` job type: executes only a few selected tests but on currently unsupported configurations. They are configurations that are either not working yet or not stable enough to be supported but that we would like to support in the future. This allows us to see the remaining work before we can support them officially. This job is executed once every month.

The configurations are defined in a custom Pipeline step named `dockerConfigurations` which returns an array of configurations and that you use like this:

```
def dockerConfigurationList = dockerConfigurations(type)
```

Where `type` is the job type (`docker-latest`, `docker-all` or `docker-unsupported`).

Then we have another custom step named `xwikiDockerBuild` which executes Maven N times with the passed configuration list. The Pipeline part looks like this:

```
private void buildDocker(type)
{
    def dockerConfigurationList
    def dockerModuleList
    def customJobProperties
    node() {
        // Checkout platform to find all docker configurations and test modules so that we can then
        // parallelize executions of configs and modules across Jenkins agents.
        checkout skipChangeLog: true, scm: scm
        dockerConfigurationList = dockerConfigurations(type)
        if (type == 'docker-unsupported') {
            dockerModuleList = ['xwiki-platform-core/xwiki-platform-menu']
        }
    }
}
```



```

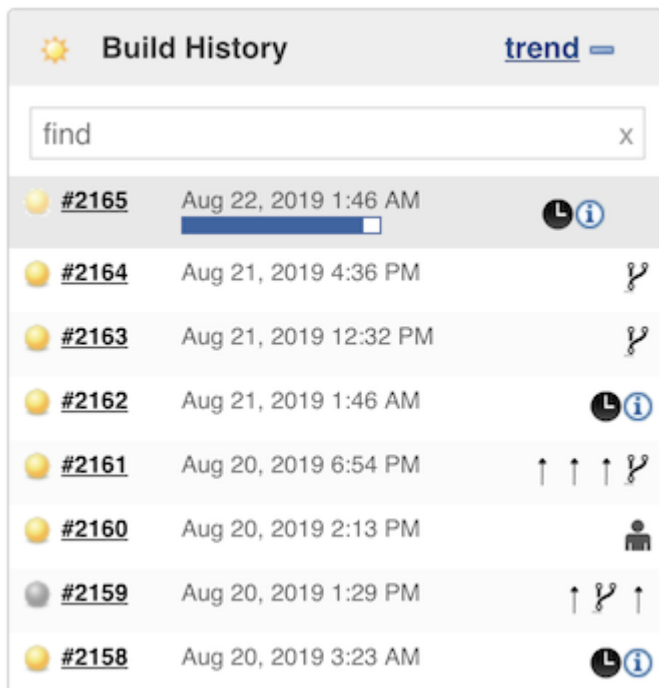
    } else {
        dockerModuleList = dockerModules()
    }
    customJobProperties = getCustomJobProperties()
}

xwikiDockerBuild {
    configurations = dockerConfigurationList
    modules = dockerModuleList
    // Make sure that we don't reset the job properties!
    jobProperties = customJobProperties
}
}

```

Note that one very tricky aspect was finding how to integrate these scheduled jobs as part of the main pipeline for building XWiki (the pipeline that executes the XWiki build whenever a commit is done). In other words, finding out how to add scheduled job execution as part of the Jenkinsfile and mix regular execution with scheduled execution. The technical solution is documented at <http://massol.myxwiki.org/xwiki/bin/view/Blog/ScheduledJenkinsfile>.

The following screenshot shows the intermixed execution of standard pipeline execution with the docker scheduled execution (displayed with the clock and information icons).



9.1.4 Tested Configurations

The following configurations have been defined and are used:

For docker-latest:

| | Database | Servlet Engine | Browser |
|-----------------|----------------------------------|----------------|---------|
| Configuration 1 | MySQL 5.7.x (JDBC driver 5.1.45) | Tomcat 8.5.x | Chrome |



| | | | |
|-----------------|---|------------------|---------|
| Configuration 2 | PostgreSQL 11.x (JDBC driver 42.2.5) | Jetty 9.2.x | Chrome |
| Configuration 3 | HSQLDB Embedded | Jetty Standalone | Firefox |
| Configuration 4 | Oracle 11g Release 2 (JDBC driver 12.2.0.1) | Tomcat 8.5.x | Firefox |

For docker-all:

| | Database | Servlet Engine | Browser |
|------------------|---------------------------------------|------------------|---------|
| Configuration 5 | MySQL 5.7.x (JDBC driver 5.1.45) | Tomcat 8.x | Chrome |
| Configuration 6 | MySQL 5.5.x (JDBC driver 5.1.45) | Tomcat 8.x | Firefox |
| Configuration 7 | PostgreSQL 11.x (JDBC driver 42.2.5) | Jetty 9.x | Chrome |
| Configuration 8 | PostgreSQL 9.4.x (JDBC driver 42.2.5) | Jetty 9.x | Firefox |
| Configuration 9 | PostgreSQL 9.6.x (JDBC driver 42.2.5) | Jetty 9.x | Chrome |
| Configuration 10 | HSQLDB Embedded | Jetty Standalone | Firefox |

For docker-unsupported:

| | Database | Servlet Engine | Browser |
|------------------|--|-----------------------|---------|
| Configuration 11 | MySQL 5.7.x with utf8mb4 encoding (JDBC driver 5.1.45) | Tomcat 8.x | Chrome |
| Configuration 12 | MySQL 5.7.x (JDBC driver 5.1.45) | Tomcat 8.x on Java 11 | Firefox |

Note that we currently do not have configurations for IE, Edge and Safari since there are no Docker containers for them that would run on linux. Also, TestContainers support for Windows is very primitive at the moment. This is not too much of a problem since the XWiki UI test do not test for layout issues and thus the browser used will not have much impact in practice.

9.1.5 Migrated Tests

The XWiki functional UI tests have to be converted to the new TestContainer-based setup before they can be executed on the various configurations using Docker. This is a long and slow process. The XWiki project has voted the usage of the new framework for any new test but old tests need to be ported. The strategy is to do so at least when we need to bring changes to existing tests or when we find flaky tests.

At the date of this writing 20 functional test classes have been ported (totaling 43 tests). Note that functional UI tests are very large tests since they are scenarios (unlike unit tests) and they are written usually as a single test to not occur the long fixture time.

| | |
|-------------------------|-----|
| Test classes migrated | 20 |
| Tests migrated | 43 |
| Test executions overall | 516 |

The test executions correspond to $43 * \text{number of configurations} = 43 * 12 = 516$ executions;
 More precisely, when comparing from before that's $516 - 43 = 473$ more tests being executed.



9.1.6 Results

It is difficult to gauge how well the configuration testing is working because we are now testing with various configurations when developing and thus, we have less configuration-related bugs in the XWiki releases and thus less JIRA issues created.

We can compare the number of configuration-related JIRA issues between several periods:

- Between 2016-12-31 (start of STAMP) to 2018-06-30. June 2018 is about the time when we started having TestContainers-based configuration tests
- Between 2018-06-30 to 2019-08-22 (date of this writing)

| Period | Configuration bugs | Bugs per month |
|--------------------------|--------------------|----------------|
| 2016-12-31 to 2018-06-30 | 55 | 3/month |
| 2018-06-30 to 2019-08-22 | 20 | 1.4/month |

In conclusion, it seems the strategy is working since we have more than halved the number of configuration-related bugs thanks to configuration testing. There is still progress to do to convert more of our existing functional tests to TestContainers-based tests and possibly in creating new and more exotic configurations that would allow us to catch less obvious configuration issues.

9.2 The TellU Case-study

TelluCloud is architected a set of Kubernetes services. The Kubernetes specifications are YAML files, similar to the snippet below (for the "Actions" service):

kind: Deployment

...

spec:

replicas: 1 # CAMP will amplify this value with [1,2,3]

...

template:

spec:

...

containers:

- name: actions

image: tellu/actions:latest

env:

...

- name: JAVA_TOOL_OPTIONS

value: "-XX:MaxMetaspaceSize=300m -XX:MaxHeapSize=100m"

resources:

requests:

cpu: 50m

memory: 200Mi

limits:

cpu: 200m # CAMP will amplify this value with [50,200]

memory: 500Mi # CAMP will amplify this value with [200,400,600]

...

...

...

...

The YAML-files provided were on the following components:

| Component | Description |
|-------------|--|
| Actions | Performs or delegates most of the actions, such as create alarms, send mail sms etc. In TelluCloud |
| Filterstore | Filters incoming observations and stores them to history database |
| Cassandra | Stores data to the cassandra instance |

For now, we have tested the Actions component, where the number of replicas for this service, as well as the limits for the CPU and RAM will be amplified by CAMP. To test this Actions component, we have added a function in another component that will trigger 1000 Alarm Action messages to the queue for the component.

10.5.1 Tests

The last line in the YAML files were repeated 4 times in the bottom. This was due to a regression in a previous version of CAMP, which is now corrected in the latest version of CAMP.

```

      timeoutSeconds: 2
timeoutSeconds: 2
2
timeoutSeconds: 2

```

We have tested by deploying the provided YAML files for Actions in one of our test-environments. We also performed the tests on the default configuration for Actions. For each configuration, that was successfully deployed, we've generated 1000 alarms and calculated execution efficiency by the means we have available in the cluster.

These are:

Grafana (measured in intervals. Not very accurate):

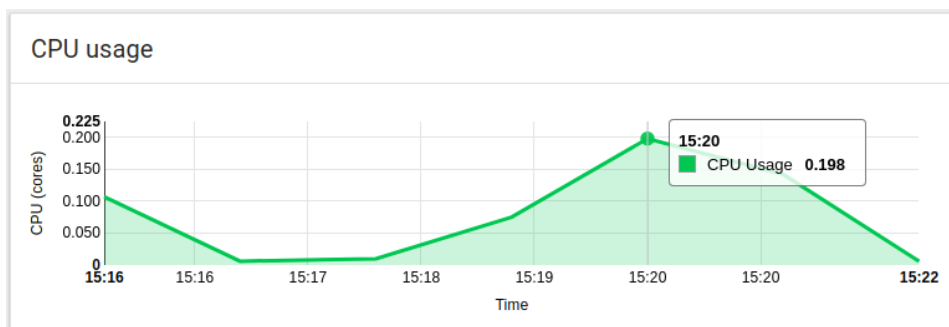
- **Queue Peak:** The highest recorded messages in the queue simultaneously
- **Throughput Peak (TP):** Throughput is the number of messages consumed per second by the component
- **Actions Throughput Peak (AT):** How long messages stay in the queue average
- **Total Time:** The measured time in Grafana from when the queue started growing until it reports all messages consumed. (This is probably a poor measure. It registers in a window of 15, so if there is a message in the queue at the end of the last second of the first window and then one in the first second of the last window 30 seconds will be added. Not very accurate)

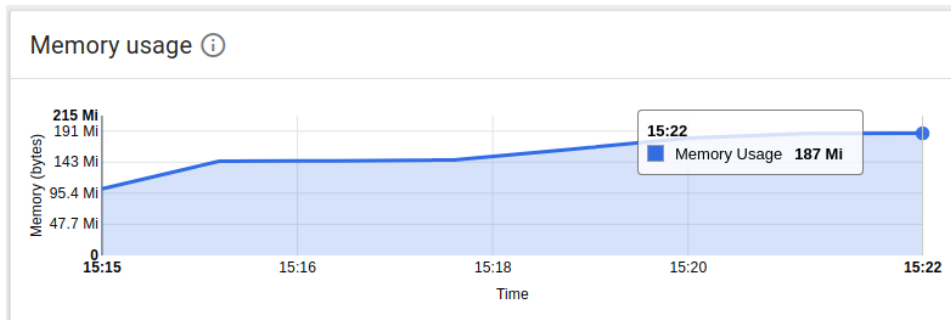
TelluCloud:

- **Time Difference:** The difference in time by comparing timestamp of the first alarm created and the last alarm created.

Kubernetes:

- **CPU Usage peak:** As read for the deployment in kubernetes
- **Memory Usage peak:** As read for the deployment in kubernetes



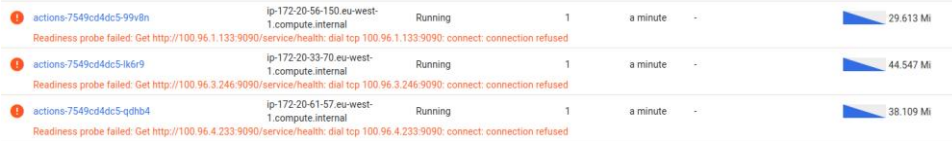


The table below show the configurations that have been generated by CAMP and tested by Tellu. Configurations are identified with a triplet [a, b, c], where:

- a: number of replicas
- b: CPU limit
- c: RAM limit

| Config. | Queue Peak | AT / TP | Time Difference | Total Time | CPU / Memory | Other Observations |
|-----------------------|------------------------------|-------------|-----------------|------------|--------------|---|
| Tellu | 917 | 18.5 | 93.99699 | 105 | | |
| Config 1 [1,50,300] | Failed to load configuration | | | | | |
| Config 2 [3,200,500] | 1000 | 9.01 | 75.62 | 90 | | |
| Config 3 [5,350,700] | 895 | 9.62 / 11.8 | 31.44 | 60 | | |
| Config 4 [1,500,900] | 999 | 9.70 / 7.4 | 44.16 | 75 | | |
| Config 5 [5,50,300] | Failed to load configuration | | | | | |
| Config 6 [1,50,900] | Failed to load configuration | | | | | |
| Config 7 [1,50,700] | Failed to load configuration | | | | | |
| Config 8 [1,50,500] | Failed to load configuration | | | | | |
| Config 9 [5,50,700] | Failed to load configuration | | | | | |
| Config 10 [5,50,900] | Failed to load configuration | | | | | |
| Config 11 [5,500,300] | 0 | 9.83 / 18.4 | 18.94 | - | | This configuration did not start consuming messages before pods had been restarted. Also it created 1028 alarms and not 1000. Measures done here are done in a second run after everything was ok. Queue Peak of 0 is probably because of grafana frequency |
| Config 12 [5,200,300] | 943 | 9.53 / 17.8 | 56.37 | 75 | | |
| Config 13 [5,350,300] | 825 | 8.2 / 13.7 | 35.31 | 60 | | |

| Config 14 [3,500,300] | 534 | 10.15 / 18 | 29.42 | 30 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|---|-------------|----------|-----------|---------------|--|------|------|--------|----------|-----|-------------|----------------|--------------------------|---|---------|---|-----------|-------|-----------|--|--|--|--|--|--|--|--------------------------|---|---------|---|-----------|-------|-----------|--|--|--|--|--|--|--|--------------------------|--|---------|---|-----------|-------|-----------|--|--|--|--|--|--|--|--------------------------|--|---------|---|----------|-------|-----------|--|--|--|--|--|--|--|--------------------------|--|---------|---|----------|-------|-----------|--|--|--|--|--|--|--|
| Config 15 [3,350,300] | 819 | 9.92 / 17.5 | 44.43 | 60 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 16 [1,350,300] | 900 | 10.66 / 22 | 63 | 75 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 17 [3,350,500] | 926 | 9.22 / 16.4 | 43.74 | 45 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 18 [3,500,500] | 888 | 8.31 / 27.3 | 28.52 | 45 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 19 [5,500,500] | 623 | 9.40 / 19.1 | 31.97 | 45 | 0.735 / 915mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 20 [1,500,500] | 828 | 9.36 / 25.2 | 51.07 | 60 | 0.33 / 205mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 21 [5,200,500] | 858 | 9.62 / 15.8 | 58.02 | 75 | 0.822 / 947mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 22 [5,350,500] | 715 | 9.96 / 21.1 | 33.07 | 60 | 0.810 / 863mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 23 [1,350,500] | 921 | 9.58 / 21 | 60.56 | 75 | 0.326 / 178mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 24 [1,200,500] | 998 | 8.01 / 13.8 | 119.43 | 135 | 0.198 / 184mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 25 [3,50,500] | <p>Failed to load configuration</p> <table> <thead> <tr> <th>Name</th><th>Node</th><th>Status</th><th>Restarts</th><th>Age</th><th>CPU (cores)</th><th>Memory (bytes)</th></tr> </thead> <tbody> <tr> <td>actions-65bc869b45-4vzpz</td><td>ip-172-20-61-57.eu-west-1.compute.internal</td><td>Running</td><td>3</td><td>5 minutes</td><td>0.048</td><td>63.730 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.4.6:9090/service/health: dial tcp 100.96.4.6:9090: connect: connection refused</td></tr> <tr> <td>actions-65bc869b45-vlqm9</td><td>ip-172-20-33-53.eu-west-1.compute.internal</td><td>Running</td><td>3</td><td>5 minutes</td><td>0.048</td><td>58.617 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.2.111:9090/service/health: dial tcp 100.96.2.111:9090: connect: connection refused</td></tr> <tr> <td>actions-65bc869b45-w8zk7</td><td>ip-172-20-33-70.eu-west-1.compute.internal</td><td>Running</td><td>3</td><td>5 minutes</td><td>0.048</td><td>62.242 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.3.216:9090/service/health: dial tcp 100.96.3.216:9090: connect: connection refused</td></tr> </tbody> </table> | | | | | | Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | actions-65bc869b45-4vzpz | ip-172-20-61-57.eu-west-1.compute.internal | Running | 3 | 5 minutes | 0.048 | 63.730 Mi | Readiness probe failed: Get http://100.96.4.6:9090/service/health: dial tcp 100.96.4.6:9090: connect: connection refused | | | | | | | actions-65bc869b45-vlqm9 | ip-172-20-33-53.eu-west-1.compute.internal | Running | 3 | 5 minutes | 0.048 | 58.617 Mi | Readiness probe failed: Get http://100.96.2.111:9090/service/health: dial tcp 100.96.2.111:9090: connect: connection refused | | | | | | | actions-65bc869b45-w8zk7 | ip-172-20-33-70.eu-west-1.compute.internal | Running | 3 | 5 minutes | 0.048 | 62.242 Mi | Readiness probe failed: Get http://100.96.3.216:9090/service/health: dial tcp 100.96.3.216:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-4vzpz | ip-172-20-61-57.eu-west-1.compute.internal | Running | 3 | 5 minutes | 0.048 | 63.730 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.4.6:9090/service/health: dial tcp 100.96.4.6:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-vlqm9 | ip-172-20-33-53.eu-west-1.compute.internal | Running | 3 | 5 minutes | 0.048 | 58.617 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.2.111:9090/service/health: dial tcp 100.96.2.111:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-w8zk7 | ip-172-20-33-70.eu-west-1.compute.internal | Running | 3 | 5 minutes | 0.048 | 62.242 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.3.216:9090/service/health: dial tcp 100.96.3.216:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 26 [5,50,500] | <p>Failed to load configuration</p> <table> <thead> <tr> <th>Name</th><th>Node</th><th>Status</th><th>Restarts</th><th>Age</th><th>CPU (cores)</th><th>Memory (bytes)</th></tr> </thead> <tbody> <tr> <td>actions-65bc869b45-7dr25</td><td>ip-172-20-56-150.eu-west-1.compute.internal</td><td>Running</td><td>0</td><td>a minute</td><td>0.048</td><td>86.590 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.1.133:9090/service/health: dial tcp 100.96.1.133:9090: connect: connection refused</td></tr> <tr> <td>actions-65bc869b45-qdbvl</td><td>ip-172-20-61-57.eu-west-1.compute.internal</td><td>Running</td><td>0</td><td>a minute</td><td>0.049</td><td>60.066 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.4.54:9090/service/health: dial tcp 100.96.4.54:9090: connect: connection refused</td></tr> <tr> <td>actions-65bc869b45-appzx</td><td>ip-172-20-33-70.eu-west-1.compute.internal</td><td>Running</td><td>0</td><td>a minute</td><td>0.049</td><td>55.852 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.3.32:9090/service/health: dial tcp 100.96.3.32:9090: connect: connection refused</td></tr> <tr> <td>actions-65bc869b45-t424s</td><td>ip-172-20-33-53.eu-west-1.compute.internal</td><td>Running</td><td>0</td><td>a minute</td><td>0.048</td><td>59.719 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.2.112:9090/service/health: dial tcp 100.96.2.112:9090: connect: connection refused</td></tr> <tr> <td>actions-65bc869b45-7zfg4</td><td>ip-172-20-33-70.eu-west-1.compute.internal</td><td>Running</td><td>0</td><td>a minute</td><td>0.049</td><td>54.133 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.3.31:9090/service/health: dial tcp 100.96.3.31:9090: connect: connection refused</td></tr> </tbody> </table> | | | | | | Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | actions-65bc869b45-7dr25 | ip-172-20-56-150.eu-west-1.compute.internal | Running | 0 | a minute | 0.048 | 86.590 Mi | Readiness probe failed: Get http://100.96.1.133:9090/service/health: dial tcp 100.96.1.133:9090: connect: connection refused | | | | | | | actions-65bc869b45-qdbvl | ip-172-20-61-57.eu-west-1.compute.internal | Running | 0 | a minute | 0.049 | 60.066 Mi | Readiness probe failed: Get http://100.96.4.54:9090/service/health: dial tcp 100.96.4.54:9090: connect: connection refused | | | | | | | actions-65bc869b45-appzx | ip-172-20-33-70.eu-west-1.compute.internal | Running | 0 | a minute | 0.049 | 55.852 Mi | Readiness probe failed: Get http://100.96.3.32:9090/service/health: dial tcp 100.96.3.32:9090: connect: connection refused | | | | | | | actions-65bc869b45-t424s | ip-172-20-33-53.eu-west-1.compute.internal | Running | 0 | a minute | 0.048 | 59.719 Mi | Readiness probe failed: Get http://100.96.2.112:9090/service/health: dial tcp 100.96.2.112:9090: connect: connection refused | | | | | | | actions-65bc869b45-7zfg4 | ip-172-20-33-70.eu-west-1.compute.internal | Running | 0 | a minute | 0.049 | 54.133 Mi | Readiness probe failed: Get http://100.96.3.31:9090/service/health: dial tcp 100.96.3.31:9090: connect: connection refused | | | | | | |
| Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-7dr25 | ip-172-20-56-150.eu-west-1.compute.internal | Running | 0 | a minute | 0.048 | 86.590 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.1.133:9090/service/health: dial tcp 100.96.1.133:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-qdbvl | ip-172-20-61-57.eu-west-1.compute.internal | Running | 0 | a minute | 0.049 | 60.066 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.4.54:9090/service/health: dial tcp 100.96.4.54:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-appzx | ip-172-20-33-70.eu-west-1.compute.internal | Running | 0 | a minute | 0.049 | 55.852 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.3.32:9090/service/health: dial tcp 100.96.3.32:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-t424s | ip-172-20-33-53.eu-west-1.compute.internal | Running | 0 | a minute | 0.048 | 59.719 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.2.112:9090/service/health: dial tcp 100.96.2.112:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-65bc869b45-7zfg4 | ip-172-20-33-70.eu-west-1.compute.internal | Running | 0 | a minute | 0.049 | 54.133 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.3.31:9090/service/health: dial tcp 100.96.3.31:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 27 [3,50,900] | <p>Failed to load configuration</p> <table> <thead> <tr> <th>Name</th><th>Node</th><th>Status</th><th>Restarts</th><th>Age</th><th>CPU (cores)</th><th>Memory (bytes)</th></tr> </thead> <tbody> <tr> <td>actions-5ccc57d854-7dthb</td><td>ip-172-20-61-57.eu-west-1.compute.internal</td><td>Running</td><td>1</td><td>a minute</td><td>0.048</td><td>65.438 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.4.67:9090/service/health: dial tcp 100.96.4.67:9090: connect: connection refused</td></tr> <tr> <td>actions-5ccc57d854-4cmj</td><td>ip-172-20-56-150.eu-west-1.compute.internal</td><td>Running</td><td>1</td><td>a minute</td><td>0.048</td><td>62.090 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.1.149:9090/service/health: dial tcp 100.96.1.149:9090: connect: connection refused</td></tr> <tr> <td>actions-5ccc57d854-t49xc</td><td>ip-172-20-33-70.eu-west-1.compute.internal</td><td>Running</td><td>1</td><td>a minute</td><td>0.047</td><td>69.789 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.3.48:9090/service/health: dial tcp 100.96.3.48:9090: connect: connection refused</td></tr> </tbody> </table> | | | | | | Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | actions-5ccc57d854-7dthb | ip-172-20-61-57.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 65.438 Mi | Readiness probe failed: Get http://100.96.4.67:9090/service/health: dial tcp 100.96.4.67:9090: connect: connection refused | | | | | | | actions-5ccc57d854-4cmj | ip-172-20-56-150.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 62.090 Mi | Readiness probe failed: Get http://100.96.1.149:9090/service/health: dial tcp 100.96.1.149:9090: connect: connection refused | | | | | | | actions-5ccc57d854-t49xc | ip-172-20-33-70.eu-west-1.compute.internal | Running | 1 | a minute | 0.047 | 69.789 Mi | Readiness probe failed: Get http://100.96.3.48:9090/service/health: dial tcp 100.96.3.48:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-5ccc57d854-7dthb | ip-172-20-61-57.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 65.438 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.4.67:9090/service/health: dial tcp 100.96.4.67:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-5ccc57d854-4cmj | ip-172-20-56-150.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 62.090 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.1.149:9090/service/health: dial tcp 100.96.1.149:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-5ccc57d854-t49xc | ip-172-20-33-70.eu-west-1.compute.internal | Running | 1 | a minute | 0.047 | 69.789 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.3.48:9090/service/health: dial tcp 100.96.3.48:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 28 [3,50,300] | <p>Failed to load configuration</p> <table> <thead> <tr> <th>Name</th><th>Node</th><th>Status</th><th>Restarts</th><th>Age</th><th>CPU (cores)</th><th>Memory (bytes)</th></tr> </thead> <tbody> <tr> <td>actions-6b56b77c7f-gkvh2</td><td>ip-172-20-56-150.eu-west-1.compute.internal</td><td>Running</td><td>1</td><td>a minute</td><td>0.048</td><td>65.488 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.1.160:9090/service/health: dial tcp 100.96.1.160:9090: connect: connection refused</td></tr> <tr> <td>actions-6b56b77c7f-j75vw</td><td>ip-172-20-33-53.eu-west-1.compute.internal</td><td>Running</td><td>1</td><td>a minute</td><td>0.049</td><td>95.223 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.2.113:9090/service/health: dial tcp 100.96.2.113:9090: connect: connection refused</td></tr> <tr> <td>actions-6b56b77c7f-jxsvn</td><td>ip-172-20-61-57.eu-west-1.compute.internal</td><td>Running</td><td>1</td><td>a minute</td><td>0.048</td><td>60.180 Mi</td></tr> <tr> <td colspan="7">Readiness probe failed: Get http://100.96.4.76:9090/service/health: dial tcp 100.96.4.76:9090: connect: connection refused</td></tr> </tbody> </table> | | | | | | Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | actions-6b56b77c7f-gkvh2 | ip-172-20-56-150.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 65.488 Mi | Readiness probe failed: Get http://100.96.1.160:9090/service/health: dial tcp 100.96.1.160:9090: connect: connection refused | | | | | | | actions-6b56b77c7f-j75vw | ip-172-20-33-53.eu-west-1.compute.internal | Running | 1 | a minute | 0.049 | 95.223 Mi | Readiness probe failed: Get http://100.96.2.113:9090/service/health: dial tcp 100.96.2.113:9090: connect: connection refused | | | | | | | actions-6b56b77c7f-jxsvn | ip-172-20-61-57.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 60.180 Mi | Readiness probe failed: Get http://100.96.4.76:9090/service/health: dial tcp 100.96.4.76:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Name | Node | Status | Restarts | Age | CPU (cores) | Memory (bytes) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-6b56b77c7f-gkvh2 | ip-172-20-56-150.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 65.488 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.1.160:9090/service/health: dial tcp 100.96.1.160:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-6b56b77c7f-j75vw | ip-172-20-33-53.eu-west-1.compute.internal | Running | 1 | a minute | 0.049 | 95.223 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.2.113:9090/service/health: dial tcp 100.96.2.113:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| actions-6b56b77c7f-jxsvn | ip-172-20-61-57.eu-west-1.compute.internal | Running | 1 | a minute | 0.048 | 60.180 Mi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Readiness probe failed: Get http://100.96.4.76:9090/service/health: dial tcp 100.96.4.76:9090: connect: connection refused | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Config 29 [5,200,900] | 459 | 10 / 30.2 | 32.15 | 30 | 0.423 / 869mi | Didn't process messages before a restart of the pods. 1046 alarms made, despite 1000 being | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | | | | |
|--------------------------|--|-----------------|--------|-----|------------------|---|
| | | | | | | ordered. Jira created. Measures are from fresh run. |
| Config 30 [5,200,700] | 965 | 9.08 / 12.1 | 58.74 | 75 | 0.813 / 855mi | |
| Config 31 [3,500,900] | 688 | 7.72 / 13.7 | 34.63 | 60 | 0.425 / 597mi | |
| Config 32 [3,500,700] | 731 | 11.07 / 35.4 | 29.96 | 60 | 0.557 / 558mi | |
| Config 33 [1,500,700] | 994 | 11.43 / 27.9 | 44.44 | 75 | 0.259 / 182mi | |
| Config 34 [1,350,900] | 994 | 9.11 / 19.7 | 67.73 | 90 | 0.221 / 215mi | |
| Config 35 [1,350,700] | 998 | 9.32 / 19.6 | 66.62 | 90 | 0.210 / 183mi | |
| Config 36 [1,200,900] | 1000 | 8.43 / 13.4 | 122.09 | 135 | 0.196 / 180mi | |
| Config 37 [3,350,900] | 730 | 9.18 / 13.5 | 40.52 | 60 | 0.531 / 666mi | |
| Config 38 [3,50,700] | <p>Failed to load configuration</p>  | | | | | |
| Config 39 [5,500,700] | 264 | 6.68 / 12.6 | 22.28 | 30 | 0.867 / 913mi | |
| Config 40 [5,350,900] | 1000 | 10.64 / 15.5 | 33.17 | 60 | 0.912 / 866mi | One of the pods took 2 minutes to come up, including a restart. |
| Config 41 [5,500,900] | 806 | 6.21 / 15.9 | 27.88 | 30 | 0.970 / 874mi | |
| Config 42 [3,200,300] | 890 | 9.10 / 15.2 | 73.68 | 75 | 0.398 / 537mi | |
| Config 43 [1,200,700] | 1000 | 8.0 / 13.0 | 126.81 | 150 | 0.197 / 185mi | |
| Config 44 [3,200,900] | 1000 | 9.37 / 12.6 | 75.78 | 105 | 0.572 / 547mi | |
| Config 45 [3,200,700] | 1000 | 8.71 / 11.3 | 74.93 | 90 | 0.556 / 567mi | |
| Config 46 [3,350,700] | 987 | 10.45 / 19.3 | 48.45 | 75 | 0.733 / 542mi | |
| Config 47 [1,500,300] | 961 | 11.3 / 24.0 | 48.74 | 75 | 0.280 / 183mi | |
| Config 48 [1,200,300] | 981 | 8.73 / 11.0 | 122.84 | 135 | 0.198 / 187mi | |

10 Conclusions

This deliverable presented the consolidated version of CAMP. Compared to the previous version, it includes:

- A consolidated metamodel for specifying base configurations and variations around those configurations (Section 6)



- New strategies for selecting configurations, to automatically select a small set of representative configurations, rather than enumerating all possible configurations (Section 7)
- A new execution engine to run the generated configuration, execute tests and aggregate test reports, for JUnit and JMeter. (Section 8)
- Stronger empirical evidences about the usefulness of CAMP, with for example 20 new configuration bugs identified in XWiki since 01-07-2018, and new configuration bugs discovered in Tellu Cloud (Section 9).