

Load testing and functional testing with Taurus

- [Overview](#)
- [Requirements](#)
- [Description](#)
 - [Load testing example, with SLAs](#)
 - [Setup: checking out the JMeter project and setup of auxiliary variables](#)
 - [Configuring the Build steps](#)
 - [Configuring the Post-build actions](#)
 - [Load testing example, without SLAs](#)
 - [Functional testing example](#)
- [Room for improvement](#)
- [References](#)

Overview

[Taurus](#) is an open-source automation tool used for load and also functional testing.

Taurus acts as an abstraction layer on top of different load and functional tools, allowing users to easily maintain the YAML/JSON-based test scripts by storing them in the source control system.

Taurus can interact with JMeter, Gatling, Robot and other tools.

By using its [CLI](#), Taurus can easily be integrated in the CI pipeline.

It can produce XML/CSV based reports or even JUnit XML based reports. Reports can also be uploaded to BlazeMeter where they can be analyzed in further extent.

Taurus provides a kind of SLA/SLO mechanism based on "[pass/fail](#)" [criteria](#). Criteria can be defined based on typical load testing metrics (i.e. the "runtime criteria") and/or on the [monitoring data](#) obtained from the services running the target system. Thus, Taurus can be used as a way to extend JMeter, for example, and provide it advanced SLAs.

Requirements

- Taurus
- JMeter (installed automatically)
- Jenkins (optional)

Description

In the following example, we'll perform testing on a [fictitious "Simple Travel Agency" site](#) (kindly provided by BlazeMeter for demo purposes).

The overall approach to have visibility of the performance results in Xray will be as follows:

1. run Taurus in command line
2. generate multiple test reports
 - a. standard results in CSV and XML formats
 - b. custom JUnit XML report with additional info
3. submit results to Xray along with the previously generated report assets
 - a. fill out the "Description" field of the corresponding created Test Execution issue with
 - i. link to project/job in Jenkins
 - ii. link to BlazeMeter report
 - iii. summary results formatted as a table

Load testing example, with SLAs

In this example, we will load test the site.

Welcome to the Simple Travel Agency!

The is a sample site you can test with BlazeMeter!

Check out our [destination of the week! The Beach!](#)

Choose your departure city:

Paris ▼

Choose your destination city:

Buenos Aires ▼

Find Flights

The [testing scenario](#) exercises 10 users, with a ramp-up period of 40s, doing a (partial) reservation interaction: go to the site, and reserve a flight from Paris to Buenos Aires.

There are several "labels" (i.e. transactions), grouping one or more actions (i.e. HTTP requests). However, there are no explicit assertions.

The scenario itself and execution-related details are described in an YAML file.

execution.yml

```
---
execution:
- concurrency: 10
  hold-for: 2m
  ramp-up: 40s
  scenario: Thread Group
scenarios:
  Thread Group:
    requests:
    - label: blazedemo
      method: GET
      url: http://blazedemo.com/
    - body:
        fromPort: Paris
        toPort: Buenos Aires
      label: reserve
      method: POST
      url: http://blazedemo.com/reserve.php
```

Multiple reporting modules are configured to process the results. A custom module produces a JUnit XML report; this module is a customized variant over the standard [junit-xml](#) module.

passfail_config.yml

```
---
modules:
  custom-junit-xml:
    class: bitcoder.bzt.customreporting.JUnitXMLReporter

reporting:
- module: passfail
  criteria:
    - avg-rt>10ms for 7s, stop as failed
    - hits of reserve >10 for 13s, continue as failed
- module: custom-junit-xml
  filename: junit_report.xml
  data-source: pass-fail
  classname: bzt
- module: final-stats
  summary: true # overall samples count and percent of failures
  percentiles: true # display average times and percentiles
  summary-labels: true # provides list of sample labels, status, percentage of completed, avg time and errors
  failed-labels: true # provides list of sample labels with failures
  test-duration: true # provides test duration
  dump-xml: dump.xml
  dump-csv: dump.csv
- module: blazemeter
  report-name: Jenkins Build
  test: Taurus Demo
  project: Taurus
  upload-artifacts: true # upload artifacts when test is finished
  browser-open: none # can be "start", "end", "both", "none"
```

This modified custom-junit-xml module accepts one additional setting "classname" that can be used to customize the classname attribute on the target <testcase> elements of the XML report ; otherwise, "bzt" will be used. We can use to have an unique identifier such as "PurchaseUserPath".

The module also provides additional information that may be shown in Xray.

bitcoder/bzt/customreporting.py

```
"""
Basics of reporting capabilities

Copyright 2015 BlazeMeter Inc.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
"""
import copy
import csv
import locale
import os
import sys
import time
import pdb
from collections import Counter, OrderedDict
from datetime import datetime

from terminaltables import AsciiTable
```

```

from bzt import TaurusInternalException, TaurusConfigError
from bzt.engine import Reporter
from bzt.modules.aggregator import DataPoint, KPISet, AggregatorListener, ResultsProvider
from bzt.modules.blazemeter import BlazeMeterUploader, CloudProvisioning
from bzt.modules.functional import FunctionalAggregatorListener
from bzt.modules.passfail import PassFailStatus
from bzt.utils import etree, iteritems, get_full_path, is_windows

if is_windows():
    from terminaltables import AsciiTable as SingleTable
else:
    from terminaltables import SingleTable

class FinalStatus(Reporter, AggregatorListener, FunctionalAggregatorListener):
    """
    A reporter that prints short statistics on test end
    """

    def __init__(self):
        super(FinalStatus, self).__init__()
        self.last_sec = None
        self.cumulative_results = None
        self.start_time = time.time() # default value
        self.end_time = time.time()
        self.first_ts = float("inf")
        self.last_ts = 0

    def startup(self):
        self.start_time = time.time()

    def prepare(self):
        super(FinalStatus, self).prepare()
        if isinstance(self.engine.aggregator, ResultsProvider):
            self.engine.aggregator.add_listener(self)
        elif self.engine.is_functional_mode():
            self.engine.aggregator.add_listener(self)

    def aggregated_second(self, data):
        """
        Just store the latest info

        :type data: bzt.modules.aggregator.DataPoint
        """
        self.first_ts = min(self.first_ts, data[DataPoint.TIMESTAMP])
        self.last_ts = max(self.last_ts, data[DataPoint.TIMESTAMP])
        self.last_sec = data

    def aggregated_results(self, results, cumulative_results):
        """
        Just store the latest info

        :type cumulative_results: bzt.modules.functional.ResultsTree
        :type results: bzt.modules.functional.ResultsTree
        """
        self.cumulative_results = cumulative_results

    def shutdown(self):
        self.end_time = time.time()

    def post_process(self):
        """
        Log basic stats
        """
        super(FinalStatus, self).post_process()

        if self.parameters.get("test-duration", True):
            self.__report_duration()

        if self.last_sec:

```

```

summary_kpi = self.last_sec[DataPoint.CUMULATIVE]["]

if self.parameters.get("summary", True):
    self.__report_samples_count(summary_kpi)
if self.parameters.get("percentiles", True):
    self.__report_percentiles(summary_kpi)

if self.parameters.get("summary-labels", True):
    self.__report_summary_labels(self.last_sec[DataPoint.CUMULATIVE])

if self.parameters.get("failed-labels"):
    self.__report_failed_labels(self.last_sec[DataPoint.CUMULATIVE])

if self.parameters.get("dump-xml"):
    self.__dump_xml(self.parameters.get("dump-xml"))

if self.parameters.get("dump-csv"):
    self.__dump_csv(self.parameters.get("dump-csv"))
elif self.cumulative_results:
    self.__report_summary()

report_mode = self.parameters.get("report-tests", "failed")
if report_mode == "failed":
    self.__report_failed_tests()
else:
    self.__report_all_tests()

def __plural(self, count, noun):
    return noun + 's' if count > 1 else noun

def __report_all_tests(self):
    for test_suite in self.cumulative_results.test_suites():
        for case in self.cumulative_results.test_cases(test_suite):
            full_name = case.test_suite + "." + case.test_case
            self.log.info("Test %s - %s", full_name, case.status)
            print_trace = self.parameters.get("print-stacktrace", True)
            if print_trace and case.error_trace:
                self.log.info("Stacktrace:\n%s", case.error_trace)

def __report_failed_tests(self):
    for test_suite in self.cumulative_results.test_suites():
        for case in self.cumulative_results.test_cases(test_suite):
            if case.status in ("FAILED", "BROKEN"):
                full_name = case.test_suite + "." + case.test_case
                msg = "Test {test_case} failed: {error_msg}".format(test_case=full_name, error_msg=case.
error_msg)

                if case.error_trace:
                    msg += "\n" + case.error_trace
                self.log.warning(msg)

def __report_summary(self):
    status_counter = Counter()
    for test_suite in self.cumulative_results.test_suites():
        for case in self.cumulative_results.test_cases(test_suite):
            status_counter[case.status] += 1

    # FIXME: it's actually not tests, but test cases
    total = sum(count for _, count in iteritems(status_counter))
    self.log.info("Total: %s %s", total, self.__plural(total, 'test'))

def __report_samples_count(self, summary_kpi_set):
    """
    reports samples count
    """
    if summary_kpi_set[KPISet.SAMPLE_COUNT]:
        err_rate = 100 * summary_kpi_set[KPISet.FAILURES] / float(summary_kpi_set[KPISet.SAMPLE_COUNT])
        self.log.info("Samples count: %s, %.2f%% failures", summary_kpi_set[KPISet.SAMPLE_COUNT], err_rate)

def __report_percentiles(self, summary_kpi_set):
    """
    reports percentiles

```

```

"""
fmt = "Average times: total %.3f, latency %.3f, connect %.3f"
self.log.info(fmt, summary_kpi_set[KPISet.AVG_RESP_TIME], summary_kpi_set[KPISet.AVG_LATENCY],
              summary_kpi_set[KPISet.AVG_CONN_TIME])

data = [("Percentile, %", "Resp. Time, s")]
for key in sorted(summary_kpi_set[KPISet.PERCENTILES].keys(), key=float):
    data.append((float(key), summary_kpi_set[KPISet.PERCENTILES][key]))
    # self.log.info("Percentile %.1f%%: %.3f", )
table = SingleTable(data) if sys.stdout.isatty() else AsciiTable(data)
table.justify_columns[0] = 'right'
table.justify_columns[1] = 'right'
self.log.info("Percentiles:\n%s", table.table)

def __report_failed_labels(self, cumulative):
    """
    reports failed labels
    """
    report_template = "%d failed samples: %s"
    sorted_labels = sorted(cumulative.keys())
    for sample_label in sorted_labels:
        if sample_label != "":
            failed_samples_count = cumulative[sample_label]['fail']
            if failed_samples_count:
                self.log.info(report_template, failed_samples_count, sample_label)

def __console_safe_encode(self, text):
    return text.encode(locale.getpreferredencoding(), errors='replace').decode('unicode_escape')

def __get_sample_element(self, sample, label_name):
    failed_samples_count = sample['fail']
    success_samples_count = sample['succ']
    total_samples_count = failed_samples_count + success_samples_count
    assert total_samples_count > 0, "Total samples is zero for %s" % label_name
    success_samples_perc = (success_samples_count * 100) / total_samples_count

    errors = set()
    for err_desc in sample['errors']:
        errors.add(self.__console_safe_encode(err_desc["msg"]))

    return (
        self.__console_safe_encode(label_name),
        "FAIL" if failed_samples_count > 0 else "OK",
        "{0:.2f}%".format(round(success_samples_perc, 2)),
        "{0:.3f}".format(round(sample['avg_rt'], 3)),
        "\n".join(errors)
    )

def __report_summary_labels(self, cumulative):
    data = [("label", "status", "succ", "avg_rt", "error")]
    justify = {0: "left", 1: "center", 2: "right", 3: "right", 4: "left"}

    sorted_labels = sorted(cumulative.keys())
    for sample_label in sorted_labels:
        if sample_label != "":
            data.append(self.__get_sample_element(cumulative[sample_label], sample_label))
    table = SingleTable(data) if sys.stdout.isatty() else AsciiTable(data)
    table.justify_columns = justify
    self.log.info("Request label stats:\n%s", table.table)

def __report_duration(self):
    """
    asks executors start_time and end_time, provides time delta
    """
    date_start = datetime.fromtimestamp(int(self.start_time))
    date_end = datetime.fromtimestamp(int(self.end_time))
    self.log.info("Test duration: %s", date_end - date_start)

def __dump_xml(self, filename):
    self.log.info("Dumping final status as XML: %s", filename)
    root = etree.Element("FinalStatus")

```

```

if self.first_ts < float("inf") and self.last_ts > 0:
    duration_elem = etree.Element("TestDuration")
    duration_elem.text = str(round(float(self.last_ts - self.first_ts), 3))
    root.append(duration_elem)

report_info = get_bza_report_info(self.engine, self.log)
if report_info:
    link, _ = report_info[0]
    report_element = etree.Element("ReportURL")
    report_element.text = link
    root.append(report_element)
if self.last_sec:
    for label, kpi_set in iteritems(self.last_sec[DataPoint.CUMULATIVE]):
        root.append(self.__get_xml_summary(label, kpi_set))

with open(get_full_path(filename), 'wb') as fhd:
    tree = etree.ElementTree(root)
    tree.write(fhd, pretty_print=True, encoding="UTF-8", xml_declaration=True)

def __get_xml_summary(self, label, kpi_set):
    elem = etree.Element("Group", label=label)
    for kpi_name, kpi_val in iteritems(kpi_set):
        if kpi_name in (KPISet.ERRORS, KPISet.RESP_TIMES):
            continue

        if isinstance(kpi_val, dict):
            for param_name, param_val in iteritems(kpi_val):
                elem.append(self.__get_kpi_xml(kpi_name, param_val, param_name))
        else:
            elem.append(self.__get_kpi_xml(kpi_name, kpi_val))

    return elem

def __get_kpi_xml(self, kpi_name, kpi_val, param=None):
    kpi = etree.Element(kpi_name)
    kpi.attrib['value'] = self.__val_to_str(kpi_val)
    elm_name = etree.Element("name")
    elm_name.text = kpi_name
    if param is not None:
        kpi.attrib['param'] = self.__val_to_str(param)
        elm_name.text += "/" + param

    kpi.append(elm_name)

    elm_value = etree.Element("value")
    elm_value.text = self.__val_to_str(kpi_val)
    kpi.append(elm_value)

    return kpi

def __val_to_str(self, kpi_val):
    if isinstance(kpi_val, float):
        return '%.5f' % kpi_val
    elif isinstance(kpi_val, int):
        return '%d' % kpi_val
    elif isinstance(kpi_val, str):
        return kpi_val
    else:
        raise TaurusInternalException("Unhandled kpi type: %s" % type(kpi_val))

def __dump_csv(self, filename):
    self.log.info("Dumping final status as CSV: %s", filename)
    # FIXME: what if there's no last_sec
    with open(get_full_path(filename), 'wt') as fhd:
        fieldnames = self.__get_csv_dict('', self.last_sec[DataPoint.CUMULATIVE]['']).keys()
        writer = csv.DictWriter(fhd, fieldnames)
        writer.writeheader()
        for label, kpi_set in iteritems(self.last_sec[DataPoint.CUMULATIVE]):
            writer.writerow(self.__get_csv_dict(label, kpi_set))

```

```

def __get_csv_dict(self, label, kpi_set):
    kpi_copy = copy.deepcopy(kpi_set)
    res = OrderedDict()
    res['label'] = label

    # sort label
    for key in sorted(kpi_copy.keys()):
        res[key] = kpi_copy[key]

    del res[KPISet.ERRORS]
    del res[KPISet.RESP_TIMES]
    del res[KPISet.RESP_CODES]
    del res[KPISet.PERCENTILES]

    percentiles = list(iteritems(kpi_set[KPISet.PERCENTILES]))
    for level, val in sorted(percentiles, key=lambda lv: (float(lv[0]), lv[1])):
        res['perc_%s' % level] = val

    resp_codes = list(iteritems(kpi_set[KPISet.RESP_CODES]))
    for rcd, val in sorted(resp_codes):
        res['rc_%s' % rcd] = val

    for key in res:
        if isinstance(res[key], float):
            res[key] = "%.5f" % res[key]

    return res

class JUnitXMLReporter(Reporter, AggregatorListener, FunctionalAggregatorListener):
    """
    A reporter that exports results in Jenkins JUnit XML format.
    """

    def __init__(self):
        super(JUnitXMLReporter, self).__init__()
        self.last_second = None
        self.report_file_path = None
        self.cumulative_results = None

    def prepare(self):
        if isinstance(self.engine.aggregator, ResultsProvider):
            self.engine.aggregator.add_listener(self)
        elif self.engine.is_functional_mode():
            self.engine.aggregator.add_listener(self)

    def aggregated_second(self, data):
        self.last_second = data

    def aggregated_results(self, _, cumulative_results):
        """
        :type cumulative_results: bzt.modules.functional.ResultsTree
        """
        self.cumulative_results = cumulative_results

    def post_process(self):
        """
        Get report data, generate xml report.
        """
        filename = self.parameters.get("filename", None)
        if not filename:
            filename = self.engine.create_artifact(XUnitFileWriter.REPORT_FILE_NAME, XUnitFileWriter.
REPORT_FILE_EXT)
        self.parameters["filename"] = filename # reflect it in effective config

        #pdb.set_trace()
        if self.cumulative_results is None:
            test_data_source = self.parameters.get("data-source", "sample-labels")
            class_name = self.parameters.get("classname", "bzt")
            #pdb.set_trace()

```



```

        if test_data_source == "sample-labels":
            if not self.last_second:
                self.log.warning("No last second data to generate XUnit.xml")
            else:
                writer = XUnitFileWriter(self.engine, class_name)
                self.process_sample_labels(writer)
                writer.save_report(filename)
        elif test_data_source == "pass-fail":
            writer = XUnitFileWriter(self.engine, class_name)
            self.process_pass_fail(writer)
            writer.save_report(filename)
        else:
            raise TaurusConfigError("Unsupported data source: %s" % test_data_source)
    else:
        writer = XUnitFileWriter(self.engine, class_name)
        self.process_functional(writer)
        writer.save_report(filename)

    self.report_file_path = filename # TODO: just for backward compatibility, remove later

def process_sample_labels(self, xunit):
    """
    :type xunit: XUnitFileWriter
    """
    xunit.report_test_suite('sample_labels')
    labels = self.last_second[DataPoint.CUMULATIVE]

    for key in sorted(labels.keys()):
        if key == "": # skip total label
            continue

        errors = []
        for er_dict in labels[key][KPISet.ERRORS]:
            rc = str(er_dict["rc"])
            msg = str(er_dict["msg"])
            cnt = str(er_dict["cnt"])
            if er_dict["type"] == KPISet.ERRTYPE_ASSERT:
                err_element = etree.Element("failure", message=msg, type="Assertion Failure")
            else:
                err_element = etree.Element("error", message=msg, type="Error")
            err_desc = "%s\n(status code is %s)\n(total errors of this type: %s)" % (msg, rc, cnt)
            err_element.text = err_desc
            errors.append(err_element)

        #pdb.set_trace()
        duration = str(round(labels[key]['avg_rt'], 3))
        xunit.report_test_case('sample_labels', key, errors, duration)

def process_pass_fail(self, xunit):
    """
    :type xunit: XUnitFileWriter
    """
    xunit.report_test_suite('bzt_pass_fail')
    mods = self.engine.reporters + self.engine.services # TODO: remove it after passfail is only reporter
    pass_fail_objects = [_x for _x in mods if isinstance(_x, PassFailStatus)]
    self.log.debug("Processing passfail objects: %s", pass_fail_objects)
    fail_criteria = []

    for pf_obj in pass_fail_objects:
        if pf_obj.criteria:
            for _fc in pf_obj.criteria:
                fail_criteria.append(_fc)

    for fc_obj in fail_criteria:
        if 'label' in fc_obj.config:
            data = (fc_obj.config['subject'], fc_obj.config['label'], fc_obj.config['condition'],
                    fc_obj.config['threshold'])
            tpl = "%s of %s%s%s"
        else:
            data = (fc_obj.config['subject'], fc_obj.config['condition'], fc_obj.config['threshold'])
            tpl = "%s%s%s"

```

```

        if fc_obj.config['timeframe']:
            tpl += " for %s"
            data += (fc_obj.config['timeframe'],)
        disp_name = tpl % data

        if fc_obj.is_triggered and fc_obj.fail:
            error = etree.Element("error", message=str(fc_obj), type="pass/fail criteria triggered")
            error.text = str(fc_obj)
            errors = [error]
        else:
            errors = ()

        #pdb.set_trace()
        # fc_obj.__dict__
        # fc_obj.label
        xunit.report_test_case('bzt_pass_fail', disp_name, errors)

def process_functional(self, xunit):
    for suite_name, samples in iteritems(self.cumulative_results):
        duration = max(s.start_time for s in samples) - min(s.start_time for s in samples)
        duration += max(samples, key=lambda s: s.start_time).duration
        attrs = {
            "name": suite_name,
            "tests": str(len(samples)),
            "errors": str(len([sample for sample in samples if sample.status == "BROKEN"])),
            "skipped": str(len([sample for sample in samples if sample.status == "SKIPPED"])),
            "failures": str(len([sample for sample in samples if sample.status == "FAILED"])),
            "time": str(round(duration, 3)),
            # TODO: "timestamp" attribute
        }
        xunit.add_test_suite(suite_name, attributes=attrs)
        for sample in samples:
            attrs = {
                "classname": sample.test_suite,
                "name": sample.test_case,
                "time": str(round(sample.duration, 3))
            }
            children = []
            if sample.status == "BROKEN":
                error = etree.Element("error", type=sample.error_msg)
                if sample.error_trace:
                    error.text = sample.error_trace
                children.append(error)
            elif sample.status == "FAILED":
                failure = etree.Element("failure", message=sample.error_msg)
                if sample.error_trace:
                    failure.text = sample.error_trace
                children.append(failure)
            elif sample.status == "SKIPPED":
                skipped = etree.Element("skipped")
                children.append(skipped)
            xunit.add_test_case(suite_name, attributes=attrs, children=children)

def get_bza_report_info(engine, log):
    """
    :return: [(url, test), (url, test), ...]
    """
    result = []
    if isinstance(engine.provisioning, CloudProvisioning):
        cloud_prov = engine.provisioning
        test_name = cloud_prov.settings.get("test")
        report_url = cloud_prov.results_url
        result.append((report_url, test_name if test_name else report_url))
    else:
        bza_reporters = [_x for _x in engine.reporters if isinstance(_x, BlazeMeterUploader)]
        for bza_reporter in bza_reporters:
            if bza_reporter.results_url:
                test_name = bza_reporter.parameters.get("test")
                report_url = bza_reporter.results_url
                result.append((report_url, test_name if test_name else report_url))

```

```

        if len(result) > 1:
            log.warning("More than one blazemeter reporter found")
        return result

class XUnitFileWriter(object):
    REPORT_FILE_NAME = "xunit"
    REPORT_FILE_EXT = ".xml"

    def __init__(self, engine, class_name):
        """
        :type engine: bzt.engine.Engine
        """
        super(XUnitFileWriter, self).__init__()
        self.engine = engine
        self.log = engine.log.getChild(self.__class__.__name__)
        self.test_suites = OrderedDict()
        bza_report_info = get_bza_report_info(engine, self.log)
        self.class_name = class_name #bza_report_info[0][1] if bza_report_info else "bzt-" + str(self.
__hash__())
        self.report_urls = ["BlazeMeter report link: %s\n" % info_item[0] for info_item in bza_report_info]

    def save_report(self, fname):
        """
        :type fname: str
        """
        try:
            if os.path.exists(fname):
                self.log.warning("File %s already exists, it will be overwritten", fname)
            else:
                dirname = os.path.dirname(fname)
                if dirname and not os.path.exists(dirname):
                    os.makedirs(dirname)

                testsuites = etree.Element("testsuites")
                for _, suite in iteritems(self.test_suites):
                    testsuites.append(suite)
                etree_obj = etree.ElementTree(testsuites)

                self.log.info("Writing JUnit XML report into: %s", fname)
                with open(get_full_path(fname), 'wb') as _fds:
                    etree_obj.write(_fds, xml_declaration=True, encoding="UTF-8", pretty_print=True)
        except BaseException:
            raise TaurusInternalException("Cannot create file %s" % fname)

    def report_test_suite(self, suite_name):
        """
        :type suite_name: str
        :type children: list[lxml.etree.Element]
        """
        self.add_test_suite(suite_name, attributes={"name": suite_name, "package_name": "bzt"})

    def report_test_case(self, suite_name, case_name, children=None, duration=None):
        """
        :type suite_name: str
        :type case_name: str
        :type children: list[lxml.etree.Element]
        """
        children = children or []
        if self.report_urls:
            system_out = etree.Element("system-out")
            system_out.text = "".join(self.report_urls)
            children.insert(0, system_out)
        if duration is not None:
            self.add_test_case(suite_name, attributes={"classname": self.class_name, "name": case_name, "time":
duration}, children=children)
        else:
            self.add_test_case(suite_name, attributes={"classname": self.class_name, "name": case_name},
children=children)

```

```
def add_test_suite(self, suite_name, attributes=None, children=()):
    attributes = attributes or {}

    suite = etree.Element("testsuite", **attributes)

    for child in children:
        suite.append(child)

    if not suite_name in self.test_suites:
        self.test_suites[suite_name] = suite

def add_test_case(self, suite_name, attributes=None, children=()):
    attributes = attributes or {}

    case = etree.Element("testcase", **attributes)

    for child in children:
        case.append(child)

    self.test_suites[suite_name].append(case)
```

Test scenarios can be run using the [command line bzt tool](#).

We'll use Jenkins as our CI tool and we'll configure a freestyle project for performing our load testing.



Please note

If we aim to send the reports to BlazeMeter, we need to configure the API token used by Taurus' bzt. Instead of hard-coding this in Taurus configuration files, it can be set on the Jenkins user' home settings (more info [here](#)).

~/.bzt-rc

```
# BlazeMeter reporting settings
#modules:
# blazemeter:
#   token: <key id>:<key secret> # API id and API secret joined with ':'

modules:
  blazemeter:
    token: 22321b40xxxxxx:d1d3abcb97xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    public-report: true
```

Setup: checking out the JMeter project and setup of auxiliary variables

We need to setup some variables related to the Jira instance to be able to attach some files to the Test Execution issue later on, if we want to, using the `attach_files_to_issue.sh` shell script.

These are somehow redundant with the Xray instance configuration but are necessary if we wish to expose them.

We start by defining one variable for the Jira server base URL as build parameter.

☒ This project is parameterized

String Parameter

Name

JIRA_BASEURL

Default Value

http://192.168.56.102

Description

[Plain text] [Preview](#)

☒ Trim the string

Add Parameter

Using the [Credentials Binding plugin](#), we will populate two variables for the Jira instance's username and password; these will be, in turn, obtained from the credentials already stored and linked to the Xray instance configuration in Jenkins.

Bindings

Username and password (separated)

Username Variable

JIRA_USERNAME

Password Variable

JIRA_PASSWORD

Credentials

☒ Specific credentials ☐ Parameter expression

admin/***** (Jira admin user)

Add

The "code" will be checked out from our source code versioning system (e.g. Git), which contain the Taurus configuration files along with some additional scripts.

Source Code Management

☐ None ☒ Git

Repositories

Repository URL

ssh://git@localhost/home/git/repos/taurus-exp.git

Credentials

git/*****

Add

Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/master

X

Add Branch

Repository browser

(Auto)

Configuring the Build steps

The "build" is composed of several steps, starting with the one that runs Taurus.

Here we may decide to enforce the build step as successful, as shown in the screenshot, or let it fail or not depending on the load testing results. The latter would require to define the additional build steps as post-actions though.

Build

Execute shell

Command

```
# make build always pass, otherwise other build steps arent executed
./run3.sh || true
```

See [the list of available environment variables](#)

We'll store the artifact inside the directory artifacts. We'll also customize the report name in BlazeMeter so that it contains the Jenkins build number.

./run3.sh

```
#!/bin/bash
bzt -v -o settings.artifacts-dir=artifacts  examples/3/execution.yml examples/3/passfail_config.yml -o modules.
blazemeter.report-name="Jenkins Build ${BUILD_NUMBER}"
```

Optionally, we'll add two build steps to store the tabular aggregate report in an environment variable (e.g. AGGREGATE_TABLE) as a string. This requires the [Environment Injector plugin](#).

Execute shell

Command

```
#!/bin/bash
echo BLAZEMETER_URL=$(cat junit_report.xml | egrep -o "BlazeMeter report link: .*" | sort | uniq) > envvars.properties
echo AGGREGATE_TABLE="$(./process_aggregate.sh)" >> envvars.properties
```

./process_aggregate.sh

```
#!/bin/bash

cat dump.csv | sed -e 's/^,/TOTAL,/g' | tr ",," " " | sed -e 's/^/|/' | sed -e 's/\r$/|\\n\\n/' | sed -e '1 s/|/|/g'
```

Inject environment variables

Properties File Path

envvars.properties

Properties Content

Configuring the Post-build actions



Bonus tip!

The Jenkins' [Performance plugin](#) can optionally be used to create some trend charts in Jenkins and also as means to mark the build as failed or unstable depending on absolute or relative thresholds.

Publish Performance test result report

Source data files (autodetects format):

Regex for included samplers

Select evaluation mode ☐ Expert Mode ☒ Standard Mode

Standard Mode

Select mode: ☒ Relative Threshold ☐ Error Threshold

Use Error thresholds on single build:

Unstable	<input type="text" value="-1"/>
Failed	<input type="text" value="-1"/>

Advanced...

Use Relative thresholds for build comparison:

	(-)	(+)
Unstable % Range	<input type="text" value="-1,0"/>	<input type="text" value="-1,0"/>
Failed % Range	<input type="text" value="-1,0"/>	<input type="text" value="-1,0"/>

☐ Compare with previous Build ☒ Compare with Build number

Compare based on

Expert Mode

Constraint settings ☐ Ignore Failed Builds ☐ Ignore Unstable Builds ☐ Save constraint log to workspace

JUnit output file (optional)

Constraints

Advanced...

Test results can be submitted to Xray either by using a command line tool (e.g. `curl`) or by using a specific CI plugin which in our case will be the "[Xray – Test Management for Jira Plugin](#)".

We could choose the "JUnit XML" as the format in the "Xray: Results Import Task", that would be simpler to setup.

However, if we use the "JUnit XML multipart" format, we can further customize the Test Execution issue. We'll use this as means to provide a link to the Jenkins build along with a link to more in-depth details at BlazeMeter site. We may also provide the aggregate report table stored previously as an environment variable.

Post-build Actions

Xray: Results Import Task

Jira Instance

xray-vm

Format

JUnit XML multipart

Parameters

Import to Same Test Execution

☐

When this option is checked, if you are importing multiple execution report files using a glob expression, the results will be imported to the same Test Execution

Execution Report File (file path with file name)

junit_report.xml

Test Execution fields

JSON Content

```
{
  "fields": {
    "project": {
      "key": "CALC"
    },
    "summary": "Taurus (JMeter) performance results",
    "description": "Build URL: ${BUILD_URL}\n\n${BLAZEMETER_URL}\n\nAggregate results summary"\n\n${AGGREGATE_TABLE}\n",
    "issuetype": {
      "name": "Test Execution"
    }
  }
}
```

If using this format, you'll need to provide the Test Execution's issue type name (or the id) and the project key.

Test Execution fields (JSON content) - example1

```
{
  "fields": {
    "project": {
      "key": "CALC"
    },
    "summary": "Taurus (JMeter) performance results",
    "description": "Build URL:  ${BUILD_URL}.\n\n${BLAZEMETER_URL}\n\n*Aggregate results summary*\n\n${AGGREGATE_TABLE}\n",
    "issuetype": {
      "name": "Test Execution"
    }
  }
}
```

You may also specify the Test Plan, Revision and Test Environments fields but you'll need to obtain their custom field ID from Jira's administration. Note that these IDs are specific to each Jira instance. In the following example, "customfield_10033" corresponds to the Revision CF, "customfield_11805" to the Test Environments CF and "customfield_11807" to the Test Plan CF.

Test Execution fields (JSON content) - example2

```
{
  "fields": {
    "project": {
      "key": "CALC"
    },
    "summary": "Taurus (JMeter) performance results",
    "description": "Build URL:  ${BUILD_URL}.\n\n${BLAZEMETER_URL}\n\n*Aggregate results summary*\n\n${AGGREGATE_TABLE}\n",
    "issuetype": {
      "name": "Test Execution"
    },
    "customfield_10033": "123",
    "customfield_11805" : [
      "staging"
    ],
    "customfield_11807": [
      "CALC-1200"
    ]
  }
}
```



Bonus tip!

You may also attach some files (e.g. logs, reports) to the created Test Execution issue.

The Jenkins plugin exports the `XRAY_TEST_EXECS` variable containing the issue key of the Test Execution that was created.

Post build task

Tasks

Log text

Operation **-- AND --**

Add

Script

```
ISSUEKEY=$(echo $XRAY_TEST_EXECS | cut -d ";" -f 1)
echo "issue_key: $ISSUEKEY"
./attach_files_to_issue.sh $ISSUEKEY artifacts/bzt.log
```

Run script only if all previous steps were successful ☐

Escalate script execution status to job status ☐

For the time being, the Jenkins plugin can't upload other files; however, we can make a basic shell script (e.g. `attach_files_to_issue.sh`) for that.

attach_files_to_issue.sh

```
#!/bin/bash

BASEURL=${JIRA_BASEURL:-http://yourjiraserver.example.com}
USERNAME=${JIRA_USERNAME:-admin}
PASSWORD=${JIRA_PASSWORD:-admin}

ISSUEKEY=$1

for file in "${@:2}"
do
    curl -D- -u $USERNAME:$PASSWORD -X POST -H "X-Atlassian-Token: nocheck" -F "file=@$file" $BASEURL/rest
    /api/2/issue/$ISSUEKEY/attachments
done
```

After running Jenkins job, some performance information will be directly available in Jenkins; this is provided by the Performance plugin (if you've previously configured it as mentioned earlier), either on the project page or on the build page.

Jenkins

3

search

admin | log out

Jenkins

taurus-loadtesting-jmeter

Back to Dashboard

Status

Changes

Workspace

Build with Parameters

Delete Project

Configure

Performance Trend

Rename

Build History

trend

find

#24

Jun 2, 2020 12:19 PM

#23

Jun 1, 2020 12:24 AM

#22

May 31, 2020 9:32 PM

RSS for all

RSS for failures

Project taurus-loadtesting-jmeter

Workspace

Last Successful Artifacts

dashBoard_dump.xml

640 B

view

standardResults.xml

468 B

view

Recent Changes

Permalinks

Last build (#24) · 7 min 30 sec ago

Last stable build (#24) · 7 min 30 sec ago

Last successful build (#24) · 7 min 30 sec ago

Last completed build (#24) · 7 min 30 sec ago

Performance Trend

add description

Disable Project

Response time

Percentage of errors

Jenkins

3

search

admin | log out

Jenkins

taurus-loadtesting-jmeter

Performance Trend

Back to Dashboard

Status

Changes

Workspace

Build with Parameters

Delete Project

Configure

Performance Trend

Rename

Build History

trend

find

#24

Jun 2, 2020 12:19 PM

#23

Jun 1, 2020 12:24 AM

#22

May 31, 2020 9:32 PM

RSS for all

RSS for failures

Performance Trend

Last Report

Filter trend data

Test file: dump.xml

Response time

Percentage of errors

Trend report

Jenkins

3

search

admin | log out

Jenkins

taurus-loadtesting-jmeter

#24

Performance

Back to Project

Status

Changes

Console Output

Edit Build Information

Delete build '#24'

Parameters

Environment Variables

Git Build Data

No Tags

Performance Report

View External Report

Previous Build

Performance Breakdown by URI: dump.xml

Response time

Percentage of errors

Response time trends for build: "taurus-loadtesting-jmeter #24"

Comparison with previous build

URI	Samples	Average (ms)	Min(ms)	Median(ms)	Line 90.0(ms)	Max(ms)	Http Code	Errors (%)	Average (KB)	Total (KB)
blazedemo	17	2256 +482	856 +816	2204 +700	3226 +492	3606 -388		11.765 % +7.417 %	0.0 0.0	0.0 0.0
reserve	13	725 +144	183 -276	746 +181	906 +249	912 -76		7.692 % +7.692 %	0.0 0.0	0.0 0.0
All URIs	30 -14	1592 +388	183 +143	1137 +149	2898 +622	3606 -388		10.0 % +7.727 %	126811.3	3804339.0

As we submitted the processed test results to Xray (e.g. [junit_report.xml](#)), we can now track them in Jira.

A Test Execution will be created containing a summary of results along with some useful links to access additional information in Jenkins.

Taurus (JMeter) performance results

[Edit](#) [Comment](#) [Synchronize Tests from...](#) [More](#) [Close Issue](#) [Reopen Issue](#) [Admin](#)

Details

Type: **Test Execution**
Priority: **Major**
Affects Version/s: **None**
Component/s: **None**
Labels: **None**
Test Environments: **None**
Test Plan: **None**

Status: **RESOLVED** ([View Workflow](#))
Resolution: **Fixed**
Fix Version/s: **None**

Description

Build URL: <http://192.168.56.102:8081/job/taurus-loadtesting-jmeter/24/>.

BlazeMeter report link: <https://a.blazemeter.com/app/#/masters/28891504>

Aggregate results summary

label	avg_ct	avg_lt	avg_rt	bytes	concurrency	fail	stddev_rt	succ	throughput	perc_0.0	perc_50.0	perc_90.0	perc_95.0	perc_99.0	perc_99.9	perc_100.0	rc_200	rc_SocketException
TOTAL	0.11513	0.34323	1.59277	3804339	4	3	0.93863	27	30	0.18300	1.13700	2.89800	3.22600	3.60600	3.60600	3.60600	27	3
reserve	0.07238	0.28169	0.72515	72534	4	1	0.18807	12	13	0.18300	0.74600	0.90600	0.91200	0.91200	0.91200	0.91200	12	1
blazedemo	0.14782	0.39029	2.25624	3731805	5	2	0.71524	15	17	0.85600	2.20400	3.22600	3.60600	3.60600	3.60600	3.60600	15	2

Tests

[+ Add](#)

Overall Execution Status

1 PASS 1 FAIL

Total Tests: 2

[Filter\(s\)](#)

[Filter](#)

Show [100](#) entries

Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	CALC-6592	avg-rt of >10ms for 7s	Generic	0	0	Administrator	FAIL
2	CALC-6591	hits of reserve >10 for 13s	Generic	0	0	Administrator	PASS

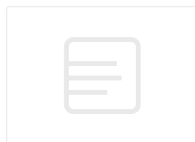
Showing 1 to 2 of 2 entries

[First](#) [Previous](#) [Next](#) [Last](#)

The attachments section on the Test Execution issue provide direct access to some reports and also to a zipped file containing the dashboard report generated by JMeter.

Attachments

[Drop files to attach, or browse.](#)



[bzt.log](#)

18 minutes ago

97 kB



Unstructured (i.e. "generic") Test issues will be auto-provisioned (unless they already exist), one per each SLA criteria defined in the `passfail` module configuration. The "Generic Definition" field acts as the unique test identifier for subsequent imports and is composed by a prefix along with the criteria (e.g. "PurchaseUserPath.avg-rt of >10ms for 7s", "bzt.avg-rt of >10ms for 7s").

If `classname` configuration has been specified in the `custom-junit-xml` module, it will be used as the prefix; otherwise "bzt" will be used.

The execution details of a specific Test Run show whether the pass/fail criteria (i.e. our SLA) passed or not.

The following screenshot showcases the details of a failed criterium (e.g. average response-time greater than 10ms for 19 seconds).

[Export Test as Text](#)[Return to Test Execution](#)[Next](#)

Execution Status **FAIL**



Assignee: **Administrator**

Versions: -

Executed By: **Administrator**

Revision: -

Started On: **02/Jun/20 12:19 PM**

Finished On: **02/Jun/20 12:19 PM**

Tests -
environments:

Comment

Preview Comment

Execution Defects (0)

Create Defect

Create Sub-Task

Add Defects

Execution Evidence (0)

Add Evidence

Execution Details

Test Description

None

Test Details

Test Type: Generic

Definition: **PurchaseUserPath.avg-rt of >10ms for 7s**

Results

Context	Output	Duration	Status
TestSuite bzt_pass_fail	Failed: avg-rt>10ms for 19 sec	-	FAIL

Results can be further analyzed on BlazeMeter site (if you've previously configured the "blazemeter" reporter).

You can access the report by using a specific link within the Jenkins build screen, or by using the link provided inside the Test Execution's description field in Xray.

Jenkins

Jenkins > taurus-loadtesting-jmeter > #24

[Back to Project](#)
[Status](#)
[Changes](#)
[Console Output](#)
[Edit Build Information](#)
[Delete build '#24'](#)
[Parameters](#)
[Environment Variables](#)
[Git Build Data](#)
[No Tags](#)
[Performance Report](#)
[View External Report](#)
[Previous Build](#)

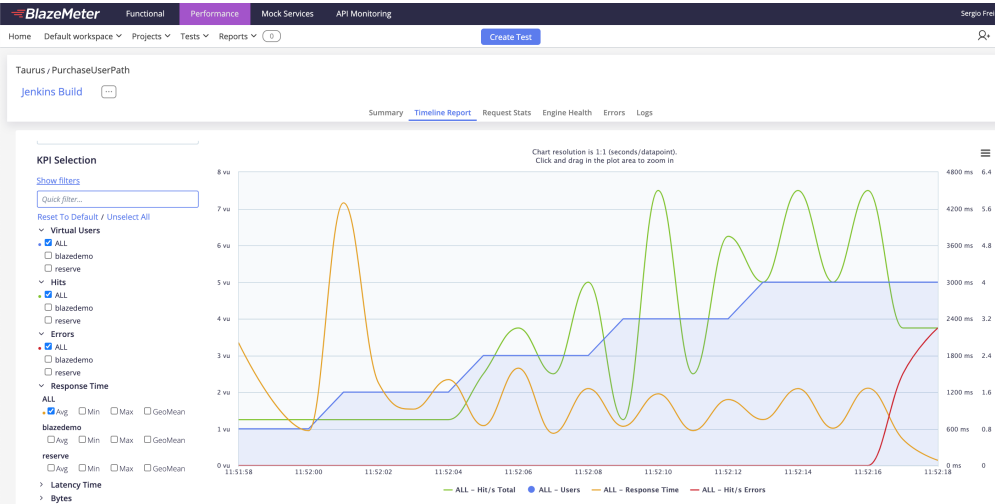
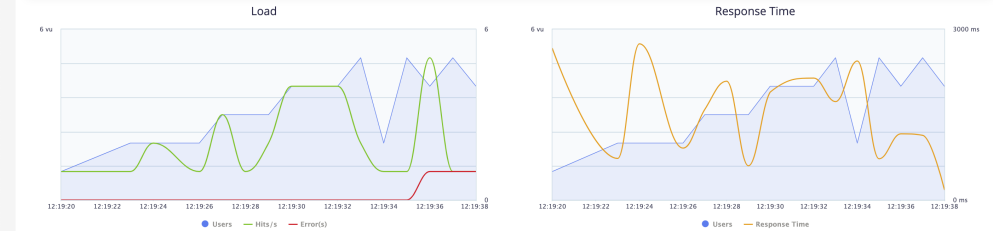
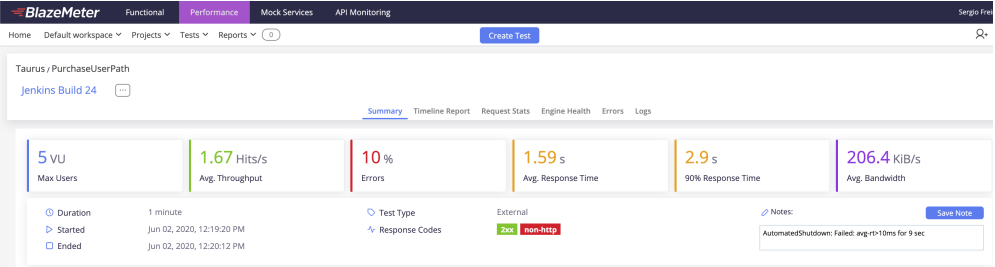
Build #24 (Jun 2, 2020 12:19:05 PM)

[Build Artifacts](#)
 [dashBoard_dump.xml](#) 640 B [view](#)
 [standardResults.xml](#) 468 B [view](#)

No changes.

Started by user [admin](#)

Revision: 61772ed5a4800d9459d57852bca9ec30e056c1e7
• refs/remotes/origin/master



BlazeMeter Functional Performance Mock Services API Monitoring Sergio Frei

Home Default workspace Projects Tests Reports Create Test

Taurus / PurchaseUserPath
Jenkins Build

Summary Timeline Report Request Stats Engine Health Errors Logs

Filter By Label:

Search label...

Results per page: 25 Go to page: 1

Results: 3 out of 3 labels

Element Label ▼ ▲	# Samples ▼ ▲	Avg. Response Time (ms) ...	Avg. Hits/s ▼ ▲	90% line (ms) ▼ ▲	95% line (ms) ▼ ▲	99% line (ms) ▼ ▲	Min Response Time (ms) ...	Max Response Time (ms) ...	Avg. Bandwidth (KBytes/s) ...	Error Percentage ▼ ▲
ALL	57	1036	2.85	1594	1719	4296	41	4296	398.57	8.77%
blazemeter	29	1515	1.45	1719	2005	4296	41	4296	391.13	3.45%
reserve	28	540	1.4	706	738	922	71	922	7.43	14.29%



Bonus tip!

After Tests are auto-provisioned in Xray, they can be manually linked (i.e. cover) to a performance-related requirement/user story issue. This will provide you the ability to track coverage directly on the requirement issue.



Calculator / CALC-6595

site reserve performance

Edit Comment Assign More Start Progress Resolve Issue Close Issue Admin

Details

Type: Requirement
Priority: Major
Affects Version/s: None
Component/s: None
Labels: None
Requirement Status: UNCOVERED
Status: OPEN (View Workflow)
Resolution: Unresolved
Fix Version/s: None

Description

Test Coverage

Create Test Create Sub-Test Execution + Link

No Tests were found testing the requirement.

Add Tests to Issue CALC-6595

Select Search JQL

Tests CALC-6591 x CALC-6592 x +

Start typing to get a list of possible matches or press down to select from a list of existing Tests to add to this Issue



Calculator / CALC-6595

site reserve performance

Edit Comment Assign More Start Progress Resolve Issue Close Issue Admin

Details

Type: Requirement
Priority: Major
Affects Version/s: None
Component/s: None
Labels: None
Requirement Status: NOK
Status: OPEN (View Workflow)
Resolution: Unresolved
Fix Version/s: None

Description

Test Coverage

Create Test Create Sub-Test Execution + Link

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; Version: None - latest execution; Environment: All Environments

NOK

Filter(s)



Show 10 entries Columns

P	Status	Resolution	Key	Summary	Test Runs	Test Status
<input type="checkbox"/>	OPEN	Unresolved	CALC-6591	hits of reserve >10 for 13s	10	PASS
<input type="checkbox"/>	OPEN	Unresolved	CALC-6592	avg-rt of >10ms for 7s	10	FAIL

Load testing example, without SLAs

This example is similar to the previous one (please have a look at it first), except that we won't define SLAs using the pass-fail module.

execution.yml

```
---
execution:
- concurrency: 10
  hold-for: 2m
  ramp-up: 40s
  scenario: Thread Group
scenarios:
  Thread Group:
    requests:
    - label: blazedemo
      method: GET
      url: http://blazedemo.com/
    - body:
        fromPort: Paris
        toPort: Buenos Aires
      label: reserve
      method: POST
      url: http://blazedemo.com/reserve.php
```

Multiple reporting modules are configured to process the results. A custom module (see code in the previous example) produces a JUnit XML report; this module is a customized variant over the standard [junit-xml](#) module.

modules_config.yml

```
---
modules:
  custom-junit-xml:
    class: bitcoder.bzt.customreporting.JUnitXMLReporter

reporting:
- module: custom-junit-xml
  filename: junit_report.xml
  data-source: sample-labels
  classname: PurchaseUserPath
- module: final-stats
  summary: true # overall samples count and percent of failures
  percentiles: true # display average times and percentiles
  summary-labels: true # provides list of sample labels, status, percentage of completed, avg time and errors
  failed-labels: true # provides list of sample labels with failures
  test-duration: true # provides test duration
  dump-xml: dump.xml
  dump-csv: dump.csv
- module: blazemeter
  report-name: Jenkins Build
  test: Taurus Demo
  project: Taurus
  upload-artifacts: true # upload artifacts when test is finished
  browser-open: none # can be "start", "end", "both", "none"
```

Upon submission to Xray (e.g. [junit_report.xml](#)), each labeled request is converted to an unstructured (i.e. Generic) Test, uniquely identified by a prefix (e.g. PurchaseUserPath) along with the label.



Calculator / CALC-6596

Execution results - junit_report.xml - [1591102880441]

[Edit](#) [Comment](#) [Synchronize Tests from...](#) [More](#) [Close Issue](#) [Reopen Issue](#) [Admin](#)

Details

Type: **Test Execution** Status: **RESOLVED** ([View Workflow](#))
Priority: **Medium** Resolution: **Fixed**
Affects Version/s: **None** Fix Version/s: **None**
Component/s: **None**
Labels: **None**
Test Environments: **None**
Test Plan: **None**

Description

Tests

[+ Add](#)

Overall Execution Status

2 PASS

Total Tests: 2

[Filter\(s\)](#)

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status	
1	CALC-6597	blazedemo	Generic	0	0	Administrator	PASS	
2	CALC-6598	reserve	Generic	0	0	Administrator	PASS	

Calculator / [Test Execution: CALC-6596](#) / [Test: CALC-6598](#)
[reserve](#)

[Export Test as Text](#) [Return to Test Execution](#) [Previous](#)

Execution Status: **PASS**

Started On: **02/Jun/20 2:01 PM** Finished On: **02/Jun/20 2:01 PM**

Assignee: **Administrator** Versions: -
Executed By: **Administrator** Revision: -
Tests: -
environments: -

[Comment](#) [Preview Comment](#) [Execution Defects \(0\)](#) [Create Defect](#) [Create Sub-Task](#) [Add Defects](#) [Execution Evidence \(0\)](#) [Add Evidence](#)

Execution Details

Test Description

None

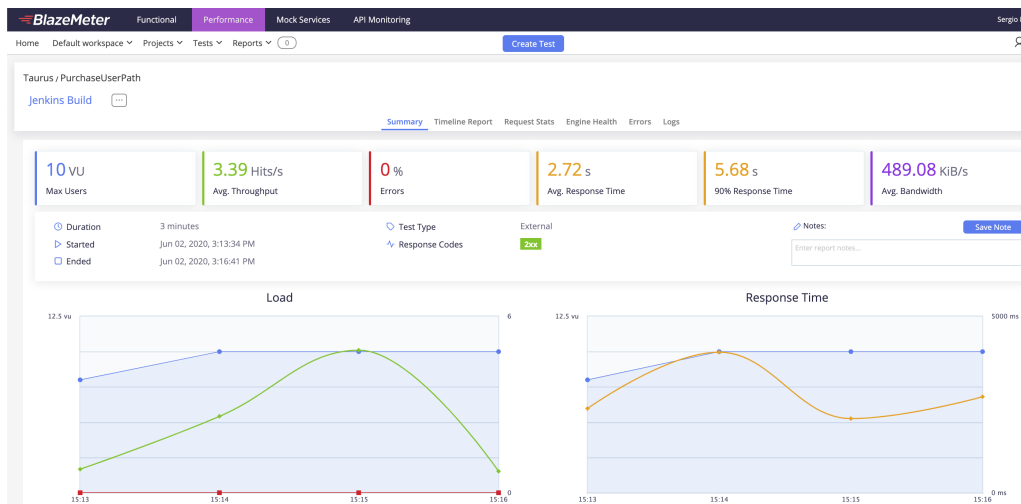
Test Details

Test Type: **Generic**
Definition: **PurchaseUserPath.reserve**

Results

Context	Output	Duration	Status
TestSuite sample_labels	-	1 sec	PASS

Detailed reports can be analyzed in BlazeMeter site.



Functional testing example

Taurus can also be used for functional testing. Taurus can execute Selenium-based tests directly or invoke other test automation frameworks (e.g. JUnit) to run these type of tests.

To assist you writing U/Selenium test scripts, you may use a specific [Chrome extension](#) that is able of generation a YAML configuration file Taurus friendly.

This file can then be further customized to your needs (e.g. for adding assertions).

functional_selenium.yml

```
modules:
  custom-junit-xml:
    class: bitcoder.bzt.customreporting.JUnitXMLReporter
  selenium:
    remote: https://22321b4098ad5b2e7b2060ee:
dld3abcb97d59125619277ced5d7bf9a000263c672301d998d7cd00515a8aa1531b7be72@a.blazemeter.com/api/v4/grid/wd/hub
  capabilities:
    blazemeter.testName: Purchase test
    blazemeter.projectId: 634491
    blazemeter.locationId: US East (Virginia) - Functional GUI Test
  custom-junit-xml:
    class: bitcoder.bzt.customreporting.JUnitXMLReporter

execution:
- executor: selenium
  scenario: full_purchase_user_path
  iterations: 1
  capabilities:
    browserName: chrome

- executor: selenium
  scenario: full_purchase_user_path
  iterations: 1
  capabilities:
    browserName: firefox

scenarios:
  full_purchase_user_path:
    generate-flow-markers: true
    headless: false
    timeout: 60s
    think-time: 0s
    requests:
      - label: Open Travel Agency site
        actions:
```

```

- go(http://blazedemo.com/)
- assertTitle(BlazeDemo)
- label: select From
  actions:
    - selectByName(fromPort): "Portland"
    - clickByName(fromPort)
    - assertTitle(BlazeDemo)
- label: select To
  actions:
    - selectByName(toPort): "Dublin"
    - clickByName(toPort)
- label: click Find
  actions:
    - clickByCSS(input.btn.btn-primary)
    - assertTitle(BlazeDemo - reserve)
- label: click Choose
  actions:
    - clickByXPath(("//input[@value='Choose This Flight']")[3])
    - assertTitle(BlazeDemo Purchase)
- label: enter Name
  actions:
    - clickByID(inputName)
    - typeByID(inputName): "John"
- label: enter City
  actions:
    - clickByID(city)
    - typeByID(city): "Madrid"
- label: click Purchase
  actions:
    - clickByCSS(input.btn.btn-primary)
    - assertTitle(BlazeDemo Confirmation)
  assert:
    - contains:
      - 'Thank you for your purchase today!'
    subject: body

```

reporting:

```

- module: final-stats
  summary: true # overall samples count and percent of failures
  percentiles: true # display average times and percentiles
  summary-labels: true # provides list of sample labels, status, percentage of completed, avg time and errors
  failed-labels: true # provides list of sample labels with failures
  test-duration: true # provides test duration
  dump-xml: dump.xml
  dump-csv: dump.csv
- module: custom-junit-xml
  filename: junit_report.xml
  data-source: sample-labels # sample-labels, pass-fail
- module: blazemeter
  report-name: full purchase user path
  test: Taurus Demo
  project: Taurus
  upload-artifacts: true # upload artifacts when test is finished
  browser-open: none # can be "start", "end", "both", "none"

```

```
17:13:07 INFO: Average times: total 7.157, latency 0.000, connect 0.000
17:13:07 INFO: Percentiles:
```

Percentile, %	Resp. Time, s
0.0	4.552
50.0	5.5
90.0	10.696
95.0	11.248
99.0	11.248
99.9	11.248
100.0	11.248

```
17:13:07 INFO: Request label stats:
```

label	status	succ	avg_rt	error
Open Travel Agency site	OK	100.00%	4.634	
click Choose	OK	100.00%	4.935	
click Find	OK	100.00%	4.732	
click Purchase	OK	100.00%	5.441	
enter City	OK	100.00%	8.183	
enter Name	OK	100.00%	8.184	
select From	OK	100.00%	10.974	
select To	OK	100.00%	10.169	

```
17:13:07 INFO: Dumping final status as XML: dump.xml
17:13:07 INFO: Dumping final status as CSV: dump.csv
17:13:07 WARNING: File junit_report.xml already exists, it will be overwritten
17:13:07 INFO: Writing JUnit XML report into: junit_report.xml
17:13:07 INFO: Sending remaining KPI data to server...
17:13:09 INFO: Uploading all artifacts as artifacts.zip ...
17:13:10 INFO: Uploading /Users/smsf/exps/taurus1/2020-06-01_17-11-35.146669/bzt.log
17:13:11 INFO: Ending data feeding...
17:13:12 INFO: Online report link: https://a.blazemeter.com/app/#/masters/28873337
```

After test is run, a JUnit XML report is produced (e.g. [junit_report.xml](#)). This can be submitted to Xray.

An unstructured (i.e. Generic) Test will be created per each action.

▼ Tests

+ Add ▼

Overall Execution Status

8 PASS

Total Tests: 8

Filter(s)

▼

Show 100 entries

Columns ▼

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status	
1	CALC-6566	click Choose	Generic	0	0	Administrator	PASS	▶ ...
2	CALC-6565	enter City	Generic	0	0	Administrator	PASS	▶ ...
3	CALC-6564	enter Name	Generic	0	0	Administrator	PASS	▶ ...
4	CALC-6563	select From	Generic	0	0	Administrator	PASS	▶ ...
5	CALC-6568	Open Travel Agency site	Generic	0	0	Administrator	PASS	▶ ...
6	CALC-6561	click Find	Generic	0	0	Administrator	PASS	▶ ...
7	CALC-6560	select To	Generic	0	0	Administrator	PASS	▶ ...
8	CALC-6559	click Purchase	Generic	0	0	Administrator	PASS	▶ ...

The unique identifier for the Test (i.e. the value of the Generic definition field) is composed by a prefix along with the label.

Calculator / Test Execution: CALC-6567 / Test: CALC-6568
[Open Travel Agency site](#)

Execution Status: **PASS**

Started On: 31/May/20 6:17 PM Finished On: 31/May/20 6:17 PM

Assignee: Administrator Executed By: Administrator

Tests environments: -

Comment Preview Comment

Execution Defects (0) Create Defect Create Sub-Task Add Defects

Execution Evidence (0) Add Evidence

Execution Details

Test Description

None

Test Details

Test Type: Generic

Definition: **bzt.Open Travel Agency site**

Results

Context	Output	Duration	Status
TestSuite sample_labels	-	4 sec	PASS

Results can be further analyzed in BlazeMeter' site.



Please note

Unless you run your tests in BlazeMeter using the remote webdriver, test results will appear under the Performance section and you won't be able to see action level pass/fail information.

Please check the [Selenium Executor](#) and the [Apiritif Executor](#) documentation.

Our test was run against two different browsers, thus we can see the two distinct results.

BlazeMeter Functional Performance Mock Services API Monitoring Sergio Freire

Home Default workspace Projects Tests Reports 0 Create Test

Taurus

Purchase test

GUI Functional Test • Updated 7 minutes ago

☐ Send email when test is done

History

TESTS Non-Debug Debug

Test Name	Last Run	Pass %	Pass/Fail
Purchase test	Today at 5:11 PM	100%	Pass
Purchase test	Today at 5:11 PM	100%	Pass

TRENDS

Date	Value
Jun 01 05:11	8
Jun 01 05:11	8

On each test result, it is possible to evaluate the step/group of actions (i.e. label) result along with the inner actions. It's also possible to watch a recording of the test session.

BlazeMeter

FunctionalPerformanceMock ServicesAPI Monitoring

Sergio P...

HomeDefault workspaceProjectsTestsReports

Create Test

Search

Taurus / Purchase test

Purchase test

Select:firefox_65 (r-sg-5ed528bbd4209)

SummaryDetails

VideoWaterfallLogsMetadata

00:00:00 / 00:01:14

BlazeMeter

Test steps onlyAll commands

full_purchase_user_path

Open Travel Agency site

00:10Open Urlhttp://www.taurusdemo.com/

00:12Execute/SyncScript: if (window.__webdriverAlerts() & returns) window.__webdriverAlerts...Args: []

00:13Get Page Titletaurusdemo

select From

select To

click Find

click Choose

enter Name

enter City

click Purchase

Web driver closed

BlazeMeter

FunctionalPerformanceMock ServicesAPI Monitoring

Sergio P...

HomeDefault workspaceProjectsTestsReports

Create Test

Search

Taurus / Taurus Demo

full_purchase_user_path

SummaryTimeline ReportRequest StatsEngine HealthErrorsLogs

2 VUMax Users

0.24 Hits/sAvg. Throughput

0 %Errors

7.16 sAvg. Response Time

10.7 s90% Response Time

0.48 B/sAvg. Bandwidth

Duration2 minutes

StartedJun 01, 2020, 5:11:51 PM

EndedJun 01, 2020, 5:13:13 PM

Test TypeExternal

Response Codes

Notes

Save Note

Load

Response Time

BlazeMeter

FunctionalPerformanceMock ServicesAPI Monitoring

Sergio P...

HomeDefault workspaceProjectsTestsReports

Create Test

Search

Taurus / Taurus Demo

full_purchase_user_path

SummaryTimeline ReportRequest StatsEngine HealthErrorsLogs

05:11:5105:12:57

05:11:5105:11:5305:11:5605:11:5905:12:0705:12:1705:12:2605:12:57

Filter by Label:

Search label...

Results per page:25Go to page:1

Results: 9 out of 9 labels

Element Label	Samples	Avg. Response Time (ms)	Avg. Hits/s	90% line (ms)	95% line (ms)	99% line (ms)	Min Response Time (ms)	Max Response Time (ms)	Avg. Bandwidth (B/bytes/s)	Error Percentage
ALL	16	7156	0.24	10696	11248	11248	4552	11248	0	0%
click Choose	2	4935	0.03	4992	4992	4992	4876	4992	0	0%
click Find	2	4732	0.03	4752	4752	4752	4712	4752	0	0%
click Purchase	2	5440	0.03	5500	5500	5500	5380	5500	0	0%
enter City	2	8182	0.03	8192	8192	8192	8172	8192	0	0%
enter Name	2	8183	0.03	8192	8192	8192	8167	8192	0	0%
Open Travel Agency si...	2	4634	0.03	4716	4716	4716	4552	4716	0	0%

It's also possible to access an execute summary report.

←

→

↺

🔒

a.blazemeter.com/app/#/accounts/530446/workspaces/532492/projects/634491/masters/28873337/summary

🔍

☆

⚙️

🔔

👤

📄

🚫

BlazeMeter

Functional

Performance

Mock Services

API Monitoring

Serg

Home

Default workspace

Projects

Tests

Reports

0

Create Test

Taurus / Taurus Demo

full purchase user path

Share Report

Enable Accessibility Features

Executive Summary

Compare Report

Danger Zone

Delete

2 VU

Max Users

0 %

Errors

7.16 s

Avg. Response Time

10.7 s

90% Response Time

0.48 B/s

Avg. Bandwidth

⌚ Duration

▶ Started

☐ Ended

Jun 01, 2020, 5:13:11 PM

Jun 01, 2020, 5:13:13 PM

Jun 01, 2020, 5:13:13 PM

Test Type

Response Codes

External

Notes:

Save Note

Enter report notes...

Load

Response Time

5 vu

0.3

5 vu

10000

←

→

↺

🔒

a.blazemeter.com/app/executive-summary/index.html?master_id=28873337#/

🔍

☆

🚫

BlazeMeter

LOAD TEST REPORT

full purchase user path

≡B

Report created by Sergio Freire.
Date of Run: Mon, 06/01/2020 - 17:11
Duration: 2 minutes

HIDE SUMMARY

DESCRIPTIVE SUMMARY / CONCLUSIONS

📈

Average Throughput

0.2

Hits/s

🕒

Avg. Response Time

7156

Milliseconds

📊

90% Response Time

10696

Milliseconds

🚫

Error Rate

0.00

%

TOP 5 SLOWEST RESPONSES (BY AVG. RESPONSE TIME)

Request	# Samples	Avg Time	90% Time	Max Time
select From	2	10974.00 ms	11248 ms	11248 ms
select To	2	10169.00 ms	10320 ms	10320 ms
enter Name	2	8183.00 ms	8192 ms	8192 ms
enter City	2	8182.00 ms	8192 ms	8192 ms
click Purchase	2	5440.00 ms	5500 ms	5500 ms



Please note

Assertion errors can be tracked in the Errors Report section of the overall test results.

Errors Report

This report displays all errors received during the test run, categorized by labels (pages) and error types.

Seeing Response Code **200** as an error? [Read this post to understand why.](#)

Group errors by Label Response Code Assertion Name

This groups all the data by each label

> ALL

Response Codes: **1** Asse

▼ click Purchase

Response Codes: **1** Asse

Response Codes

Code	Description	Count
	AssertionError: 0 == 0 : Assertion: 'Thank you for your purchase today!' not found in BODY	1

Room for improvement

- abstract the whole Taurus test as a single Test
- use Robot Framework XML report instead of JUnit to provide more granular details
- provide the possibility of linking test(s) to an existing requirement in Xray

References

- <https://github.com/Blazemeter/taurus>
- <https://gettaurus.org/>
- <https://gettaurus.org/docs/PassFail/>
- <https://gettaurus.org/learn/>
- <https://gettaurus.org/docs/JMeter/#Assertions>
- <https://gettaurus.org/kb/SeleniumActions/>
- <https://chrome.google.com/webstore/detail/blazemeter-the-continuous/mbopgmdnpscbohpnfpglghlbhfhongabi?hl=en>
- <https://gettaurus.org/kb/Reporting/>
- <https://www.blazemeter.com/blog/how-to-perform-local-GUI-functional-test-through-Taurus>
- <https://gettaurus.org/docs/BlazemeterReporter/>