Anuket Reference Conformance for RA2 based Implementations (RC2)

Anuket

Sep 12, 2022

Contents

This is the reference conformance specification (RC2) for RA2-based implementations.

1 Introduction

1.1 Executive Summary

The Reference Conformance for the Kubernetes-based workstream (RC2) was established to ensure implementations of the Anuket Reference Architecture 2 (RA2), such as the Reference Implementation 2 (RI2), meet functional and performance requirements specified in RA2 and the Anuket Reference Model (RM). Cloud infrastructure and workload verification and validation will be utilised to evaluate **Conformance** (i.e. adherence) to the RA2 and RM requirements. Conformance scope includes:

- Test cases, with traceability to requirements, to validate that the cloud infrastructure implementation meets the expected capabilities specified in RA-2 and that the workloads consume compliant cloud infrastructure resources
- Verify, with requirements traceability, that the installation cookbooks (manifests) of RI-2 are in conformance with the RA-2 specifications (for example, software versions, plugins, and configurations)
- Guidelines for processes, environments, and tools for enabling conformance testing

In summary, **Conformance** testing will be performed as part of cloud infrastructure and workload lifecycle testing which includes **Verification** and **Validation**, defined further as:

- **Verification** is performed using reviews (e.g., manifests) to ensure that the cloud infrastructure or workload is delivered as per implementation specifications.
- **Validation** confirms the infrastructure or workload meet the expected or desired behaviour by means of automated testing.

All Terms utilized throughout this chapter are intended to align with LFN Compliance and Verification Committee (CVC) definitions, and their use through CVC documentation, guidelines, and standards.

1.2 Overview

Document Purpose

This chapter includes traceability from requirements to test cases and a test case execution framework to ensure Kubernetes infrastructure meets the design, feature, and capability expectations of the RM and RA2. Ultimately, RC2 will reduce the amount of time and cost it takes each operator to on-board and maintain cloud infrastructure and CNFs.

1.3 RC2 End User Requirements

Telecommunication service providers / operators are the primary intended audience for RC2 results. Specifically those selecting infrastructure and network function technologies to use in their network. The RC2 result artifact(s) should be clear and provide confidence to the service provider that the test results meet the requirements *they have*.

Vendors/Developers are a secondary audience. They should be able to clearly see the mapping from a specific test result (pass or fail) to the requirement.

Operator/SP Test Result Requirements

- Clickable links from test cases to requirements
- Pass, Fail, or Skipped for every test
- Reason for failures
- Reason for skipping
- Format supporting clickable links (e.g., HTML)
- Provide a stable set of point-in-time requirements and tests to achieve conformance
- Testing tools allow users to select between validation of mandatory and optional requirements
- Enable clear tracability with versioning to know which requirements have and haven't been covered and track changes over time
- Tests must be available to run locally
- Testing tools must produce machine-readable result formats

Vendor Test Result Requirements

- Clear mapping between requirements and test results
- Enable clear tracability to know which requirements have and haven't been covered and track changes over time
- Failures should provide additional content to inform the user where or how the requirement was violated without having to read the test
- Expected preconditions and environment requirements for any test tooling must be defined

1.4 Scope

This document covers aspects of conformance for both Kubernetes based cloud infrastructure and workloads. The document will cover the following topics:

- Identify in detail the requirements of test-cases (mapped from RA2 and RM)
- Test criteria that shows a certain capability or feature of the system-under-test exists and behaves as expected
- An E2E framework for conformance of Kubernetes infrastructures and workloads, including specification for conformance test infrastructure (lab environment and tools)
- Analysis to identify where the gaps are in the industry for implementing conformance test objectives (tooling, methods, process, etc)

Not in Scope

- Functional testing / validation of the application provided by the workload is outside the scope of this work
- Testing to confirm anything not in RM or RA2 requirements
- VNFM/NFVO, like ONAP, is not used in the process flow for infrastructure verifications or validations
- Upgrades to workloads, and the respective processes of verifying upgrade procedures and validating (testing) the success and compatibility of upgrades is not in scope

1.5 Guidelines

The objectives of the Reference Conformance for cloud infrastructure is to verify implementations against the reference architecture which satisfies infrastructure needs for workloads. The objectives of the Reference Conformance for workloads is to verify workload implementations consume resources and behave as expected against the reference architecture.

These guidelines will drive RC2 deliverables:

- RC2 requirements are completely derived from RM and RA2 which specify infrastructure capabilities including compute, memory, storage, resource capabilities, performance optimization capabilities, and monitoring capabilities.
- Requirements in the RM and RAs that are performance related may not have minimum performance criteria identified but where feasible will have tests with metrics to show relevant capabilities are present and working as expected.
- Must/shall conformance criteria are testable as pass/fail and/or reporting of quantitative test results. This will ensure infrastructures and workloads meet minimum thresholds of functional operability and/or performance behavior. This is the focus of RC2 since it is what will drive a commercially significant badging program.
- Should/may conformance criteria, which may or may not be testable, provide recommendations or best-practices for functional operability and/or performance behavior. These criteria and associated tests can be very useful

for developing, evaluating or deploying a cloud infrastructure but are not critical to a commercially significant badging program.

1.6 Conformance Methodologies

The RC2 test suite will provide validation to ensure workloads can interoperate with the RA2 conformant infrastructure. Upstream projects will define features/capabilities, test scenarios, and test cases to be executed. 3rd Party test platforms may also be leveraged if desired.

Dependencies infrastructure and workload validation will rely upon test harnesses, test tools, and test suites provided by upstream projects, including Anuket and CNF conformance. These upstream projects will be reviewed semi-annually to verify they are still healthy and active projects. Over time, the projects representing the conformance process may change, but test parity is required if new test suites are added in place of older, stale projects.

1.7 Reading Guide and Usage

RC2 focuses on testing of Kubernetes based cloud infrastructure thus the chapter structure is designed to facilitate this by matching test cases to requirements and building test cookbooks. If you are looking for requirements or the reasons behind them, please refer to the RA2. Chapters 2 and 3 cover Kubernetes infrastructure conformance while 4 and 5 cover CNF conformance.

Chapter 2 takes the requirements from the RA2 and matches them to upstream test cases. This will cover how specific test cases map to requirements and the overall coverage of requirements with test cases. Chapter 3 outlines how these test cases can be integrated together into an automated toolchain to test conformance of the Kubernetes infrastructure.

Similarly, Chapter 4 maps test cases map to requirements for CNFs and Chapter 5 builds a testing cookbook. Chapter 6 encompasses any gaps in the Reference Conformance 2.

2 Kubernetes Test Cases and Requirements Traceability

2.1 Introduction

All of the requirements for RC2 have been defined in the Reference Model (RM) and Reference Architecture (RA2). The scope of this chapter is to identify and list down test cases based on these requirements. Users of this chapter will be able to use it to determine which test cases they must run in order to test compliance with the requirements. This will enable traceability between the test cases and requirements. They should be able to clearly see which requirements are covered by which tests and the mapping from a specific test result (pass or fail) to a requirement. Each requirement may have one or more test case associated with it.

Goals

- Clear mapping between requirements and test cases
- Provide a stable set of point-in-time requirements and tests to achieve conformance
- Enable clear traceability of the coverage of requirements across consecutive releases of this document
- Clickable links from test cases to requirements
- One or more tests for every MUST requirement
- A set of test cases to serve as a template for Anuket Assured

Non-Goals

- Defining any requirements
- Providing coverage for non-testable requirements

Definitions

must: Test Cases that are marked as must are considered mandatory and must pass successfully

should: Test Cases that are marked as should are expected to be fulfilled by the cloud infrastructure but it is up to each service provider whether to accept a cloud infrastructure that is not fulfilling any of these requirements. The same applies to should not.

may: Test cases that are marked as may are considered optional. The same applies to may not.

2.2 Traceability Matrix

Kubernetes API testing

The primary objectives of the e2e tests are to ensure a consistent and reliable behavior of the Kubernetes code base, and to catch hard-to-test bugs before users do, when unit and integration tests are insufficient. They are partially selected for the Software Conformance Certification program run by the Kubernetes community (under the aegis of the CNCF).

Anuket shares the same goal to give end users the confidence that when they use a certified product they can rely on a high level of common functionality. Then Anuket RC2 starts with the test list defined by K8s Conformance which is expected to grow according to the ongoing requirement traceability.

End-to-End Testing basically asks for focus and skip regexes to select or to exclude single tests:

- focus basically matches Conformance or Testing Special Interest Groups in sub-sections below
- skip excludes the SIG labels listed as optional in API and Feature Testing requirements.

The Reference Conformance suites must be stable and executed on real deployments. Then all the following labels are defacto skipped in End-to-End Testing:

- alpha
- Disruptive
- Flaky

It's worth mentioning that no alpha or Flaky test can be included in Conformance as per the rules.

Conformance

It must be noted that the default K8s Conformance testing is disruptive thus Anuket RC2 rather picks non-disruptive-conformance testing as defined by Sonobuoy.

focus: *Conformance*

skip:

- [Disruptive]
- NoExecuteTaintManager

API Machinery Testing

focus: [sig-api-machinery]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:CrossNamespacePodAffinity]
- [Feature:CustomResourceValidationExpressions]
- [Feature:StorageVersionAPI]

See API Machinery Special Interest Group and API and Feature Testing requirements for more details.

Apps Testing

focus: [sig-apps]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:DaemonSetUpdateSurge]
- [Feature:IndexedJob]
- [Feature:StatefulSet]
- [Feature:StatefulSetAutoDeletePVC]
- [Feature:StatefulUpgrade]
- [Feature:SuspendJob]

See Apps Special Interest Group and API and Feature Testing requirements for more details.

Auth Testing

focus: [sig-auth]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:BoundServiceAccountTokenVolume]
- [Feature:PodSecurityPolicy]

See Auth Special Interest Group and API and Feature Testing requirements for more details.

Cluster Lifecycle Testing

focus: [sig-cluster-lifecycle]

skip:

- [alpha]
- [Disruptive]
- [Flaky]

See Cluster Lifecycle Special Interest Group and API and Feature Testing requirements for more details.

Instrumentation Testing

focus: [sig-instrumentation]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:Elasticsearch]
- [Feature:StackdriverAcceleratorMonitoring]
- [Feature:StackdriverCustomMetrics]
- [Feature:StackdriverExternalMetrics]
- [Feature:StackdriverMetadataAgent]
- [Feature:StackdriverMonitoring]

See Instrumentation Special Interest Group and API and Feature Testing requirements for more details.

Network Testing

The regexes load.balancer, LoadBalancer and Network.should.set.TCP.CLOSE_WAIT.timeout are currently skipped because they haven't been covered successfully neither by sig-release-1.23-blocking nor by Anuket RC2 verification

Please note that a couple of tests must be skipped by name below as they are no appropriate labels.

focus: [sig-network]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:Example]
- [Feature:Ingress]
- [Feature:IPv6DualStack]
- [Feature:kubemci]

- [Feature:KubeProxyDaemonSetMigration]
- [Feature:KubeProxyDaemonSetUpgrade]
- [Feature:NEG]
- [Feature:Networking-IPv6]
- [Feature:NetworkPolicy]
- [Feature:PerformanceDNS]
- [Feature:SCTP]
- [Feature:SCTPConnectivity]
- DNS configMap nameserver
- load.balancer
- LoadBalancer
- Network.should.set.TCP.CLOSE_WAIT.timeout

See Network Special Interest Group and API and Feature Testing requirements.

Node Testing

focus: [sig-node]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Feature:ExperimentalResourceUsageTracking]
- [Feature:GRPCContainerProbe]
- [Feature:GPUUpgrade]
- [Feature:PodGarbageCollector]
- [Feature:RegularResourceUsageTracking]
- [NodeFeature:DownwardAPIHugePages]
- [NodeFeature:RuntimeHandler]

See Node Special Interest Group and API and Feature Testing requirements.

Scheduling Testing

focus: [sig-scheduling]

skip:

- [alpha]
- [Disruptive]
- [Flaky]

- [Feature:GPUDevicePlugin]
- [Feature:Recreate]

See Scheduling Special Interest Group and API and Feature Testing requirements.

Storage Testing

It should be noted that all in-tree driver testing, [Driver:+], is skipped. Conforming to the upstream gate, all PersistentVolumes NFS testing is also skipped. The following exclusions are about the deprecated in-tree GitRepo volume type:

- should provision storage with different parameters
- should not cause race condition when used for git_repo

Please note that a couple of tests must be skipped by name below as they are no appropriate labels.

focus: [sig-storage]

skip:

- [alpha]
- [Disruptive]
- [Flaky]
- [Driver:+]
- [Feature:ExpandInUsePersistentVolumes]
- [Feature:Flexvolumes]
- [Feature:GKELocalSSD]
- [Feature:VolumeSnapshotDataSource]
- [Feature:Flexvolumes]
- [Feature:vsphere]
- [Feature:Volumes]
- [Feature:Windows]
- [NodeFeature:EphemeralStorage]
- PersistentVolumes.NFS
- should provision storage with different parameters
- should not cause race condition when used for git_repo

See Storage Special Interest Group and API and Feature Testing requirements.

Kubernetes API benchmarking

Rally is a tool and framework that performs Kubernetes API benchmarking.

Functest Kubernetes Benchmarking proposed a Rally-based test case, xrally_kubernetes_full, which iterates 10 times the mainline xrally-kubernetes scenarios.

At the time of writing, no KPI is defined in Kubernetes based Reference Architecture which would have asked for an update of the default SLA (maximum failure rate of 0%) proposed in Functest Kubernetes Benchmarking

Functest xrally_kubernetes_full:

Table 2.1: Kubernetes API benchmarking

Scenarios	Iterations
Kubernetes.create_and_delete_deployment	10
Kubernetes.create_and_delete_job	10
Kubernetes.create_and_delete_namespace	10
Kubernetes.create_and_delete_pod	10
Kubernetes.create_and_delete_pod_with_configmap_volume	10
Kubernetes.create_and_delete_pod_with_configmap_volume [2]	10
Kubernetes.create_and_delete_pod_with_emptydir_volume	10
Kubernetes.create_and_delete_pod_with_emptydir_volume [2]	10
Kubernetes.create_and_delete_pod_with_hostpath_volume	10
Kubernetes.create_and_delete_pod_with_secret_volume	10
Kubernetes.create_and_delete_pod_with_secret_volume [2]	10
Kubernetes.create_and_delete_replicaset	10
Kubernetes.create_and_delete_replication_controller	10
Kubernetes.create_and_delete_statefulset	10
Kubernetes.create_check_and_delete_pod_with_cluster_ip_service	10
Kubernetes.create_check_and_delete_pod_with_cluster_ip_service [2]	10
Kubernetes.create_check_and_delete_pod_with_node_port_service	10
Kubernetes.create_rollout_and_delete_deployment	10
Kubernetes.create_scale_and_delete_replicaset	10
Kubernetes.create_scale_and_delete_replication_controller	10
Kubernetes.create_scale_and_delete_statefulset	10
Kubernetes.list_namespaces	10

The following software versions are considered to benchmark Kubernetes v1.23 (latest stable release) selected by Anuket:

Table 2.2: Software versions

Software	Version
Functest	v1.23
xrally-kubernetes	1.1.1.dev12

Dataplane benchmarking

Kubernetes perf-tests repository hosts various Kubernetes-related performance test related tools especially netperf which benchmarks Kubernetes networking performance.

As listed in netperf's README, the 5 major network traffic paths are combination of pod IP vs virtual IP and whether the pods are co-located on the same node versus a remotely located pod:

- same node using pod IP
- same node using cluster/virtual IP
- remote node using pod IP
- remote node using cluster/virtual IP
- same node pod hairpin to itself using cluster/virtual IP

It should be noted that netperf leverages iperf (both TCP and UDP modes) and Netperf.

At the time of writing, no KPI is defined in Anuket chapters which would have asked for an update of the default SLA proposed in Functest Kubernetes Benchmarking.

Security testing

There are a couple of opensource tools that help securing the Kubernetes stack. Amongst them, Functest Kubernetes Security offers two test cases based on kube-hunter and kube-bench.

kube-hunter hunts for security weaknesses in Kubernetes clusters and kube-bench checks whether Kubernetes is deployed securely by running the checks documented in the CIS Kubernetes Benchmark.

kube-hunter classifies all vulnerabilities as low, medium, and high. In context of this conformance suite, only the high vulnerabilities lead to a test case failure. Then all low and medium vulnerabilities are only printed for information.

Here are the vulnerability categories tagged as high by kube-hunter:

- RemoteCodeExec
- IdentityTheft
- PrivilegeEscalation

At the time of writing, none of the Center for Internet Security (CIS) rules are defined as mandatory (e.g. sec.std.001: The Cloud Operator **should** comply with Center for Internet Security CIS Controls) else it would have required an update of the default kube-bench behavior (all failures and warnings are only printed) as integrated in Functest Kubernetes Security.

The following software versions are considered to verify Kubernetes v1.23 (latest stable release) selected by Anuket:

Table 2.3: Software versions

Software	Version
Functest	v1.23
kube-hunter	0.3.1
kube-bench	0.3.1

Opensource CNF onboarding and testing

Running opensource containerized network functions (CNF) is a key technical solution to ensure that the platforms meet Network Functions Virtualization requirements.

Functest CNF offers 2 test cases which automatically onboard and test Clearwater IMS via kubectl and Helm. It's worth mentioning that this CNF is covered by the upstream tests (see clearwater-live-test).

The following software versions are considered to verify Kubernetes v1.23 (latest stable release) selected by Anuket:

Table 2.4: Software versions

Software	Version
Functest	v1.23
clearwater	release-130
Helm	v3.3.1

2.3 Test Cases Traceability to Requirements

The following test case must pass as they are for Reference Conformance:

Table 2.5: Mandory test cases

Container	Test suite	Criteria	Requirements
opnfv/functest-kubernetes-smoke:v1.23	xrally_kubernetes	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	k8s_conformance	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	k8s_conformance_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_api_machinery	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_api_machinery_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_apps	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_apps_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_auth	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_cluster_lifecycle	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_instrumentation	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_network	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_node	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_scheduling_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_storage	PASS	Kubernetes API testing
opnfv/functest-kubernetes-smoke:v1.23	sig_storage_serial	PASS	Kubernetes API testing
opnfv/functest-kubernetes-security:v1.23	kube_hunter	PASS	Security testing
opnfv/functest-kubernetes-security:v1.23	kube_bench_master	PASS	Security testing
opnfv/functest-kubernetes-security:v1.23	kube_bench_node	PASS	Security testing
opnfv/functest-kubernetes-benchmarking:v1.23	xrally_kubernetes_full	PASS	Kubernetes API benchmarking
opnfv/functest-kubernetes-benchmarking:v1.23	netperf	PASS	Dataplane benchmarking
opnfv/functest-kubernetes-cnf:v1.23	k8s_vims	PASS	Opensource CNF onboarding and testing
opnfv/functest-kubernetes-cnf:v1.23	helm_vims	PASS	Opensource CNF onboarding and testing

3 Kubernetes Testing Cookbook

3.1 Deploy your own conformance toolchain

At the time of writing, the CI description file is hosted in Functest and only runs the containers selected by Anuket RC2. It will be completed by the next Anuket mandatory test cases and then a new CI description file will be proposed in a shared tree.

Xtesting CI only requires internet access, GNU/Linux as Operating System and asks for a few dependencies as described in Deploy your own Xtesting CI/CD toolchains:

- python-virtualenv
- git

Please note the next two points depending on the GNU/Linux distributions and the network settings:

- SELinux: you may have to add –system-site-packages when creating the virtualenv ("Aborting, target uses selinux but python bindings (libselinux-python) aren't installed!")
- Proxy: you may set your proxy in env for Ansible and in systemd for Docker https://docs.docker.com/config/daemon/systemd/#httphttps-proxy

To deploy your own CI toolchain running Anuket Compliance:

```
virtualenv functest-kubernetes --system-site-packages
. functest-kubernetes/bin/activate
pip install ansible
ansible-galaxy install collivier.xtesting
ansible-galaxy collection install ansible.posix community.general community.grafana␣
↪kubernetes.core community.docker community.postgresql
git clone https://gerrit.opnfv.org/gerrit/functest-kubernetes functest-kubernetes-src
(cd functest-kubernetes-src && git checkout -b stable/v1.23 origin/stable/v1.23)
ansible-playbook functest-kubernetes-src/ansible/site.cntt.yml
```

Configure Kubernetes API testing

Place the kubeconfig configuration file corresponding to the Kubernetes cluster under test in the following location on the machine running the cookbook:

`/home/opnfv/functest-kubernetes/config`

Run Kubernetes conformance suite

Open http://127.0.0.1:8080/job/functest-kubernetes-v1.23-daily/ in a web browser, login as admin/admin and click on "Build with Parameters" (keep the default values).

If the System under test (SUT) is Anuket compliant, a link to the full archive containing all test results and artifacts will be printed in functest-kubernetes-v1.23-zip's console. Be free to download it and then to send it to any reviewer committee.

To clean your working dir:

```
deactivate
rm -rf functest-kubernetes-src functest-kubernetes
```

4 CNF Test Cases and Requirements Traceability

4.1 Introduction

The scope of this chapter is to identify and list test cases based on requirements defined in Kubernetes based Reference Architecture. This will serve as traceability between test cases and requirements for Kubernetes platform interoperability.

Note that each requirement may have one or more test cases associated with it.

4.2 Selection Criteria

Test cases, tools and their dependencies must be open source. The test cases (or test suite with the test case) as well as the environment needed to run the test should be reproducible by any party following publicly available documentation.

Examples of initiatives (having testing tools, test suites, etc) with test cases which could be used include K8s Conformance, K8s e2e, Sonobuoy, Anuket Functest, CNF Conformance.

4.3 Traceability Matrix

The following is a Requirements Traceability Matrix (RTM) mapping Test Case, and/or Test Case Coverage, to RM and RA requirements – configuration, deployment, runtime.

Test Case Traceability to RA2 Requirements

This section focuses on the test cases covering the requirements in Kubernetes workloads for Kubernetes workloads.

Table 4.1: Traceability to RA2 Requirements

RM/RA Ref	High-level test definition	Test name and project	Priority
ra2.app.001			Must
ra2.app.002			Must
ra2.app.003			Must
ra2.app.004			Must
ra2.app.005			Must
ra2.app.006			Must
ra2.app.007			Must

5 CNF Testing Cookbook

6 Gap Analysis and Development