Maximizing Benefits of CSIT in ONAP Development

Introduction

CSIT ("Continuous System and Integration Testing", even though you could also read that as "Component-Specific Integration Testing") cases in ONAP exist typically for verifying functionalities of a single or few ONAP components in limited testing environments suitable for lightweight installation in local development environment and Jenkins and often rely on simulators. These tests are executed once per day on images from master branch (as well as on maintenance branches of selected previous releases) or whenever the test cases themselves are committed for review. This means that by ONAP's current automated verification procedure *any new bugs that could be caught by the existing CSIT cases are not discovered until the bugs are already merged to master.* In addition, any failures in daily CSIT Jenkins are not sending any kind of notifications or raising tickets or alarms, so the teams are not even getting any automated feedback from the failures that can therefore go days or in worst cases weeks without notice (much less action).

Creating a CSIT Test

Action plan for El Alto and next brainstorming: https://etherpad.opnfv.org/p/CSIT4ONAP

CSIT jenkins page: https://jenkins.onap.org/view/CSIT/

Goal

The ultimate ideal goal for automated verification procedure should be that code review verification jobs would run also the relevant CSIT cases and have a verification vote on the review *before* it is merged. Moreover those CSIT tests should be always executed locally by the developer before committing to minimize the possibility of a failure in Jenkins and shorten the feedback loop to minimum.

Obstacles

The writer of this analysis is not aware of the original reasons to exclude CSIT from review verification jobs (or if including them was ever even considered), but several reasons for keeping them separate can be seen at the current state of ONAP:

- Component code and CSIT cases are in different repos, which means that introducing non-backward-compatible changes leads to egg-chicken problems where neither test case nor implementation modifications can pass their respective review verifications without either one of them being already merged or then by currently unexplored gerrit patch dependency tricks
- Many CSIT cases take a long time to execute, prolonging review verification feedback significantly and most likely causing bottlenecks in Jenkins leading to even further delays in day-to-day development
- Current CSIT tools do not fully support building and testing of local images
- CSIT tests might not be stable enough in all cases to rely on continuous development yet - we can't have random failures delaying or preventing merges and we have to be able to rely on the results
- Every committer would have to be able to at least execute and troubleshoot Robot test cases in order to commit any changes - do the teams currently have sufficient competence?
- Related to the required competence: executing CSIT cases in local development environments currently requires some effort - there are both some generic and project-specific peculiarities that are not sufficiently well documented

Steps Required

CSIT as Part of Automated Code Review Verification

- The most important thing needed to incorporate CSIT cases into code review verifications smoothly is to either put them under the same repo or force the use of gerrit patch dependency functionality so any code commit that changes some already verified functionality (or introduces some new functionality that requires new tests) also can (and must) bring the related CSIT changes
    - See Moving CSIT to project repositories
- CSIT execution and environment setup should be enhanced to be able to build and use images coming from the review branch
- Execution times of the CSIT tests should be shortened to the absolute minimum
    - A major part of this is closely related to the product maturity in general - image sizes (which affects the download time from Nexus) and container startup time optimization could help a lot also in CSIT cases where *significant* amount of time is spent in setting up the test environment

Recently Updated

Policy R9 Istanbul CSIT/External Lab Functional Test Cases
Oct 13, 2021 • updated by Sirisha Manchikanti • view change
Policy R9 Istanbul CSIT/External Lab Functional Test Cases
Oct 11, 2021 • updated by Liam Fallon • view change
Policy R9 Istanbul CSIT/External Lab Functional Test Cases
Jul 15, 2021 • updated by Jim Hahn • view change
Policy R8 Honolulu CSIT/External Lab Functional Test Cases
Jan 21, 2021 • updated by Jorge Hernandez • view change
Policy R6 Frankfurt CSIT/External Lab Functional Test Cases
Apr 13, 2020 • updated by Ajith Sreekumar • view change
Policy R6 Frankfurt CSIT/External Lab Functional Test Cases
Apr 08, 2020 • updated by Jorge Hernandez • view change
Policy R6 Frankfurt CSIT/External Lab Functional Test Cases
Apr 08, 2020 • updated by Pamela Dragosh • view change
Policy R6 Frankfurt CSIT/External Lab Functional Test Cases
Oct 24, 2019 • updated by Ali Hockla • view change
Creating a CSIT Test
Apr 25, 2019 • updated by Lasse Kaihlavirta • view change
Policy R4 Dublin CSIT/External Lab Functional Test Cases
Apr 25, 2019 • updated by Jim Hahn • view change
Policy R4 Dublin CSIT/External Lab Functional Test Cases
Apr 19, 2019 • updated by Pamela Dragosh • view change
Policy R4 Dublin CSIT/External Lab Functional Test Cases
Apr 16, 2019 • updated by Bilal Anwer • view change
Policy R4 Dublin CSIT/External Lab Functional Test Cases
Apr 12, 2019 • updated by Ali Hockla • view change
Policy R4 Dublin CSIT/External Lab Functional Test Cases
Apr 11, 2019 • updated by Ram Krishna Verma • view change
Creating a CSIT Test
Mar 13, 2019 • updated by Martin Klozik • view change

        - Building a new image for every new review patch would make this optimization even more crucial
    - Reducing all kinds of test-specific retries and waiting times to absolute minimum *but no more than that*
        - Note that the other side of this coin is test stability in different environments with varying resources, so special care also needs to taken to ensure that timeouts are not *over*optimized
    - In the long run this requirement is also at direct odds with the requirement that every integration-level bug correction and new feature should be verified in CSIT (if you didn't have this requirement before, you have it now 😃) - eventually the total mass of things that need to be covered in CSIT will (or at least should) grow too big to be executed in every review verification
        - Streamlined set of critical CSIT tests executed in review verification may have to be separated from full regression set that would be executed only on merged code
- CSIT tests should be reliable and working
    - non-problematic code changes should always pass on the first attempt without needing to retry them a couple of times just in case they failed for some unrelated random causes
    - people are less likely to just ignore and override negative CSIT votes if they know failing CSIT is an indication of a real problem in the commit

Manual CSIT Verification on Local Development Environment

- Developers should always verify their changes with CSIT locally (and sometimes also enhance the related CSIT suite) before committing - automated verification in Jenkins should not be considered a development tool
    - Many of the action items listed above apply also here, but will be repeated
- Tools for executing the CSIT cases in local development environments should be enhanced to enable easy deployment of local test images as part of the test environment setup
    - Currently there is partial support for this at least in some docker_run scripts, but this support does not extend to run-csit.sh and is not sufficiently documented
- Documentation of CSIT execution in local development environments should be up-to-date, sufficient and working for every project at all times so developers can actually use it to their advantage
    - Part of the problem is that there are project-specific requirements that need to be known - integration team should try to consolidate this and lay down some common requirements and, on the other hand, provide common ways of doing things that teams have had to solve for themselves until now
        - Concrete first step to remedy this is to go through all existing projects, see and document what it actually takes to make them pass, and see if there are any common improvements to be made (this can be driven by integration but needs obviously support from project teams)

Quality Assurance While CSIT Remains Outside Review Verification

- Project teams should check their CSIT Jenkins jobs *every day* and treat any failures as showstoppers
    - It is ***critical*** that failures are at least analysed ***within 30 days***, because that's how long Jenkins stores the results. After 30 days you can no longer find out via Jenkins what changes broke the build!
    - integration team could also benefit from checking these and making sure teams are alerted to any failures - in longer term this should save final integration phase troubles and keep integration team generally up to date on the overall state of the CSIT suites and therefore (at least ideally) of the ONAP project
- The importance of manual CSIT execution by developers before committing (and all actions making that as easy as possible) is emphasized since it is currently the only gatekeeper before code merges