OPNFV C release planning activities

Release Structure for Colorado (Proposal)
The Colorado release will be framed around scenarios. Colorado will include all scenarios which are available and ready by the Colorado time frame.


Scenario-based release coordination: Background and Objectives

- **Scenario definition:**
  - Within OPNFV a scenario is a "deployment of a set of components and their configuration".
  - Scenarios have a clearly identified owner and scenario leader who serves as the main point of contact. The scenario owner would typically be a project or a set of projects that all contribute features to the assembly of components that constitutes the scenario. A scenario could also be owned by a single project (that might even be the sole purpose of a project).

- **Scenario ownership:**
  - Scenarios have a clearly identified owner and scenario leader who serves as the main point of contact.

- **OPNFV test components and features at system level**
  - This is a unique value OPNFV provides to component/feature developers, who can typically only run unit tests.

- **OPNFV scenarios provide rapid feedback to developers**
  - The faster the results of a scenario become available, the higher the value to a component developer.
  - Providing system-level test-results quickly to (upstream) developers, allows for more rapid innovation and increases the relevancy of OPNFV to developers of components.

- **Accommodate for the fact that not all scenarios are created equal (some are complex, some are simple)**

Scenario-based release coordination: Approach

- **Per scenario schedules**
  - Not all scenarios are of the same complexity: Allow for per scenario schedules; allow scenarios to define their own schedule.
  - Frequently (once per month) make completed scenarios available (i.e. publish results of deployment and testing).
    - *Note: This is confusing, to whom is it published and to whom is this release targeted from a scenario every month? The monthly scenario schedule seems unnecessary, scenario's can be available when they are available without needing to establish a schedule.*
  - Scenario schedules work in parallel to macro OPNFV release schedule.
  - Completed Scenario fold into the next OPNFV release: All scenarios which are "interested" in joining a release and are "ready" (i.e. scenarios deploy, test results are accepted, documentation is available) participate in a release.
    - Scenarios which participate in a release commit to also maintain the released status (i.e. create a stable branch and maintain the branch until the next release)
      - This includes aligning with the maintenance planning for all associated and dependent projects, (installer, test project release and maintenance plans etc...)
      - "Experimental scenarios": Scenarios can choose to not participate in a release (e.g. scenario is to only provide feedback at a given point in time to upstream project and does not intend to maintain a particular composition longer term. This could e.g. be the case for a test scenario is being stood up to figure out a particular integration - and once stable, the scenario would be merged with an already existing scenario).

- **Feature complete**
  - Any scenario that intends to participate in a release needs to fulfill the timeline requirements associated with that release.
  - Scenarios that will not participate in OPNFV releases can have their own independent schedules. Though all of them will define the following milestones.
    - All features for a scenario are code complete. Unit tests are successful. For scenarios which participate in a release, the "feature complete" milestone can be a global OPNFV release milestone.
Installation ready: Features and components for a scenario can be deployed using one of the installers. Integration into at least one of the installers is complete. For scenarios which participate in a release, the "installation ready" milestone can be a global OPNFV release milestone.

Test ready: Test infra and Test cases for system level testing of features available and code complete.

Integration ready: Combination of components in scenarios work "as designed"

Release ready: Test runs complete, documentation ready etc.

### OPNFV releases and scenario availability

- An OPNFV release consists of a set of scenarios which are "ready for release".
- The Genesis project defines a common baseline for install-features and user-experience for a particular release. Through the Genesis project a common denominator across different deployments for a particular release is ensured.
- OPNFV releases have a set of milestones - some of which they share with scenarios which provides some level of synchronization between scenario editions and OPNFV releases. Note that a complete synchronization of milestones isn’t desirable, because it would lead to global synchronization and thus the slowest scenario or component would determine the overall speed (Brahmaputra already departed from a monolithic, fully synchronized release process). Global milestones could include:
  - Release planning complete
  - Scenarios definition (that desire to participate in the release) complete
  - Features code freeze
  - Installer code freeze
  - Test tools code freeze (differs from "test ready" which freezes the unit/system tests)
- Milestones like "test ready", "integration ready", "release ready" are per scenario milestones.

### Release Schedule Discussion (needs updating)

NOTE: The following needs to be updated to reflect individual scenario releases and how they fold into OPNFV releases.

#### 1) Set release milestones

We need to quickly establish our key milestones for C-Release in order that projects are able to establish their scope and intentions for the release.

<table>
<thead>
<tr>
<th>MS#</th>
<th>Timeline</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project planning start</td>
<td>8/March</td>
<td>Projects indicate intent to participate in the release</td>
</tr>
<tr>
<td>Planning complete</td>
<td>8/April</td>
<td>Close project inclusion and plans are established</td>
</tr>
<tr>
<td>Feature Code freeze</td>
<td>31/Mai</td>
<td>Close feature project development phase - freeze</td>
</tr>
<tr>
<td>Installation Code freeze</td>
<td>21/June</td>
<td>Installer/plugins code complete</td>
</tr>
<tr>
<td>Test Code freeze</td>
<td>28/June</td>
<td>Test infra and scenario tests complete</td>
</tr>
<tr>
<td>Integration ready</td>
<td>26/July</td>
<td>Scenario integration complete</td>
</tr>
<tr>
<td>Release</td>
<td>16/August</td>
<td>Release</td>
</tr>
</tbody>
</table>

#### 2) Establish milestones expectations

With clear milestone dates we should work to establish the expectations on project for those milestones. The intention of this work is to ensure we have key activities done during the project work and not after code freeze.
MS0 : Project planning

- Activities to be performed during this phase:
  - State your intention to participate in the C-Release
  - establish release plan wiki /projects/<projectame>/releaseplan
  - Define impacts on other projects:
    - installers
    - test projects
    - documentation
    - Infrastructure (pharos/CI)
  - Include estimated time and resource plans
  - Project plans in place in Jira (at the story level and cross project)

MS1 : Planning complete

- Project must
  - Have coordinated schedules with other projects and captured cross project dependencies in Jira
  - Have a concrete project time plan in place that accommodates dependencies and resource availability
  - Dependencies must include upstream dependencies impacting ability to execute on future milestones

- Activities to be performed after this milestone
  - Development of feature code
  - Integration in, or establishment of, scenario's
  - Test case development

MS2 : Feature Code freeze

- Feature project must
  - Be code complete.
  - Have completed unit test case development
  - Have documentation in place for pre-release verification

- Activities to be performed after this milestone
  - Begin release readiness activities and verification.
    - Stable branch labels can be applied from here (project to decide when to pull the stable branch).

MS3: Installation Code freeze:

- Installers code complete / code freeze
- Features successfully deploy expected deliverables on a pharos lab
- Feature/component installs with target installer(s), i.e. plugins or similar for the installers are ready.

MS4: Test Code freeze

- Test infra and Test cases for system level testing of features available and code complete.

MS5: Integration ready:

- Combination of components in scenarios work "as designed".

MS6 : Release

- Project must
  - All frozen features and scenario's have achieved release readiness.
  - All stable branch labels in place, release labels available on the branch.

- Activities to be performed after this milestone
  - Clear plans for release support according to the release stable processes

3) Definition of the development process

Iterative improvements
Our development process must include alignment on key items that we want to improve from the previous release. That can be filled after the retrospective once each project are has it's list of priority improvements. Alignment to key project improvements needs to be implemented as part of the participation in the release activity.

Development processes
This may be a monthly cycle of development and scenario readiness. If such a process is used for any given release the scenario will be available from Master until such time as MS3 of the release is passed and the scenario can be migrated to stable.

This may be as in Brahmaputra where each project implemented an own project development practice. If such a process is used there needs to be clear directives on milestone deliverables that the projects are expected to align toward in order to better manage the development effort involved in providing platform feature availability.

4) Related Links
release-c-brainstorming
OVSNFV Colorado Release Planning
More…