Landing

Lab as a Service

Lab as a service intends to provide a cloud based build and deploy environment for executing Jenkins verify and merge jobs and as a staging area for scenario deployment prior to running on our bare metal Pharos infrastructure. Intended as an infrastructure and design resource the lab as a service initiative should both streamline our development and jenkins workflows and reduce the burden on Pharos labs.

Requirements for LaaS ... Project Planning

JIRA task for "Utilize Virtual Environments for CI and developers" ...

Unable to render Jira issues macro, execution error.

LaaS Meetings

- No regularly scheduled meetings
- Issues can be brought to the weekly tech discuss
- One-off meetings can be planned when necessary

LaaS Project Page

Accepted project proposal can be found here: Lab as a Service

LaaS Motivation

Towards serving developers in a better way

- OPNFV scenarios are too heavy to run on a typical laptop
  - Developers require access to at least one beefy server for a virtual deployment to develop on
  - This “beefy server” is not always easy to find for an interested contributor
- Developers which are interested to advance features or test not always have the expertise to install the scenario they desire to enhance
  - Getting a deployment as development environment ready requires significant cycles which a developer may not be able/ready to invest
- Community-level/joint development requires open access to a lab/installation
- Current situation
  - Only very limited number community openly accessible servers available
  - No “turn key” OPNFV scenario installations available as lab as a service / sandbox

Objective for LaaS

- Offer openly accessible server resources to developers
- Offer readily installed scenarios, per the choice of the developer

What is OPNFV LaaS

- LaaS is a cloud/bare-metal environment, providing automatic
  - provisioning
  - deployment
  - delivery
  - decommissioning
- of resources, enabling the
  - creation of one click development environment, supporting different use cases
  - harmonization across environments used by developers
  - ability to bring up any installer/scenario together with the other components of the stack
  - possibility to provide end/new users to try out OPNFV without much time and effort
- reducing the burden on OPNFV Infra/Pharos and developers, end users

OPNFV LaaS initial use-cases

Use Case	Description
Snapshot deploy	Spawn a deployment from snapshot
Initial master deploy	Deploy certain OPNFV scenario from scratch from master using an existing artifact
Initial stable deploy	Deploy certain OPNFV scenario from scratch using a released OPNFV version
Build and deploy	Build the artifact and deploy certain OPNFV scenario for the given patch
Deploy with addons	Deploy certain OPNFV scenario and provide additional scenarios to deploy/develop other components of the stack such as ONAP
OPNFV for ONAP developer	Deploy and integrate OPNFV scenario and ONAP instance for developer use
OPNFV for ONAP X-CI	Deploy and integrate OPNFV scenario and ONAP instance for X-CI
OPNFV+ONAP CI/CD	Deploy and integrate OPNFV scenario and ONAP instance for full CI/CD/testing
Deploy OS	Provide a machine with OS installation only

See LaaS scenarios for a detailed description of LaaS use cases and associated requirements.

LaaS - Number of Servers

For specific information about hardware and capabilities, see the pages for the participating companies below:

- Lab-as-a-Service at the UNH-IOL

LaaS Flow Proposal

Use Case: OPNFV Developer Access to On-Demand POD

Basic workflow can be seen on below diagram (image from LaaS.vsdx).

blocked URL

Here is the workflow in detail

1. Requester: Logs in to dashboard and issues the request. The request should contain
   - Type of the environment: Deployment from snapshot, deployment from scratch, or an basic OS installation
   - Installer/scenario details: What scenario with which installer should be deployed
   - Installer/scenario version: Should it be built and deployed for a patch or from master and so on
   - Requester information: SSH public key, mail address
   - How long the resource is needed: 24 hours, 48 hours
2. Dashboard: Triggers a Jenkins Job and Updates the status of the request from New to Queued
3. Jenkins Job - Deployment:
   a. Updates the status of the request from Queued to In Progress when the job starts running for real
   b. Updates the status of the resource from Idle to In Use
   c. Does the deployment using the details from the request
   d. Adds the SSH keys of the developer
4. Jenkins updates the status of the request from In Progress to Complete - Failure or Complete - Success depending on the result
5. Sends the deploy complete notification to requester, including access and credentials info for VPN to the OPNFV POD
6. Requester: Logs in and uses the POD

(outside this flow)

1. Jenkins Job - Cleanup:
    a. Runs periodically and wipes out the deployment, removes the keys and so on from the resource
    b. Updates the status of the resource from In Use to Idle

This work can also be done in phases and the first phase could be creating basic Jenkins jobs to put the logic in place.The job could just echo Hello World initially.

Once the Jenkins job logic is in place, a basic form on dashboard can be created to trigger the job and the request details are passed to the job.

And so on.

ONAP Integration Proposal

Below are use cases for OPNFV+ONAP lab integration. Discussion on these use cases are continuing in the OPNFV Infra Working Group Meeting. Aspects under discussion include:

- What use cases we should focus on. Examples currently considered include
    - ONAP Developer Access to On-Demand OPNFV+ONAP POD (below)
    - ONAP Developer Access to OPNFV POD with ONAP Deployed (below)
    - ONAP X-CI
- How we would resource development of the dashboard and other functions that would enable this
    - At least the two developer-focused use cases would require portal updates. We would need to identify the discrete steps the portal could support through the developer experience, identify any minimal/initial functions we should develop, and find someone to do it.
- Whether in some cases we should integrate ONAP as a deployed component in an OPNFV POD (e.g. as earlier considered by the Opera project). This is per the use case ONAP Developer Access to OPNFV POD with ONAP Deployed (below). We would need to determine the impact on
    - Hardware requirements: our current understanding is that this is at least 150GB for ONAP VMs.
        - Need someone to experiment to verify that with the Nov '17 release of ONAP as it nears.
    - Deploy job duration, when including ONAP deployment
        - Do we need to assume use of pre-existing ONAP artifacts (not dynamically built)?
- Whether it will be possible to deploy only specific ONAP components as needed for a particular test or scenario
    - This will be considered as part of the proposed Auto project ref'd below
    - Someone needs to verify feasibility of this
- What labs would host the LaaS components for ONAP
    - This should be discussed in OPNFV-ONAP collaboration e.g. thru the OPNFV Infrastructure team and the ONAP Open Lab subcommittee.
- How would OPNFV and ONAP labs be interconnected via VPN (assumed essential for lab security).
    - Someone needs to experiment with dynamic lab-to-lab connections over VPN.

The goal is to come to a pretty good understanding of those aspects before, as a community, OPNFV reaches out to ONAP with a specific (concrete) proposal. In the meantime however, as we address those questions, OPNFV and ONAP members continue to work on them tactically through:

- Collaboration of OPNFV and ONAP through the OPNFV Infrastructure team and the ONAP Open Lab subcommittee.
    - Helen Chen is coordinating the OPNFV-ONAP discussions on this for ONAP, and Bryan Sullivan for OPNFV.
- As of the OPNFV Summit, the Opera project team has set a goal to continue that ONAP deployment focus as described in the video of the talk: **ONAP Integration with OPNFV via Opera - Yingjun Li, Chengli Wang**
- ONAP-Automated OPNFV (Auto) is a new project proposal which will focus on integration/verification of specific ONAP components over time, as compared to Opera's goal of developing OPNFV-installer supported scenarios that can deploy and verify ONAP as a whole.

Use Case: ONAP Developer Access to On-Demand OPNFV+ONAP POD

Basic workflow can be seen on below diagram (image from LaaS.vsdx).

blocked URL

Preliminary assumptions about the OPNFV LaaS POD resources expected for typical ONAP developer/CI use cases:

- a partially or fully allocated Pharos compliant POD server, to run a virtual OPNFV scenario deploy and deploy VNFs for testing
    - typical small/medium VNF (e.g. vRouter, vFW, vLB, vDNS, vPE, vCE): 50% of a server
    - typical large VNF (e.g. vEPC, vIMS): 1 full server
- 25% of ONAP developers spending avg 4 hours per day testing with the OPNFV LaaS resources (as needed for specific NFVI features or to work with specific OPNFV distros... the rest of ONAP devs are fine with generic/public clouds for testing VNFs)
    - 5% of ONAP developers active on OPNFV LaaS resources at any one time (assuming even timezone-distribution)
    - thus for ~150 ONAP devs, <10 active devs at any one time
- with the assumptions above, 10 Pharos compliant LaaS POD servers on avg consumed for ONAP devs

Here is the workflow in detail:

1. Requester: Logs in to dashboard and issues the request. The request should contain all of the info from the "OPNFV Developer Access to On-Demand POD" use case plus
    a. ONAP scenario details: What scenario should be deployed (ONAP "scenarios" are still being defined - for now it's assumed there is only one)
    b. ONAP/scenario version: Should it be built and deployed for a patch or from master and so on
2. Dashboard: Triggers a Jenkins Job and Updates the status of the request from New to Queued
3. Jenkins Job - Deployment:

    a. Updates the status of the request from Queued to In Progress when the job starts running for real
    b. Updates the status of the resource from Idle to In Use
    c. Does the deployment using the details from the request
    d. Adds the SSH keys of the developer
4. Jenkins Job - Deployment:
    a. Using ONAP POD management APIs (TBD), allocates an ONAP POD
    b. Establishes a VPN connection between the OPNFV POD and the ONAP POD
    c. Triggers a Jenkins Job for ONAP install and updates the status of the request to Pending ONAP
    d. Configures ONAP for access to the OPNFV POD VIM (VIM address, credentials)
5. ONAP: registers with the VIM using the credentials provided
6. Once ONAP access to the VIM has been verified, Jenkins
    a. Updates the status of the request from In Progress to Complete - Failure or Complete - Success depending on the result
    b. Sends the deploy complete notification to requester, including access and credentials info for VPN to the ONAP and OPNFV PODs
7. Requester: connects to the ONAP POD via VPN and uses it
8. Requester: optionally connects to the OPNFV POD via VPN and uses it

Use Case: ONAP Developer Access to OPNFV POD with ONAP Deployed

Basic workflow is the same as "Use Case: OPNFV Developer Access to On-Demand POD" with addition that the user can select to have ONAP deployed on the OPNFV POD as well. Analysis of the resource requirements for this use case is underway. Below are some source links:

- https://wiki.onap.org/display/DW/Minimal+Assets+for+Physical+Lab
- https://wiki.onap.org/display/DW/ONAP+instances+on+vanilla+Openstack
- https://wiki.onap.org/display/DW/Resources+and+Sizing+on+vanilla+Openstack
- https://wiki.onap.org/display/DW/Tiny+OpenStack+Lab+for+ONAP
- https://wiki.onap.org/display/DW/Tutorial%3A+Configuring+and+Starting+Up+the+Base+ONAP+Stack
- https://wiki.onap.org/display/DW/Tutorial%3A+Configuring+and+Starting+Up+the+Base+ONAP+Stack?preview=%2F1015882%2F3245312%2FONAP+DEMO+SIZING.pdf
- https://wiki.onap.org/display/DW/Tutorial%3A+Configuring+and+Starting+Up+the+Base+ONAP+Stack?focusedCommentId=3245456#comment-3245456

Completed Trials

The information below is a record of earlier trials of the LaaS concept.

Ravello Trials

> **Evaluations to use Ravello for OPNFV CI have been parked due to technical limitations and no more evaluations will be done until after C-release is out. (stability of nested virtualization)**

Ravello has been trialled as a candidate for supporting nested virtualization deployments of OPNFV in a cloud environment. The environment was found to be unsuitable for OPNFV scenario deployments with a significant amount of instability, slow execution times, and crashes due as we understand to the highly nested environment and HVM layer.

IPMI Support

1. Ravello doesn't natively support IPMI (as a service) today – but that may change in the future
2. It is possible however, to workaround this even today by having an open IPMI server as a part of the 'Ravello Application/Blueprint' that can talk to Ravello's REST APIs

Rackspace Trials

> **Evaluations to use Rackspace for OPNFV CI have been parked due to technical limitations. It is also expensive.**

Trials are starting to evaluate the ability to run nested virtualization deployments in Rack Space. The rackspace environment provides a less nested environment with the ability to potentially integrate directly through Jenkins. To be updated...

Other Options

Evaluations to use GCE, EC2, and Azure for OPNFV CI have been parked due to technical limitations. (lack of nested virtualization support)

Google Compute Engine, Amazon EC2, and Microsoft Azure have been evaluated as well. But these providers do not expose CPU virtualization features so they deemed to be not viable for OPNFV at this time.

LaaS Hardware

A key objective of LaaS is to create a development environment on the fly, whether it is vPOD, Pharos baremetal POD or and OPNFV-ONAP POD. LaaS will avoid dedicating a particular server to a particular role, but rather allow for flexible allocation following demand. As a consequence, LaaS assumes a single type of server which can be used for a vPOD, a Pharos POD, or an ONAP POD.

Server Hardware (x86 based)

- Intel XEON E5-2699 v4 2.20 GHz, 145W, 22C, 55MB (high number of cores (22), well suited to run VMs) - https://ark.intel.com/products/91317/Intel-Xeon-Processor-E5-2699-v4-55M-Cache-2_20-GHz
- 512 GB of DDR4 ECC RAM
- 2 x 10G DPDK-compliant NICs (e.g. Intel X710: 4 port 10GE) to allow for dedicated ports for admin / public / tenant / test networks
- 0.9 TB of disk (2 x 480G SSD)
- Lights-out-management (must supply IPMI compatible solution)

Server Hardware (ARM based)

- Cavium ThunderX ARM processor (64bit ARMv8 architecture, 48 cores per processor, 2.0GHz BGA 2601), e.g. ASA ASA9103-48C-TX
- 256 GB of DDR4 ECC RAM
- 2 x 10G and DPDK capable NICs (4 port 10GE each); In case of on-board NICs (like e.g. with ThunderX), a minimum of 4 ports of 10G should be supplied
- 0.9 TB of disk (2 x 480G SSD)
- Lights-out-management (must supply IPMI compatible solution)

Network Switches

Switches should support 10/25/40/100Gbps ports to future proof the setup (switch hardware PODs from 10 Gbps to 25 Gbps, 40 Gbps or 100 Gbps to also support performance testing), e.g. Cisco Nexus 92160YC-X.

Assuming the above number of servers: 38 x86 based servers, and 14 ARM based servers and further assuming that a maximum of 6 ports per server would be wired up to the switch, a total of 312 switch ports would be needed.
This means that just for connecting the server-ports (not counting ports on uplink/spine switches), a total of 7 48-port switches would be needed.

Lab-Hosting options / metal as a service

- CENGN
- University hosting (e.g. UNH)
- Bare-metal hosting: https://www.packet.net/