Virtual Labs Hosting Platform

Thirumal Ravula
VLEAD, IIIT Hyderabad
India

Venkatesh Choppella
IIIT Hyderabad
India

***Abstract*—Virtual Labs is a consortium whose agenda is to build and ensure the availability of virtual labs [10]. This effort is funded my Ministry of Human Resources and Development (MHRD), India. The consortium consists of 11 educational institutes spread across India. The consortium created a group VLEAD – Virtual Labs Engineering and Architecture Division – to undertake central platform engineering (CPE). This group is based from IIIT Hyderabad with the primary intent to build platforms and services which unburden and allow the authors of experiments to solely target the delivery of pedagogy. One such effort towards the central platform is the hosting platform. The labs developed by the different institutes are hosted on this platform. This paper talks about the evolution of the hosting platform, a vendor agnostic cloud solution to host over 150 virtual labs. The cluster that makes up the hosting platform is a union of machines serving virtual labs, machines providing basic infrastructure support for routing, domain name resolution, firewall rules configuration, monitoring etc. and machines providing other services like auto deplooyment, analytics and feedback.**

I. Introduction

When Virtual Labs as a project started, each professor responsible for building a lab composed of various experiments chose their own technology stack. This approach brought forth each lab as a unique web application with no shared UI or a common dependency list. Each experiment in a lab contained a simulation or an interactive artefact apart from text content that explained the context of the experiment. The entire lab is viewed on a browser.

Though the lab is viwed from the browser, no assumptions are made about the server side processing required for each lab. Some of them were entirely javascript, when once pulled onto the browser required no other processing from the server, while others required server side processing like a request to the server to process an image in an image processing lab [7].

Some of the labs are built using the php stack, while few others used a database. These web applications did not have a concept of sources and what constituted a runtime. This implies there is no build process defined for each of the labs.

Therefore, what begets is the question of how these myriad and independent labs are built and hosted for public access. Also, how would a hosting platform allow continous evolution of these labs.

In this paper, we discuss the challenges in conceiving the central hosting platform capable of serving over 150 virtual labs and meet the demands of millions of users. This paper looks at variuos phases in such an evolution of a platfrom that maps a url of the sources of a lab to a hosted url. As a starting point, a build interface is defined and each lab is structured to implement this interface. Each lab explicitly defines the runtime environment that constitute both the software and hardware dependencies as a deployment specification. Later, we talk about the development of the software service - auto deployment sercice (ADS) to automate the deployment of the labs. This service reads the deployment specification, creates the machine that satisfy hardware requirements and installs all the software dependencies. Using the build interface, the lab is built and deployed on this machine. This forms one part of the hosting platform.

The other aspect of the hosting platform is its abilty to maintain this cluster of machines, each one serving a lab as an application. More nodes are added to this cluster primarily to allow a single public IP to provide fully qualified domain name to each of the lab. Monitoring, security, logging are the other services provided by the platform. Such a cluster is provided the ability to spawn off either on a developer's machine, or on an on-premise cloud or on a commercial cloud.

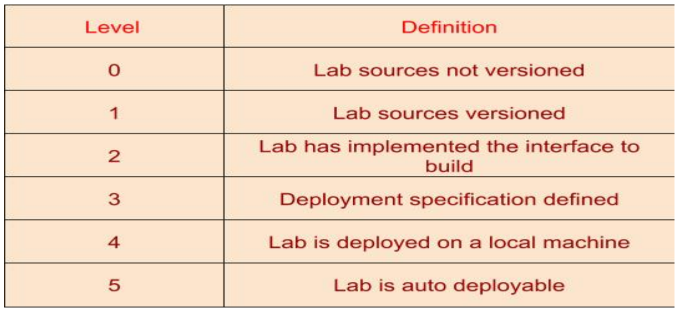
This paper elaborates on 1. deployment specification of a lab, 2. auto deployment service that maps a source url to a hosted url and 3. the architecture of the cluster that makes up the hosting platform, in the ensuing sections.

II. Integration Levels

Each virtual lab is an individual web application that has a life independent of other virtual labs. These virtual labs are built at various institutes that constitute the consortium and cover various engineering and science disciplines. Each lab initially was built as a prototype by either reaseach assistants or interns guided by the professors at these institutes. The built labs were hosted for public consumption with indiviadul public IPs. When a lab became unavaible, there was no central triage to resuscitate the failed ones. Moreover, the hosting was adhoc in nature. It is with the intent to professionally manage the hosting of virtual labs that a central platform team is initiated.

The first task of the CPE team was to define the integration levels that apply to each lab. Most of the sources were not even version controlled. The spread of levels is from 0 - when a lab is not version controlled; to 5 - when a lab is auto deployable. Each level and it's import is specified in the figure-1.

The idea is to move each lab from level 0 to level 5. Since the *Virtual Labs* project is publicly funded, the code is licensed under *AGPL 3.0*[6]. Therefore, a public organization *vlead*[11] is created on github[4] to ensure all the labs are open source. A workshop was conducted at IIIT Hyderabad where atleast one representative or lab integrator from each



Level	Definition
0	Lab sources not versioned
1	Lab sources versioned
2	Lab has implemented the interface to build
3	Deployment specification defined
4	Lab is deployed on a local machine
5	Lab is auto deployable

Fig. 1. Integration Levels.

institute participated. The wokshop held over a week dealt with helping each integrator in moving a lab from integration level 0 to level 5. Lab Integration Kit[8] is released prior to the workshop. This kit contained all the necessary documentation and software to enable a lab to move it to the integration level 5. With the acquired hands-on knowledge, the integrators would lead the team at their respective institutes and help other lab authors in achieving the same goal as at the workshop.

A. Version Control

The first step of publishing the sources of a lab as a git[3] repository moves that particular lab to level 1. Once this is achieved, the next step is to provide the interface to build the experiment. As a prerequite to this, the contents of the lab are set up to fit a predefined directory structure as shown below in the listing.

```
Lab
—— license.org
—— README.md
—— scripts
      —— labspec.json
      —— ..
—— src
      —— makefile
      —— ..
—— test
      —— ..
```

Listing 1. Lab Structure

B. Build Interface

All the sources are of a lab sit in the **src** directory along with a makefile that builds the lab. The target *make -k all* build that lab and creates a *build* folder and the files required during the runtime of a lab are copied here. This is the folder that gets hosted. This *makefile* defines the build interface of the lab. Once this is accomplished, the lab achieves the integration level 2.

C. Deployment Specification

A deployment specification facilitates the lab author to specify the hardware and software requirements to build and host a lab. The lab author specifies both the build and runtime requirements. A sample specification is listed below.

```
{
  "lab": {
    "integration_level": "3",
    "build_requirements": {
      "platform": {
        "arch": "i386",
        "build_steps": {
          "build": ["make -C ../src"],
          "installer": [],
          "os": "ubuntu",
          "osVersion": "12",
        }
      }
    }
    "runtime_requirements": {
      "platform": {
        "arch": "i386",
        "hosting": "dedicated",
        "installer": ["sudo apt-get instll apache2
        "lab_actions": {
          "backup": [],
          "clean": [],
          "init": ["cp -r ../build/* /var/www/"],
          "shutdown": [
            "service apache2 stop"
          ],
          "start": [
            "service apache2 start"
          ],
          "memory": {
            "max_required": "2gb",
            "min_required": "256mb"
          },
          "os": "ubuntu",
          "osVersion": "14",
          "storage": {
            "min_required": "10gb"
          }
        }
      }
    },
    "template": "1.0"
  }
}
```

Listing 2. Deployment Specification Sample

The knowledge of the author is building and hosting a virtual lab is brought out into a machine readable specification. The ability to build a virtual lab using an interface and the specification of build and runtime requirements allow a deployment service to automate the process of building and hosting an experiment. In the section *build requirements*, the author mentions the command to build a lab and also the enviroment in which to build. The section *runtime requirments*, specifies the steps to create an environment where the lab is hoted. The minimum requirements of resources in terms of memory and storage are mentioned. The steps to install specific services are also mentioned. When the deployment specification is error free and version controlled, the lab moves to integration level *3*.

D. Manual Deployment on developers machine

Once the deployment specification is ready, the author or the integrator, manually executes the steps in building the virtual lab, creating a virtual machine with the specific requirements, installing the software dependencies and finally hosting the lab on this machine. This phase is testing is testing the deployment specification. Once the virtual lab is successfully hosted by the manual process of executing the steps defined in the deployment specification, the stage is set for the auto deployment of the same virtual lab. At this point the lab attains the integration level *4*. During this stage, the author might tweak the deployment specification based on the results of the manual execution of the steps. The integration level is also updated in the specification.

E. Auto Deployment of the lab on the integrator's machine

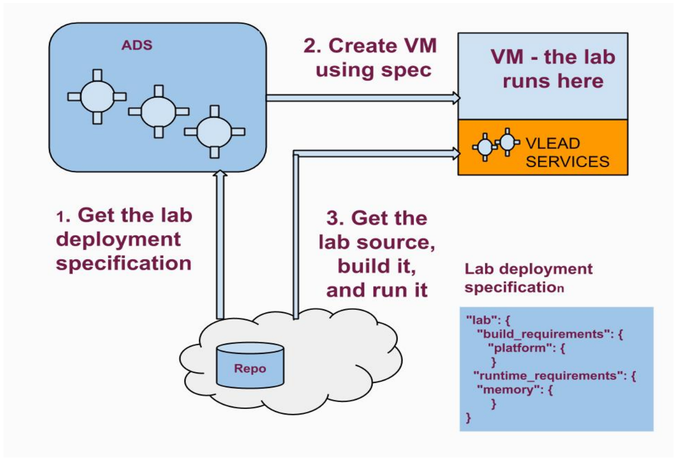
The lab ingeration kit (LIK) contains the auto deployment serice (ADS) and the documentation on how to auto deploy a lab on the integrator's machine. The interface to the service is *deploy-lab ¡source-url¿*. The service deploys the lab and provides an IP of the machine where the lab is hosted upon succesful deployment. When successful and lab is accessible, the lab reaches the integration level *5*. The ability to package the service allow integrators to work independently on a lab to allow it to reach the intergration level *5*. ADS is elaborated in the section in more detail.

III. Auto Deployment Service

Auto Deployment Service (ADS) depicted in figure-2 is a composition of micro services and as the name suggests automatically deploys a virtual lab on a virtual machine. It maps a source url to a hosted url while basing the mapping on a deployment specification present in the sources of a virtual lab.

Auto Deployment Service is made up of four different micro services: 1. front end that provides a web gui to enter the source url of the lab to be deployed, 2. adapter serivice, that picks the right adapter to create the VM, 3. lab manager, that co-ordinates the installation of the lab on the newly created virtual machine, and 4. the controller that orchestrates the entire deployment using the earlier services.

For each virtual lab, a new virtual machine is created. On this machine the lab is built, software dependencies are installed and the built lab is copied to the web server's serving location. The specification and requirements for the VM are read from the runtime section of the deployment specification. The virtual machine is created with the same resources as mentioned in the specification. One virtual machine per lab ensures each lab is seggragated from others and functions in its own world. Their is a clear demarcation of resoruce usage and removes the chances of a rogue lab eating into other labs'


ADS
2. Create VM using spec
VM - the lab runs here
VLEAD SERVICES
1. Get the lab deployment specification
3. Get the lab source, build it, and run it
Repo
Lab deployment specification

```
"lab": {
  "build_requirements": {
    "platform": {
    }
 "runtime_requirements": {
  "memory": {
    }
}
```

Fig. 2. Auto Deployment Service.

resources. A single bad lab cannot nullify the sanctity of other labs. In the next few sections, the architecuture of ADS is laid out.

A. Virtualization

The auto deployment service uses the OpenVZ[13] container virtualization for creating new virtual machines to deploy labs. Container virtualzation is operating system level virtualization[18], meaning the virtual machines or the containters or the guest machines share the kernel of the host machine. Full virtualization is another technique where the guest machines do not share the kernel of the host operating system. KVM[5] is an example of full virtualization. Full Virtualization is achieved by the use of hypervisors[17].

Container virtualization allows a secure way of allocating finite resources between distrusting applications without much overhead. The limitation of this technique is that the guest machines cannot run an operating system whose kernel is different from the host machine. The implication of this limitation is guest machines with windows operating system cannot be created. As a design choice, this limitation does not turn out to be a contraint since all the labs are open source and use software stacks that are not proprietary.

B. Creating a machine

The creation of a machine is handled by different adapters. To cater to the different needs of deployment, whether on an intergrator's machine, or on an on-premise cloud, or on a public cloud, these adapters provide the handle to create a virtual machine in these environments. These three environments can be categorized as test, stage and production environments. When the ADS service is set up, a configuration change will allow to choose the adapter. The currently supported adapters allow to create: 1. an OpenVZ container on the same local area network as the host machine sits, 2. an OpenVZ container in a private local area network and 3. an EC2 virtual machine on Amazon Cloud - AWS.

For the intergrators, a VirtualBox[14] virtual machine configured with ADS is supplied. This box is installed on any host

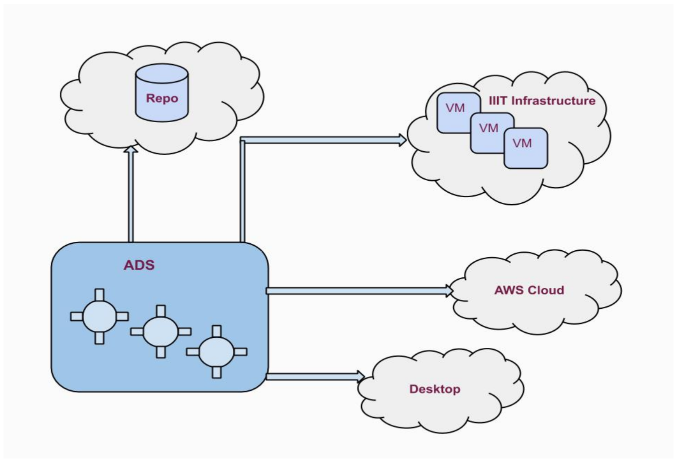


Fig. 3. Different Adapters.

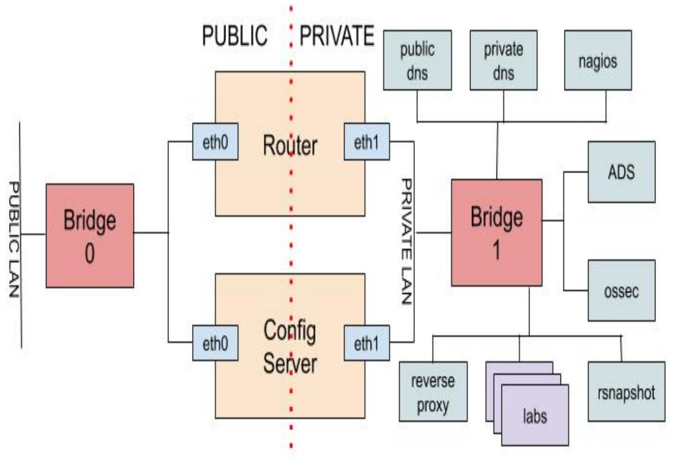


Fig. 4. Cluster Design.

machine running either Linux or Windows operating system. When a lab is deployed using this setup, the new virtual machine hosting the lab is created in the same LAN (figure-3) as the host machine sits. The lab after deployment is accessed with the IP on the same LAN.

Central Platform Engineering uses on-premise machines to create the staging enviroment for the hosting platform. The architecture of the hosting platform is explained in more detail in section-IV. In the cluster that makes up the hosting platform all the nodes except for a few sit in a private LAN. Especiaaly, all the lab machines are in the same private LAN. The second adpater (figure-3) facilitates creation of OpenVZ containers in a private lan.

AWS[2], a commercial cloud is the production environment of Virtual Labs. The aws adapter (figure-3) facilitates creation of an ec2 instance on AWS. The adapter uses the API provided by the AWS to create the virtual machine on the cloud. This API allows to set the parameters of the machine in terms of memmory, disk usage, CPU, etc. Each virtual machine is created with a private IP within a subnet where all the modes of the cluster sit. The mapping of the IP of this machine in a private to a public fully qualified domain name (FQDN) is done by other nodes in the hosting platform cluster.

C. Deployment of a lab

During the creation of the machine from the deployment specification, few services are also installed. The primary service among them is *LabInstaller*, refer step 3 of figure-2. The orchestrator service of ADS, invokes the API - *deploy-lab* - of LabInstaller to deploy the lab whose source url is passed as an argument to the deploy function. This service pulls the lab repository, goes through the build steps in the deployment specification that is part of the source and executes them. After that, it goes through other actions of installing software dependencies and copying the built lab resources to the right location as mentioned in the deployment specification, refer subsection-II-C.

IV. Hosting Platform Cluster

The auto deployment service enables mapping of a source url of a virtual lab to a hosted ip. Once, when over 150 sources are mapped to hosted ips, how would all the labs be available for public access. One of the naive ways to accomplish this is to provide 150 public Ips and register all these IPs mapped to fully qualified names with an internet service provider (ISP), and with more labs getting built and deployed, this solution unravels its infeasibility. Therefore, the solution is to build a robust and secure hosting platform (figure-4) that also allows for monitoring, logging, taking backups, etc.

Another characteristic hosting platform possesses is the portability. The hosting platform has the ability to be configured either on a developer's machine, or on an on-premise cloud or on a commercial cloud. Such a solution not only provides different environments - testing, staging and production; but also untethers the solution from a single commercial cloud vendor. The creation of the cluster that hosts the virtual labs is automated, and with few changes to the configurable parameters, the cluster is deployed in either of these environments.

A. Cluster Architecture

The plaform represented in the figure-4 contains a set of nodes - cluster - and is divided into three different subsets - infra, services and labs. The infrastructure for the platform is provided by the infra nodes. They are the *router, private-dns, public-dns, reverse-proxy and the config server*. The node *router* applies network address translation rules on the incoming and outgoing traffic. The nodes *public-dns* and *private-dns* resolve domain names to a physical IP while the node *reverse proxy* maps http and https traffic to the lab nodes. The node *config-server* is the staging node from which all other nodes - infra and services - are configured. These nodes allow the basic functioning of the cluster.

The services add extra features to the platform. Nodes *nagios, ossec and rsnapshot* provide monitoring, security and backup services respectively to the platfrom. Nagios[12] is an open source monitoring service. The server runs on one of the

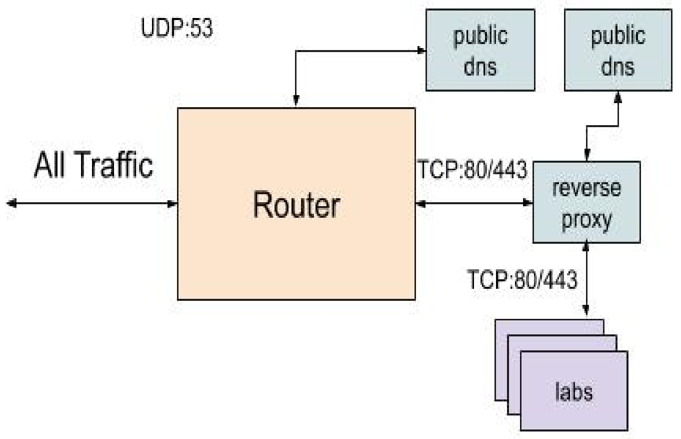


Fig. 5. Traffic Flow.

nodes while client is run on all other nodes of the cluster. The server communicates with the client and provides the monitored statistics via a dashboard. Similarly, OSSEC[15] and rsnapshot[16] work in a similar fashion for intrusion detection and managing backups respectively. ADS is the in-house built service to deploy an application and is used primarily to deploy virtual labs into the cluster. These deployed virtual labs comprise the third set of nodes of the hosting platform cluster.

B. Cluster Network

All the nodes of the cluster, refer figure-4, reside in the private subnet and are not exposed to the external world except for 2 nodes. The cluster's interface to the external world is provided by the node *router*. Node *config-server* also has a public interface, but the purpose is different from that of *router*. Node *config-server* serves as staging node from which all other nodes are configured. A public bridge and a private bridge are created. The public bridge interfaces with the external world. To this bridge, one of the network interfaces of both the *router* and the *config-server* are connected. The private bridge emulates a private LAN. All other nodes are connected to this bridge while the private interface of the nodes *router* and *config-server* are also connected to the same bridge. Both the bridges are software bridges.

C. Traffic Flow

The node *router* is the gateway to the hosting platform cluster. This is achieved by setting up network address translation (NAT) rules on this node. Node *router's* public interface is configured with a public IP and the same IP is registered with the domain registrar for the domain *vlabs.ac.in*. In effect, all traffic (refer figure-5 destined for *vlabs.ac.in* ends up on the public interface of the node *router*. It redirects all the DNS queries identified by the protocol udp and port 53 to the node *public-dns*. The node *public-dns* resolves the domain name to the public IP of the node *router*. Once the domain name is resolved to an IP, http/https requests are made to that IP. This traffic also ends up at the public interface of the node *router*. Another NAT rule redirects this traffic to the node *reverse-proxy*. The node *reverse-proxy* maps the request to the actual node serving the application. It takes the help of the node *private-dns* to resolve domain name to the ip of the lab node that sits in the private subnet. Since every node sits in the private subnet, the cluster's security now becomes a function of securing the nodes that are exposed to the external world.

D. Bootstrapping and Configuration

The cluster that makes up the hosting platfrom is fully automated. This means all the steps in setting up a functioning cluster: the creation of the nodes in the cluster, setting up of the bridges and interfaces, and the confuration of every node, are automated. The two steps of creating the nodes in the cluster and setting up of the bridges and tying the right interfaces to these bridges sets up a cluster where every node is accessible from the other in the private subnet. At this point, the public interfaces are open to all the traffic. The next step of configuration sets up the functionality of each node either infra or service while securing the cluster in a layered fashion.

Bootstrapping is done with a set of configurable parameters, like the number of nodes, private ips of the nodes and the public ips. The config-server is setup during the previous step to have ssh access to all other nodes. This is necessary since the configuration of all the nodes is drven from this node and it acts as the stage to the cluster. Each node is configured to perform its function. The router's firewall rules are set up to allow only dns and http/https traffic through it. The NAT rules are set up to forward the traffic to the right node. On all other nodes, the firewall rules are setup to accept traffic only from the subnet nodes. Key based remote login on the ssh port (22) from the external world is made posible only on the config server. Ansible[1], a configuration tool is used to build the configuration scripts and these are version controlled. Anytime, a change in configuration of a node is needed, the changes are first commited, and later the configuration is re-run to affect the changes. The changes were never made on the node itself by a hack, instead the configuration changes are always made from the config-server.

V. Conclusion

The in-house built hosting platform built totally from open source components has withstood its ground in terms of availability and resisting any attacks. The platform has ensured almost 100 percent availability of virtual labs recording a usage of nealy four lakhs[9]. The deployment service coupled with automated creation of the cluster have helped build a platform that has eased the management of dynamic hosting requests of virtual labs.

The portability of the cluster has helped the developers in quickly deploying their virtual lab in the test environment while allowing the systems team of the central platform engineering to test the new hosting requests in a staging environment paving way for a confident deployment on the production.

By bringing the configurations of all the nodes into files that are version controlled, that are later applied on the nodes to affect the configuration, the files become the repository of knowledge. This was not true before this platform was built when the live machines were the only repositories of knowledge. By affecting this transformation, the hosting platfrom in itself has become another virtual lab.

REFERENCES

[1] Ansible. Ansible, it automation tool. https://www.ansible.com/overview/how-ansible-works.
[2] AWS. Aws. https://aws.amazon.com/.
[3] git. Git version control. https://git-scm.com/.
[4] github. Github portal. https://github.com/.
[5] Red Hat. Kvm. https://www.redhat.com/en/topics/virtualization/what-is-KVM/.
[6] Open Source Initiative. Agpl 3.0. https://opensource.org/licenses/agpl-3.0.
[7] Virtual Labs. Image processing lab. http://cse19-iiith.vlabs.ac.in/.
[8] Virtual Labs. Lab integration kit. https://github.com/virtual-labs/lik/releases/tag/v1.0.0/.
[9] Virtual Labs. Outreach portal, virtual labs. https://outreach.vlabs.ac.in.
[10] Virtual Labs. Virtual labs landing page. http://vlab.co.in/.
[11] Virtual Labs. Vlead organization on github. https://github.com/vlead.
[12] Nagios. Nagios. https://www.nagios.org/.
[13] openvz. Container based virtualization. https://openvz.org/.
[14] Oracle. Virtual box. https://www.virtualbox.org/.
[15] OSSEC. Ossec. https://www.ossec.net/.
[16] rsnapshot. rsnapshot. https://rsnapshot.org/.
[17] Wikipedia. Hypervisors. https://en.wikipedia.org/wiki/Hypervisor/.
[18] Wikipedia. Os-level virtualization. https://en.wikipedia.org/wiki/OS-level_virtualization/.