Contents

Cisco OpenStack Installer

The Havana release represents a significant departure from previous Cisco OpenStack Installers. Where in the past users were required to edit the 'site.pp', a puppet manifest that describes their environment, this has now been changed to a yaml based data model. This provides significantly more flexibility, as a user can now modify almost any aspect of their OpenStack install without needing to know any puppet.

Before you begin

Please read the release notes for the release you are installing. The release notes contain important information about limitations, features, and how to use snapshot repositories if you're installing an older release.

- h.0
- h.1
- h.2
- h.3

Prerequisites for All Nodes

The Cisco OpenStack Installer has been tested and is only supported on Ubuntu Precise (12.04 LTS). It is recommended to configure RAID for all nodes other than Swift storage nodes. Details for configuring RAID on UCS B-Series and C-Series servers can be found here. Puppet relies on SSL certificates to maintain its list of nodes. This means that it is imperative that two things are set up on all nodes before you run puppet for the first time:

- The node's domain and hostname must both be what you expect. Use *hostname -d* and *hostname -f* to check this
- The nodes must all be time synced. Use *ntpdate* to check this.

If you make a mistake, you will need to clean out the puppet certificates. If you are using cobbler to deploy your nodes, these two steps will be included for you, but you must still make sure that the domain and time are correctly set on your build node before running the install script.

Reference Configuration for Testing

For the purpose of having a common configuration with which to test the installer, the following is a suggested configuration for installing and running the all-in-one scenario using a virtual machine:

- Create two host-only networks with DHCP enabled (we will attach these in a later step)
- Create a VM with at least 2048 MB of RAM, 16 GB disk and a NAT network adapter
- Boot the VM and run the Ubuntu 12.04.3 64-bit server installation ISO
    - Select defaults during the installation, except for the following:
        - If you have more than one network adapter / NIC attached already, you may be prompted to select one. Choose the interface that can connect to the outside world (often eth0)
        - Hostname: a FQDN e.g., aio-server.example.com
        - Add the OpenSSH Server package when prompted
- Reboot
- Verify networking and shut down the VM
- In addition to the existing NAT network adapter, attach the host-only adapters you created above (with promiscuous mode = "allow all")
- Boot the VM
- Update the network settings:
    - *sudo vi /etc/network/interfaces*
        - Create eth1 and eth2 entries using eth0 as a template
    - *sudo reboot*
    - Verify the networking settings were applied and active
        - *ifconfig*
        - ssh to the default_interface (typically the 192.168.*.* one)
- Now would be a good time to take a snapshot of your VM!
- Install the all-in-one scenario as detailed below, with the following additional exports:
    - `export default_interface=eth1 # This is the interface you logged into via ssh`
    - `export external_interface=eth2`

Build Server

Install Ubuntu 12.04 (AMD 64-bit) from CD/ISO or automated install (i.e. kickstart). You can find instructions for installing Ubuntu 12.04 LTS here. For All-in-One deployments, create an LVM Volume Group named cinder-volumes during the Ubuntu installation process. The volume is used by Cinder for

providing persistent storage to instances. The size of the volume will depend on your deployment needs. Keep in mind, the Volume Group is not necessary for all other deployment models. Lastly, select ssh-server as the only additional package during the Ubuntu Precise installation.

We recommend that your build node be named *build-server*. If you use another name, be sure to read on below to learn about a few lines of YAML you'll need to add to a host override file.

First, become root, then clone the Cisco OpenStack installer repository into /root:

```
sudo su -
```

Install git, which is required to pull the puppet_openstack_builder Puppet module.

```
apt-get install -y git

cd /root && git clone -b havana https://github.com/CiscoSystems/puppet_openstack_builder && cd pup
```

And select cisco as the vendor:

```
export vendor=cisco
```

A 'scenario' is a collection of roles that perform different tasks within an OpenStack cluster. For example, if you wanted an environment where some nodes run all services, the scenario 'all_in_one' would be suitable, or if you wanted to separate control and compute nodes, the '2_role' scenario can do that. Here are the currently available scenarios and their roles:

```
- Scenario: all_in_one
- Roles: all_in_one, compute
- This scenario has a node with puppet services + control services + compute services,
  and optionally additional nodes with just compute services

- Scenario: 2_role
- Roles: build, control, compute, swift_proxy, swift_storage
- This scenario separates the puppet, control, compute, and swift services into separate nodes

- Scenario: full_ha
- Roles: build, control, compute, swift_proxy, swift_storage, load_balancer
- This scenario is similar to the 2_role one, but includes a load balancer role and is used for HA

- Scenario: compressed_ha
- Roles: build, compressed_ha, compressed_ha_cephall, compressed_ha_cephosd, compressed_ha_cephmon
- This scenario is creates a full HA deployment on 3 nodes with each node serving all functions.
```

To select a scenario, export the appropriate environment variable like this:

```
export scenario=2_role
```

Now run a script that will prepare puppet, modules and repositories on the build node:

```
cd ~/puppet_openstack_builder/install-scripts
./install.sh 2>&1 | tee install.log
```

Note that *install.sh* only needs to be run once. You'll see a lot of messages scroll by as the installation takes place--a copy of them can be found in install.log when you're finished (which is handy for troubleshooting). If you make changes to files after that point, you can make them take effect by running puppet directly (part of *install.sh*'s duties are to set up puppet which is no longer necessary after the first run):

```
puppet apply -v /etc/puppet/manifests/site.pp
```

Configuration and Customization

At this point, the data model has been installed and any required customizations should be made to the data model. If you selected the all_in_one scenario, some sample data has already been placed in

```
/etc/puppet/data/hiera_data/user.yaml
```

that you should review for correctness. If the file is not there, go ahead and create it, and put in any config options that are required. Here are some of the common ones:

```
vi /etc/puppet/data/hiera_data/user.yaml
```

Required Config

```
# Set the hostname of the build node
build_node_name: build-server

# Set the hostname of the control node
coe::base::controller_hostname: control-server

# Set the IP address of the control node
coe::base::controller_node_internal: 192.168.100.20
```

Software Repository Config

```
# Configure which repository to get packages from. Can be 'cisco_repo' or 'cloud_archive'
# The former is an apt repo maintained by Cisco and the latter is the Ubuntu Cloud Archive
coe::base::package_repo: 'cisco_repo'

# Which Openstack release to install - can be 'grizzly' or 'havana'. Other versions currently unte
coe::base::openstack_release: 'havana'

# If cisco_repo is used, the mirror location:
coe::base::openstack_repo_location: 'http://openstack-repo.cisco.com/openstack/cisco'

# Cisco maintains a supplemental repo with packages that aren't core to Openstack, but
# are frequently used, such as ceph and mysql-galera.
coe::base::supplemental_repo: 'http://openstack-repo.cisco.com/openstack/cisco_supplemental'

# If you are using the ubuntu repo, you can change from 'main' (default) to 'updates'
coe::base::ubuntu_repo: 'updates'

# Set a proxy server
coe::base::proxy: '192.168.100.100'

# Set a gateway server
coe::base::node_gateway: '192.168.100.101'
```

Connectivity Config

```
# The DNS Domain
domain: domain.name

# A list of NTP servers
ntp_servers:
  - time-server.domain.name

# Used to tell Neutron agents which IP to bind to.
```

```
# We can get the ip address of a particular interface
# using a fact.
internal_ip: "%{ipaddress_eth1}"

# Similarly, VNC needs to be told the IP to bind to
nova::compute::vncserver_proxyclient_address: "%{ipaddress_eth1}"

# This interface will be used to NAT to the outside world. It
# only needs to be on the control node(s)
external_interface: eth2

# This interface is used for communication between openstack
# components, such as the database and message queue
public_interface: eth1

# This interface is used for VM network traffic
private_interface: eth1

# This will be used to tell openstack services where the DB and
# other internally consumed services are
controller_internal_address: 192.168.100.10

# This is used in the HA scenario to set the public keystone
# endpoint
controller_public_address: 192.168.100.10
```

Passwords

```
# Users can set either a single password for all services, plus
# the secret token for keystone
secret_key: secret
password: password123

# Or passwords can be specified for each service
cinder_db_password: cinder_pass
glance_db_password: glance_pass
keystone_db_password: key_pass
nova_db_password: nova_pass
network_db_password:   quantum_pass
database_root_password: mysql_pass
cinder_service_password: cinder_pass
glance_service_password: glance_pass
nova_service_password: nova_pass
ceilometer_service_password: ceilometer_pass
admin_password: Cisco123
admin_token: keystone_admin_token
network_service_password: quantum_pass
rpc_password: openstack_rabbit_password
metadata_shared_secret: metadata_shared_secret
horizon_secret_key: horizon_secret_key
ceilometer_metering_secret: ceilometer_metering_secret
ceilometer_db_password: ceilometer
heat_db_password: heat
heat_service_password: heat_pass
```

Disk Partitioning

When deploying nodes with Cobbler, you can control some aspects of disk partitioning by placing the following directives in /etc/puppet/data/hiera_data/user.common.yaml. If you do not want a separate /var from /, set *enable_var* to false. If you do not want extra disk space set aside in an LVM volume to use for Cinder volumes via iSCSI, set enable_vol_space to false (you likely want this true if you want to use iscsi

volumes on compute nodes). You can specify the minimum sizing of /var and / partitions using the var_part_size and root_part_size directives, respectively. The values should be specified in units of megabytes.

```
root_part_size: 65536
var_part_size: 432000
enable_var: true
enable_vol_space: true
```

Advanced Options

The following options can be provided to your build node to enable special features during baremetal provisioning. They can be placed in a host override file (/etc/puppet/data/hiera_data/hostname/INSERT_YOUR_HOSTNAME_HERE.yaml) or in /etc/puppet/data/hiera_data/user.common.yaml.

To install a specific kernel package and set it to be the kernel booted by default, set this directive:

```
load_kernel_pkg: 'linux-image-3.2.0-51-generic'
```

To specify command-line options that should be passed to the kernel at boot time, set this directive:

```
kernel_boot_params: 'quiet splash elevator=deadline'
```

Note: using "elevator=deadline" is currently recommended for those using iscsi volumes in Cinder as some I/O issues have been reported using the default elevator.

To set the timezone on the clock for nodes booted via Cobbler, set this directive:

```
time_zone: US/Eastern
```

Nodes

You will also need to map the roles in your selected scenario to hostnames. This is done in /etc/puppet/data/role_mappings.yaml:

```
vi /etc/puppet/data/role_mappings.yaml

control-server: controller
control-server01: controller
control-server02: controller
control-server03: controller

compute-server: compute
compute-server01: compute
compute-server02: compute
compute-server03: compute

all-in-one: all_in_one

build-server: build

load-balancer01: load_balancer
load-balancer02: load_balancer

swift-proxy01: swift_proxy
swift-proxy02: swift_proxy
```

```
swift-storage01: swift_storage
swift-storage02: swift_storage
swift-storage03: swift_storage
```

In most scenarios, if you ran *install.sh* on your build node with the scenario environment variable set you should have a puppet master already set up. If you've created a scenario in which this is not the case, you can now run the master.sh script to turn the build node into a puppet master:

```
bash master.sh
```

If you chose to set the hostname of your build node to something other than "build-server", you may also need to set up a few Cobbler-related directives in a host override file. In /etc/puppet/data/hiera_data/hostname/build-server.yaml you'll find several directives used by cobbler:

```
# set my puppet_master_address to be fqdn
puppet_master_address: "%{fqdn}"
cobbler_node_ip: '192.168.242.100'
node_subnet: '192.168.242.0'
node_netmask: '255.255.255.0'
node_gateway: '192.168.242.1'
admin_user: localadmin
password_crypted: $6$UfgWxrIv$k4KfzAEMqMg.fppmSOTd0usI4j6gfjs0962.JXsoJRWa5wMz8yQk4SfInn4.WZ3L/MCt
autostart_puppet: true
ucsm_port: 443
install_drive: /dev/sda
#ipv6_ra: 1
#interface_bonding = 'true'
```

Copy these into your /etc/puppet/data/hiera_data/hostname/[insert your build node hostname here].yaml file and set them to appropriate values for your build node.

Other Nodes

After setting up the build node, all other nodes can be deployed via one of two methods.

First: if you intend to use Cobbler to perform baremetal provisioning of the node, you can use the *clean_node.sh* script located in scripts/clean_node.sh:

```
cd /root/puppet_openstack_builder/scripts
bash clean_node.sh node01
```

Doing so will clean out any prior puppet certificates, enable netboot for the node in question, and power cycle the node. When the machine reboots, it will start a PXE install of the baremetal operating system. Once the operating system is installed, the machine will reboot again and a puppet agent will be started. The agent will immediately commence installing OpenStack.

If you have pre-provisioned your server and don't wish to use Cobbler for baremetal provisioning, you can run the following commands on the node instead:

```
apt-get install -y git
cd /root
git clone https://github.com/CiscoSystems/puppet_openstack_builder
cd puppet_openstack_builder
git checkout h.2
cd install-scripts
export build_server_ip=<YOUR_BUILD_NODE_IP>
bash setup.sh
```

```
puppet agent -td --server=<YOUR_BUILD_NODE_FQDN> --pluginsync
```

This will setup Puppet and then perform a catalog run. If you already have Puppet 3.2 installed or have a repository from which it can be installed already set on the node, you simply run the script directly from curl rather than cloning the repository first:

```
apt-get install -y curl
export build_server_ip=<YOUR_BUILD_NODE_IP>
bash <(curl \-fsS https://raw.github.com/CiscoSystems/puppet_openstack_builder/h.2/install-scripts
puppet agent -td --server=<YOUR_BUILD_NODE_FQDN> --pluginsync
```

Note: The setup script should only be run once. If you want to force an update after the initial run, simply restart the Puppet agent (using **service puppet restart** if Puppet is set up to run automatically or **puppet agent -td --server=<YOUR_BUILD_NODE_FQDN> --pluginsync** if it isn't) to perform another catalog run.

Cobbler Configuration

The 2_role and full_ha scenarios configure a build server which provides puppetmaster and, optionally, cobbler bare metal deployment services for managing the remaining nodes in the OpenStack cluster. To use cobbler, you will need to set up role mappings in /etc/puppet/data/role_mappings.yaml as previously described. You will also need to provide basic hardware details about each of the hardware nodes being managed by this instance of cobbler, including the MAC addresses of the network boot interfaces, machine host names, machine IP addresses, and management account information for the UCSM or CIMC management of the nodes. Define these parameters by editing the /etc/puppet/data/cobbler/cobbler.yaml file:

```
vi /etc/puppet/data/cobbler/cobbler.yaml
```

When editing the file, keep in mind that it has 4 major sections: preseed, profile, node-global, and 1 or more individual node definitions.

Cobbler preseed

Cobbler uses IPMI for power management of UCS C-Series servers. Install the ipmi package if using C-Series servers:

```
apt-get install -y ipmitool
```

The Cobbler preseed stanza in cobbler.yaml defines parameters that customize the preseed file that is used to install and configure Ubuntu on the servers. Parameters you might need to adjust in here include things like default repos to include in hosts, and where these repos are located.

```
preseed:
  repo: "havana"
  repo: "http://openstack-repo.cisco.com/openstack/cisco havana main"
```

Cobbler profile

The Cobbler profile stanza in cobbler.yaml defines options that specify the cobbler profile parameters to apply to the servers. This section typically will not require customization.

```
profile:
  name: "precise"
  arch: "x86_64"
  kopts: "log_port=514 \
```

```
priority=critical \
local=en_US \
log_host=192.168.242.100 \
netcfg/choose_interface=auto"
```

Cobbler node-global

The Cobbler node-global stanza in cobbler.yaml specifies various configuration parameters which are common across all servers in the cluster, such as gateway addresses, netmasks, and DNS servers. Power parameters which are standardized are also included in this stanza. You will likely need to change several parameters in this section, including power-type, power-user, power-pass, get_nameservers, get_ipaddress, get_gateway, no_default_route, partman-auto, and others.

```
node-global:
  profile: "precise-x86_64"
  netboot-enabled: "1"
  power-type: "ipmitool"
  power-user: "admin"
  power-pass: "password"
  kickstart: "/etc/cobbler/preseed/cisco-preseed"
  kopts: "netcfg/get_nameservers=2.4.1.254 \
netcfg/confirm_static=true \
netcfg/get_ipaddress={$eth0_ip-address} \
netcfg/get_gateway=192.168.242.100 \
netcfg/disable_autoconfig=true \
netcfg/dhcp_options=\"Configure network manually\" \
netcfg/no_default_route=true \
partman-auto/disk=/dev/sda \
netcfg/get_netmask=255.255.255.0 \
netcfg/dhcp_failed=true"
```

For security the default deployment assumes no default route for individual OpenStack nodes, with the build server used as a jump box to access them (and any VM access done through provider networks extended by bridging to the VMs on the OpenStack nodes). If you need a default route on the OpenStack infrastructure nodes, be sure to delete the line above which says not to add a default route:

```
netcfg/no_default_route=true \
```

Cobbler node definitions

Each individual node being managed by Cobbler is listed as a separate node definition. The node definition for each host defines, at a minimum, the hostname and interface configuration information for each server, as well as any other parameters which aren't defined in node-global. Create one stanza here for each node, using a format like this example.

```
build-server:
  hostname: "build-server.cisco.com"
  power-address: "192.168.2.101"
  interfaces:
    eth0:
      mac-address: "a1:bb:cc:dd:ee:ff"
      dns-name: "build-server.cisco.com"
      ip-address: "192.168.242.100"
      static: "0"
```

Global Parameters

A couple of parameters are defined in a "global" parameter hierarchy separate from the user.$scenario.yaml files which contain most user settings. These global parameters take precedence over any user parameters, and are intended to set global "defaults" for topological choices like:

- what storage backend do I use for Cinder?
- what storage backend do I use for Glance?
- what tunneling technology do I use for network segregation?

These settings are found in configuration files under data/global_hiera_params

Selecting a Cinder backend

In COI deployments Cinder can be backed by either Ceph block storage, or iSCSI block storage. Note that Ceph is generally the preferred choice in HA installations.

To choose Ceph, edit /etc/puppet/data/global_hiera_params/common.yaml and change the cinder_backend setting to rbd (and also see further instructions below about configuring Ceph):

```
cinder_backend: rbd
```

To choose iSCSI, edit /etc/puppet/data/global_hiera_params/common.yaml and change the cinder_backend setting to iscsi:

```
cinder_backend: iscsi
```

Selecting a Glance backend

In COI deployments Glance can be backed by local file storage, or by Swift object storage, or by Ceph object storage. Either Swift or Ceph object storage are appropriate choices for HA installations.

To choose Swift, edit /etc/puppet/data/global_hiera_params/common.yaml and change the glance_backend setting to swift:

```
glance_backend: swift
```

To choose Ceph, edit /etc/puppet/data/global_hiera_params/common.yaml and change the glance_backend setting to rbd (and also see further instructions below about configuring Ceph):

```
glance_backend: rbd
```

To choose local file storage, edit /etc/puppet/data/global_hiera_params/common.yaml and change the glance_backend setting to file:

```
glance_backend: swift
```

Advanced network topologies

In COI HA deployments, global settings for network topologies are defined in scenario-specific files. data/global_hiera_params/scenario/compressed_ha.yaml sets defaults for compressed_ha deployments while data/global_hiera_params/scenario/full_ha.yaml sets defaults for full_ha deployments.

The default values set in these files work for the defined "reference" topologies targeted by COI. These files contain several settings that may need changing in more complex network configurations, however. Examples include choice of segregation technology, or type of network plugin used.

Adding Ceph to Your OpenStack Nodes

Adding Ceph services to a node is trivial. The core ceph configuration data is in /etc/puppet/data/hiera_data/user.common.yaml Here you can configure the particulars of your cluster. Most items can stay as their defaults. You will likely need to modify all the networking options.

Once you've modified this file, you will need to create a hostname override file for your target server. This is stored in /etc/puppet/hiera_data/hostname/, with ceph01.yaml as an example. Here, you specify the disks to use as OSDs on the target node. You don't need to add any data here if you are adding a monitor to the target server.

There are three Ceph classgroups: ceph_mon, ceph_osd, and ceph_all. ceph_all aggregates ceph_mon and ceph_osd. You can add ceph services to a particular role by adding these to the target role configuration file. Eg if you want all compute nodes to offer OSD, you can add the line ceph_osd to the compute.yaml file in /etc/puppet/data/classgroups. To add services only to specific servers that will be a subset of a large scenario, you need to clone the existing scenario to a new name and add the ceph services to the new scenario. Then configure your node(s) in role_mappings.yaml to use the new scenario.

Once this is complete, the next puppet run on the target servers will bring your cluster up and online.

- There is a caveat to this. There can be only one initial ceph monitor specified in user.common.yaml. This ensures that the node running the primary mon is considered primary and comes up first. Additional mons and osds are then added during their respective hosts' puppet run.*

To configure Cinder and Glance to use Ceph for storage, you will also need to configure the following. This can be done independently of the cluster deployment process:

In /etc/puppet/data/global_hiera_params/common.yaml: (or if you are using compressed HA: /etc/puppet/data/global_hiera_params/scenario/compressed_ha.yaml

```
cinder_backend: rbd
glance_backend: rbd
```

In /etc/puppet/data/hiera_data/cinder_backend/rbd.yaml:

```
cinder::volume::rbd::rbd_pool: 'volumes'
cinder::volume::rbd::glance_api_version: '2'
cinder::volume::rbd::rbd_user: 'admin'
# keep this the same as your ceph_monitor_fsid
cinder::volume::rbd::rbd_secret_uuid: 'e80afa94-a64c-486c-9e34-d55e85f26406'
```

In /etc/puppet/data/hiera_data/glance_backend/rbd.yaml:

```
glance::backend::rbd::rbd_store_user: 'admin'
glance::backend::rbd::rbd_store_ceph_conf: '/etc/ceph/ceph.conf'
glance::backend::rbd::rbd_store_pool: 'images'
glance::backend::rbd::rbd_store_chunk_size: '8'
```

You can add disks on OSD hosts by creating host override files in /etc/puppet/data/hiera_data/hostname/${short_hostname_of_your_data_node}.yaml like this:

```
# has_compute must be set for any server running nova compute
# nova uses the secret from virsh
cephdeploy::has_compute: true

# These are the disks for this particular host that you wish to use as OSDs.
# Specify disks here will DESTROY any data on this disk during the first puppet run.
cephdeploy::osdwrapper::disks:
  - sdb
  - sdc
```

Having trouble? You must follow these steps on a node that failed to setup OSDs properly

```
ceph-deploy uninstall $HOST
ceph-deploy purge $HOST
ceph-deploy purgedata $HOST
dd if=/dev/zero of=/dev/yourdisk count=200 bs=1M
(do the above to each OSD disk)
ceph-deploy disk zap host:disk
userdel -r cephdeploy
```

Once those steps are done, then run the puppet agent again.

Configuring Nova for Migrations

Migration enables an administrator to move a virtual machine instance from one compute host to another. This feature is useful when a compute host requires maintenance. Migration can also be useful to redistribute the load when many VM instances are running on a specific physical machine.

The migration types are:

Migration (or non-live migration). The instance is shut down (and the instance knows that it was rebooted) for a period of time to be moved to another hypervisor.

Live migration (or true live migration). Almost no instance downtime. Useful when the instances must be kept running during the migration.

The types of live migration are:

- Shared storage-based live migration. Both hypervisors have access to shared storage.*
- Block live migration. No shared storage is required.*

Volume-backed live migration. When instances are backed by volumes rather than ephemeral disk, no shared storage is required, and migration is supported (currently only in libvirt-based hypervisors).*

Currently, NFS migration and true migration are available with a few caveats.

Ceph volume-backed true live migration is still a WIP.

Configuring Keystone with SSL

To enable SSL in keystone,

- Make sure you have the ssl certificates generated either self signed or signed with some authority CA.

If you want to generate your own certificates you can follow the instructions here.

- To begin with, do your deployment without SSL
- Copy your certificates on to control nodes under /etc/keystone/ssl. Make sure the files are owned by keystone user and group.
- Then on build node, update the user.common.yaml with following settings
- set the protocol to https

```
controller_public_protocol: 'https'
```

- Enable ssl and update your certificate paths

```
enable_ssl: true

ssl_certfile: '/etc/keystone/ssl/certs/server.pem'

ssl_keyfile:  '/etc/keystone/ssl/private/serverkey.pem'

ssl_ca_certs: '/etc/keystone/ssl/certs/cacert.pem'

ssl_ca_key: '/etc/keystone/ssl/private/ca.key'

ssl_cert_subject: '/C=US/ST=Unset/L=Unset/O=Unset/CN=192.168.255.208'
```

- On your control nodes rerun puppet agent. You should now have SSL enabled in keystone.
- Note that this is still work in progress, So this might not fully function in HA scenarios.

Troubleshooting, Known Issues, and Common Questions

For Troubleshooting information, including answers to common questions and installation problems, visit the OpenStack Troubleshooting Page.