UMBC Cyber Dawgs - Advanced VMs

By: Zack Orndorff and Christian Beam

Outline

You're going to configure a VM to serve DNS, SSH, MySQL, and Git all in one machine. It's an awful idea for production, but it's great for learning!

Protip: Don't copy-paste from this document. Google Drive made the quotes curly quotes, and curly quotes and UNIX don't mix.

We recommend you have another Linux desktop VM to connect to this VM from, but it's possible to do from your host as well.

Install CentOS 7

1. You can download the ISO from here: https://www.centos.org/download/
    a. I recommend the Minimal ISO
    b. Fast mirror is: http://mirror.umd.edu/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-1708.iso
        i. We have a fast connection to most other universities, and College Park is especially fast
    c. SHA256 sum should be: `bba314624956961a2ea31dd460cd860a77911c1e0a56e4820a12b9c5dad363f5`
    d. Yes, verify it, it doesn't take that long.
2. Create a new VM with these settings:
    a. Red Hat (64-bit)
        i. CentOS is based on Red Hat
    b. 1024 MB RAM (you can probably get by with 512 though)
    c. 10 GB Disk
3. Configure the network in the VM (do it now, it'll save you frustration)
    a. Go to VM settings
    b. Adapter 1 -> NAT
    c. Adapter 2 -> Host-only
        i. If you are on a Linux host (If you are not sure, you aren't), you will need to configure a host-only network. Go to File > Preferences, then go to the Network tab, then the Host-only tab of that and create one.
4. Start up the installer
5. Configure "Installation Destination"
    a. Just click Done, the defaults are fine for what we're doing

6. Configure “Network and Host Name
    a. Click each network adapter and turn on the switch
    b. Click done
    c. This saves you having to manually enable the adapters later. It’s easier, I promise.
    d. Change the hostname to “advlab.prac.umbccd.net”
        i. You can pick your own domain, but it is best practice to use one you actually own.
        ii. We own that one :)
        iii. Just pick one if you want, this isn’t production
    e. Click Done.
7. Click “Begin Installation”
8. Set a root password, and create a user for yourself.
    a. Make yourself an administrator, that way sudo works. Otherwise you’ll have to manually add yourself to the group.
9. Wait for it to finish
10. When it finishes, click the button to restart the VM into your new install.
11. When the machine boots, log in as your user.

Set a static IP address

As a general rule, you want your DNS servers to have a static IP address, otherwise you don’t have a server to contact in order to find out what their address is.
You also want servers in general to have a static IP...

1. `sudo vi /etc/sysconfig/network-scripts/ifcfg-enp0s8`
2. Set `ONBOOT=yes` if not already set
    a. (otherwise you’d need to start it manually on every reboot)
3. Change `BOOTPROTO=dhcp` to `BOOTPROTO=static`
4. Add the following:
    a. `IPADDR=192.168.56.25`
    b. `NETMASK=255.255.255.0`
    c. `GATEWAY=192.168.56.1`
    d. `DEFROUTE=no`
        i. This keeps it from trying to route your internet traffic over the host-only adapter.
5. `sudo systemctl restart network`
6. Run `ip addr` or `ip a` to check changes
    a. These are the newer version of `ifconfig`, if you’ve heard of it

Disable the firewall

1. **Yes, this is a bad idea. In real work, you should have a firewall.**

2. But this isn't the firewalls lab… and we have **time constraints**. So:
3. `sudo systemctl disable firewalld`
4. `sudo systemctl stop firewalld`

You can think of it as a sort of "exercise for the reader" to set up when you're done :)

Configure SSH

1. Source: https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2
2. We are going to configure ssh to use keys, and disable password authentication
3. SSH keys are superior to passwords because they are not nearly as easy to bruteforce
4. Generating a key pair will give use two keys: a public key, and a private key. The public key will go on our ssh server (our centos vm), and the private key will be used with the client that is trying to ssh into the server..
5. This won't be that long or hard, let's get started
6. First, on your other VM (or host), generate the RSA key pair
    a. Linux:
        i. Run: `ssh-keygen -b 4096`
        ii. It will ask where to save it, default is within the .ssh directory of our home directory. The default should be just fine, but if you like being different then go right ahead! Just don't ask us to find it when you lose them :P (Seriously, just leave them there, it's a good idea.)
        iii. Next it will ask for a passphrase for the file. For this lab we can leave it with no passphrase (unless you seriously plan to use this VM as a ssh server, which we would advise against). In normal circumstances, you would set a password.
        iv. You should now see a `id_rsa` and `id_rsa.pub` file within the `~/.ssh` directory
    b. Windows
        i. PuTTY:
            1. Open PuTTYgen and generate a new key. Save a copy of where it says there's a key for OpenSSH. Save the private key somewhere safe.
            2. Add the public key for OpenSSH to the `authorized_keys` file as discussed below
7. Copy the public key over to the server in the `~/.ssh/authorized_keys` file. You could do this in a lot of ways, but an easy way is with the ssh-copy-id command.
    a. `ssh-copy-id` <username>`@`<ssh_server_IP>
    b. Or you can just '`cat ~/.ssh/id_rsa.pub`' and place the contents of that in the `authorized_keys` file
        i. If you have to create either the `.ssh` directory or the `authorized_keys` file, you'll need to set the following permissions, SSH is pretty picky about them:

1. `mkdir .ssh`
2. `chmod 700 .ssh`
3. `touch .ssh/authorized_keys`
4. `chmod 600 .ssh/authorized_keys`

8. Next, we will disable password authentication.
    a. `sudo vi /etc/ssh/sshd_config`
    b. Find the line that says "`PasswordAuthentication`" (Pro-vi-tip: when in vi, type '/' followed by what you want to look for and hit enter)
    c. Uncomment the line and change it to "`PasswordAuthentication no`"
    d. Save the file and exit
9. Restart the ssh service
    a. `sudo systemctl restart sshd`

Install a DNS Server (BIND)

1. We'll be following this tutorial here: https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-centos-7
    a. I've shamelessly stolen most of their material, so it's easier for you to follow. The above link serves as credit to prove I'm not an awful person
2. `sudo yum install bind bind-utils`
3. `sudo vi /etc/named.conf`
    a. Add a section above the options section:

```
acl "trusted" {
     192.168.56.0/24;
     127.0.0.0/8;
};
```

    b. Add `192.168.56.25` to the listen-on port 53 directive.
    c. Change allow-query to `allow-query { trusted; }`
        i. This is because improperly configured can be used in DoS attacks, so I'm mentioning it here.
        ii. In the real world, you'd be running an authoritative DNS server (IE listed in public DNS), so you'd allow queries from everywhere, but disable recursion.
    d. At the bottom of the file, add `include "/etc/named/named.conf.local"`
4. `sudo vi /etc/named/named.conf.local`
    a. Add the following:

```
zone "prac.umbccd.net" {
     type master;
     file "/etc/named/zones/db.prac.umbccd.net";
};
zone "56.168.192.in-addr.arpa" {
```

```
        type master;
        file "/etc/named/zones/db.56.168.192";
};
```

i. This tells BIND where to look for the **zone files**, where the data is stored.

b. Note the semicolon after the braces

5. `sudo mkdir /etc/named/zones`
    a. Tab complete won't work, b/c /etc/named isn't world readable for some reason
6. `sudo vi /etc/named/zones/db.prac.umbccd.net`
    a. Add the following:

```
$TTL    900 ; Default TTL (seconds) [15 minutes]
@       IN      SOA     advlab.prac.umbccd.net. root.prac.umbccd.net. (
                        1               ; Serial (change every time you modify this file)
                               86400          ; Refresh (master <- slave)
                               43200          ; Retry   (master <- slave retry)
                        604800          ; Expire  (master <- slave cache expiration)
                        900 )           ; Negative Cache TTL (TTL for NXDOMAIN)

; name servers
advlab          IN      A       192.168.56.25
@               IN      NS      advlab

git             IN      CNAME   advlab
info            IN      CNAME   advlab

; root of domain (it shouldn't be a CNAME because (valid) reasons)
@               IN      A       192.168.56.25
```

7. `sudo cp /etc/named/zones/db.prac.umbccd.net /etc/named/zones/db.56.168.192`
8. `sudo vi /etc/named/zones/db.56.168.192`
    a. Make it look like this (you really didn't want to retype it lol, that's why we used cp)

```
$TTL    900 ; Default TTL (seconds) [15 minutes]
@       IN      SOA     advlab.prac.umbccd.net. root.prac.umbccd.net. (
                        1               ; Serial (change every time you modify this file)
                               86400          ; Refresh (master <- slave)
                               43200          ; Retry   (master <- slave retry)
                        604800          ; Expire  (master <- slave cache expiration)
                        900 )           ; Negative Cache TTL (TTL for NXDOMAIN)

@       IN      NS      advlab.prac.umbccd.net.

; PTR records
25      IN      PTR     advlab.prac.umbccd.net.
```

9. `sudo named-checkconf`
    a. (make sure it doesn't throw any errors)
10. `sudo named-checkzone prac.umbccd.net /etc/named/zones/db.prac.umbccd.net`
11. `sudo named-checkzone 56.168.192.in-addr.arpa /etc/named/zones/db.56.168.192`
12. `sudo systemctl start named`
13. `sudo systemctl enable named`
14. `dig @127.0.0.1 advlab.prac.umbccd.net`
    a. If you did everything correctly, this should work :D

Now, we'll make our server use itself as DNS

1. `sudo vi /etc/sysconfig/network-scripts/ifcfg-enp0s8`
    a. Add `DNS1="192.168.56.25"`
2. `sudo vi /etc/sysconfig/network-scripts/ifcfg-enp0s3`
    a. Add `PEERDNS="no"`
3. `sudo systemctl restart network`
4. `dig git.prac.umbccd.net`
    a. Should return `192.168.56.25` if you did it right :D

Now go into your other VM's network settings and set this one as DNS. Or you can do it on your host, but just don't forget to change it back.

Install MySQL

1. Source: https://www.digitalocean.com/community/tutorials/how-to-install-mariadb-on-centos-7
2. Use Yum to install MariaDB
    a. `sudo yum install mariadb-server`
3. Start the server
    a. `sudo systemctl start mariadb`
4. Check its status
    a. `sudo systemctl status mariadb`
    b. It should say active (running), let us know if it says something else
5. Enable MariaDB to start on boot
    a. `sudo systemctl enable mariadb`
    b. You should see something like "Created symlink from …… to ……"
6. Secure the server
    a. We really should make you do this manually, but for time's sake we will allow you to use this security script provided by MariaDB (also we think it's best practice)
        i. `sudo mysql_secure_installation`
        ii. Hit enter when it asks for a root password (you haven't set it yet)
        iii. Hit y and provide a root password for mysql
        iv. Proceed to hit y for the rest of the options provided
7. Test the server
    a. `mysqladmin -u root -p version`
    b. This should output something like "mysqladmin Ver 9.0 Distrib 5.5.56-MariaDB, for Linux on x86_64..." and more

Install Gogs

Gogs is a Git server. It's like GitHub, only free, and you can host it yourself. Its competitors include Bitbucket and GitLab. But Gogs is easier to install, so we're doing that.

1. `yum install git`
    a. Install the Git version control software
2. `sudo useradd git`
    a. Not a special name, but it's used by convention.
3. `sudo -iu git`
    a. Equivalent to `sudo su - git`
4. `curl -LO https://cdn.gogs.io/0.11.29/linux_amd64.tar.gz`
5. `tar xf linux_amd64.tar.gz`
    a. Older versions of tar would have required tar xzf, with the z standing for Gzip.
    b. Any recent version of GNU tar should work with just tar xf
    c. https://xkcd.com/1168/
6. `cd gogs`
7. `mysql -u root -p`
    a. *type password*
    b. `source scripts/mysql.sql`
        i. creates a database for gogs, it's a simple script provided by them
    c. `CREATE USER 'gogs'@'localhost' IDENTIFIED BY 'gogsisawesome';`
        i. As a general rule, most database users are for applications, not for human users.
        ii. In production, you'd actually pick a decent password.
    d. `GRANT ALL PRIVILEGES ON gogs.* TO 'gogs'@'localhost';`
        i. You can give less permissions in theory, but it still has to be able to read everything...
    e. exit
8. `./gogs web`
    a. Run the application
9. Go to http://advlab.prac.umbccd.net:3000/ in a web browser (somewhere you set up your DNS)
10. Configure Gogs:
    a. to use the database and user (NOT root) you just created for MySQL
    b. Change the Application URL to be http://advlab.prac.umbccd.net:3000/
    c. Under "Server and Other Services Settings", enable offline mode
        i. You shouldn't need this... but it'll reduce problems if stuff breaks.
    d. Expand the menu to create an admin account and create one for yourself.
11. Play around
12. Now go Ctrl+c the Gogs program

13. Log out of the git user (type exit)
14. Let's configure systemd to start Gogs automatically on boot
15. `sudo cp /home/gogs/gogs/scripts/systemd/gogs.service /etc/systemd/system`
    a. gogs.service is a **systemd unit file**, equivalent to an old-style init script
16. `sudo vi /etc/systemd/system/gogs.service`
    a. Change the After= line with all the databases to be just After=mariadb.service
    b. Don't change the other After= lines
17. `sudo systemctl start gogs.service`
18. `sudo systemctl enable gogs.service`
19. Now go try to log into Gogs again in your web browser and see if it still works.

Install Nginx

We're setting Nginx up as a reverse proxy for Gogs. This means that Nginx will be directly interacting with users, and it'll pass traffic to Gogs as necessary. This means you can use multiple server blocks/virtual hosts and share port 80 across different domain names.

1. `sudo yum install epel-release`
    a. CentOS only includes certain packages in the default repos, carrying certain support guarantees.
    b. Other packages are in EPEL, IIRC maintained by Fedora
    c. Still pretty good support
2. `sudo yum install nginx`
    a. You'll have to okay EPEL's signing key since this will be your first package coming from there
3. `sudo systemctl start nginx`
4. `sudo systemctl enable nginx`
5. Now go to http://advlab.prac.umbccd.net/ in a web browser and see if it works.
    a. Should be the nginx default page
6. `sudo vi /etc/nginx/nginx.conf`
7. Add a new server block below the existing one as follows:

```
server {
      listen 80;
      server_name git.prac.umbccd.net;

      location / {
            proxy_pass http://localhost:3000/;
      }
}
```

What's a server block you ask? Each server corresponds to a "virtual host" in Apache terms.

8. `sudo setsebool -P httpd_can_network_connect true`
    a. Tell SELinux that reverse proxying is okay
    b. The -P is for permanent, IE for after a reboot

9. `sudo systemctl restart nginx`
10. Try going to http://git.prac.umbccd.net/
    a. It should work now
11. `sudo -iu git`
12. `vi gogs/custom/conf/app.ini`
    a. Change `ROOT_URL` to http://git.prac.umbccd.net/
    b. Add a line below that `SSH_DOMAIN = git.prac.umbccd.net`
    c. Tells Gogs that we just changed its domain name
13. Exit from the git user
14. `sudo systemctl restart gogs`
15. Test again
16. At this point, if we were doing firewalls, we would block off all access to port 3000 from the outside.
17. Notice that going to http://advlab.prac.umbccd.net still works, and is still the default page. This is the beauty of virtual hosting.

Test your configuration

1. Create a Gogs repo
2. Clone it to your VM
3. Make some changes
4. Push them back. Check them out in the Web interface