Package management

Drawbacks of traditional installations

- By traditional installations, we refer to packages that are obtained in the form of `.tar.gz` files.
- This type of installation is done by either decompressing the file into a specific directory, or compiling the package from source using `./configure` and `make install`.
- It does not handle dependencies
- It may overwrite customized configuration files
- It's hard to uninstall, especially if it was compiled from source

That's why packages exist

- They preserve configuration files. If a new version of the file is to be installed, it is saved and `pkg.rpm.new`. For example, when a new version of `httpd` is installed, the `httpd.conf` is created as `httpd.conf.rpmnew`
- Dependencies are considered during installation
- Pre and post install scripts can be deployed with the application
- Multiple packages can be wrapped inside the same package for complex installations
- They can be easily uninstalled including or excluding their dependencies
- In the following slides we'll explore different ways of package management on different Linux distros.

Linux package types

- RPM: stands for Red Hat Package. It is used by Red Hat and RH based systems as well as SUSE. Ends in .rpm.
- Debian: used by Ubuntu and other debian systems. Ends with .deb.
- To install an RPM package, the rpm command is used.
- For installing a debian package, the dpkg command is used
- Both of those commands do not handle dependencies, cannot be used to search for packages, and do not offer metadata about the package.
- Because of this shortcoming, the Yellow dog Updater, Modifed (yum), the Advanced Package Tool (APT), and ZYpper were introduced to Red Hat Debian, and SUSE respectively.

Red Hat's rpm

- Used to install, validate, and search for packages (on the system).
- It contains a lot of useful options, among which:
  - -i : install
  - -U : upgrade
  - -e : erase
  - -q : query. It has to be combined with anther command line option to work. For example, `rpm -qa` displays all the packages installed on the system.
  - You can run `rpm -qa | grep package` to determine whether or not a package is installed
- rpm does not handle dependencies automatically. For example, if you tried to install vim (VI Improved) using rpm -i vi, you will find that you will have to download other dependencies.
- If the package does not have dependencies, it will be installed

Debian's dpkg

- It does the same role of rpm on Debian packages
- It contains some useful options, among which:
  - --install
  - --remove
  - -l to display installed packages
- The `dpkg -l | grep package` to determine whether or not a package is installed
- If the package does not have dependencies, it will be installed. For example, the htop

Linux package management systems

- They solve the shortcomings of using low level commands like `rpm` and `dpkg`.
- They can search, locate, and install the packages automatically.
- To do this, they must use a repository. A repository is like a container containing packages, with their respective metadata.
- Each vendor offers it's own repository (repo). Example: Red Hat offers the Red Hat Network (paid), Centos, and Oracle Linux have their own repos too.
- The packages are offered for download in HTTP or FTP
- In it's simplest form, you could install a package using yum install or apt-get install for Red Hat and Ubuntu respectively. This will handle searching, and downloading the package with all it's dependencies.

Ubuntu apt-get

- Before installing or updating any new package, it's a good practice to run apt-get update to update the data sources (repositories) to reflect the latest changes.
- It can be configured by editing the `/etc/apt/sources.list`. This file contains the source URLs for the packages.
- You needn't change this file unless you want to use your own APT repository.
- A typical `sources.list` file contains different types of packages: those fully supported by Ubuntu (main), unsupported open source packages (universe), and paid unsupported packages (multiverse). Within each of those, there is an *updates* or *bug-fixes* component.

Making a local apt-get repo

- Used if you are managing a large number of Ubuntu machines, and you need all of them share one local repo to save bandwidth
- Install `apt-mirror` using `apt-get install apt-mirror`
- The configuration file is stored in `/etc/apt/mirror.list`
- You can start the mirroring process by typing apt-mirror. The first run will take a lot of time because of the amount of data that will be downloaded. Subsequent runs will consume much less time and can be automated by using cron jobs.
- The packages themselves are downloaded to `/var/spool/apt-mirror`. So you should ensure you have enough free space in that directory (at least 50GB). This path can be changed in the configuration file.
- You can run `/var/spool/apt-mirror/var/clean.sh` to delete obsolete packages
- To make your repo usable, just make it web reachable via HTTP or FTP. For example, make a symbolic link from your web directory to the packages directory. Of course all clients have to change their `sources.list` files to point to the local repo.

Running apt-get unattended

- You can automate system upgrades by using cron jobs
- It's a good practice to run apt-get update before attempting to run the upgrade to ensure that you have the latest data.
- The upgrade command is either apt-get upgrade or `apt-get dist-upgrade`. The later may delete packages that it *regards* no more compatible with the upgraded system, so you should use it with care.
- For full automation, use the –yes command argument with apt-get upgrade. This will confirm any and all dialogs that the system may ask in the upgrade process. So again be careful when specifying it.
- Some updates require system reboot to be reflected. For example, kernel upgrades.

Downloading packages only

- As a precaution, you can opt to download the packages only, and install them later after reviewing them. This can be achieved by using the --download-only command line argument.
- In this case, packages are downloaded into `/var/cache/apt`. You can use apt-get install `/path/to/package` to install/update the specified package.
- As more and more files get downloaded to this directory, you can use `apt-get autoclean` to remove files that are no longer used.

Red Hat yum

- It is used to install new packages and updates to a Red Hat system the same way apt-get is used with Ubuntu and Debian.
- The configuration file is located in `/etc/yum.conf`. While the repositories themselves are located in `/etc/yum.conf.d/`. Theoretically, you can add the repository information to the same configuration file, but this is a bad practice.
- To make system-wide upgrade, you use `yum update` (in contrast to `apt-get update`, which only updates the package information cache). Yum upgrade also does the same task.
- `Yum install`, `yum update`, and `yum remove` are used to install, update, or uninstall packages respectively.
- Auto-confirmation can be triggered by `-y` to the `yum` command. Discretion is advised.
- It cannot match partial package names unless an asterisk is added before, after, or before and after the package name.

Making local yum repo

- Because a typical Red Hat installation DVD (not the live CD) will contain a lot of packages already, it can be used as a local repo for the system. Especially if Internet connection is not available (or firewalled).
- This can be done by specifying a `file:///` URL that points to the mounted filesystem of the DVD.
- If needed, this can be further extended to serve other systems on the same network from the same DVD in the interest of saving time. Just use a symbolic link in the web directory that points to the mounted DVD filesystem, and update the appropriate client repo configuration to point to it.

Making a local mirror of yum repos

- The same target of using apt-mirror on Ubuntu but slightly more complex.
- It is done using `reposync` command. It is part of the `yum-utils` package, so you have to install this first.
- The following steps are used to mirror a remote repository:
  - Create a directory with enough space to contain the downloaded packages
  - Issue `reposync -r reponame -p /path/to/directory`
  - Instruct the system that this directory is a repository (create the necessary files and metadata) by issuing `createrepo /path/to/directory`.
  - Make a symbolic link from your web directory to the packages directory to make it web reachable
  - Make appropriate changes to the clients' repo configuration files to point to the new mirrored repo.