VirtualBox Install

- VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product.
- VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts.
- VirtualBox supports a large number of guest operating systems including but not limited to:
  - Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7),
  - DOS/Windows 3.x,
  - Linux (2.4 and 2.6),
  - Solaris and OpenSolaris.
- Go to https://www.virtualbox.org/wiki/Downloads and get the appropriate installer.
- Install it using standard procedures for your operating system.
- For help check https://www.virtualbox.org/manual/ch02.html.

Vagrant Install

- Vagrant provides easy to configure, reproducible, and portable work environments.

- Go to http://downloads.vagrantup.com/ and get the appropriate installer or package for your platform.
- Install it using standard procedures for your operating system.

- The installer will automatically add `vagrant` to your system path so that it is available in terminals.
- If it is not found, please try logging out and logging back in to your system (this is particularly necessary sometimes for Windows).

- Documentation at `http://docs.vagrantup.com/v2/`

(Vagrant Uninstall)

- On **Windows**, uninstall using the **add/remove programs** section of the **control panel**.

- On **Mac OS X**, remove the `/Applications/Vagrant` **directory** and the `/usr/bin/vagrant` **file**.

- On Linux, remove the `/opt/vagrant` **directory** and the `/usr/bin/vagrant` **file**.

- Remove the `~/.vagrant.d` **directory** to delete the user data.

Setup

- Configure Vagrant using a `Vagrantfile`. The purpose of the `Vagrantfile` is twofold:
  - Mark the root directory of your project. A lot of the configuration of Vagrant is relative to this root directory.
  - Describe the kind of machine and resources you need to run your project, as well as what software to install and how you want to access it.
- Initialize a directory for usage with Vagrant: `vagrant init`
  - `$ mkdir vagrant_lab`
  - `$ cd vagrant_lab`
  - `$ vagrant init`
- This will place a `Vagrantfile` in your current directory. You can take a look at the `Vagrantfile` if you want, it is filled with comments and examples.

Boxes (I)

- Instead of building a **virtual machine from scratch**, Vagrant uses a **base image** to quickly clone a virtual machine.
- These base images are known as **boxes** in Vagrant.

- Specifying the box to use for your Vagrant environment is always the first step after creating a new `Vagrantfile`.

- Boxes are added to Vagrant with vagrant box add. This stores the box under a specific name so that multiple Vagrant environments can re-use it.

  - `$ vagrant box add precise64 http://files.vagrantup.com/precise64.box`

- This will download the box from an HTTP source and save it as `"precise64"` in a directory that Vagrant manages (away from your project). You can also add boxes from a local file path.

Boxes (II)

- Added boxes can be re-used by multiple projects.
- Each project uses a box as an initial image to clone from, and never modifies the actual base image.
- This means that if you have two projects both using the precise64 box we just added, adding files in one guest machine will have no effect on the other machine.

- Now that the box has been added to Vagrant, we need to configure our project to use it as a base.
- Open the `Vagrantfile` and change the contents to the following:

```
Vagrant.configure("2") do |config|
  config.vm.box = "precise64"
end
```

- The `"precise64"` in this case must match the name you used to add the box above. This is how Vagrant knows what box to use.

Up and SSH

- To boot the guest machine, run the following:
  - `$ vagrant up`

- In less than a minute, this command will finish and you'll have a virtual machine running Ubuntu.
- Vagrant runs the virtual machine **without a UI**.

- To prove that it is running, you can SSH into the machine:
  - `$ vagrant ssh`
- Interact with the machine and do whatever you want.

- Be careful about `rm -rf /`, since Vagrant shares a directory at `/vagrant` with the directory on the host containing your `Vagrantfile`, and this can delete all those files.

- When you're done fiddling around with the machine, run `vagrant destroy` back on your host machine, and Vagrant will remove all traces of the virtual machine.

Syncing

- By default, Vagrant shares your project directory (remember, that is the one with the `Vagrantfile`) to the `/vagrant` directory in your guest machine:

```
$ vagrant up
$ vagrant ssh
vagrant@precise64:~$ ls /vagrant
Vagrantfile
vagrant@precise64:~$ touch /vagrant/foo
vagrant@precise64:~$ exit
$ ls
foo Vagrantfile
```

Networking

- We will use **port forwarding** (other options available).
- **Port forwarding** allows you to specify ports on the **guest** machine to share via a port on the **host** machine.

- Edit the `Vagrantfile`:

```
Vagrant.configure("2") do |config|
  config.vm.box = "precise64"
  config.vm.network :forwarded_port, host: 8080, guest: 8080
end
```

- Run a `vagrant reload` or `vagrant up` (depending on if the machine is already running) so that these changes can take effect.

Teardown

- Suspending the virtual machine:
  - `vagrant suspend` will save the current running state of the machine and stop it.
  - When you run `vagrant up`, the virtual machine will be resumed from where you left off.
  - Usually taking only 5 to 10 seconds to stop and start your work.
  - The virtual machine requires disk space to store all the state of the virtual machine RAM on disk.
- Halting the virtual machine
  - `vagrant halt` will gracefully shut down the guest operating system and power down the guest machine.
  - When you run `vagrant up` when you're ready to boot it again.
  - The contents of disk are preserved.
  - It'll take some extra time to start from a cold boot.
- Destroying the virtual machine
  - `vagrant destroy` will remove all traces of the guest machine from your system.
  - It'll stop the guest machine, power it down, and remove all of the guest hard disks.
  - The disk space and RAM consumed by the guest machine is reclaimed and your host machine is left clean.

Java SE 7

- Work on the host (physical machine)
  - Go to http://www.oracle.com/technetwork/java/javase/downloads/index.html
  - Click on Java Platform (JDK) 7u45
  - Read & accept License Agreement, download jdk-7u45-linux-x64.tar.gz
  - Move the tar.gz file in the vagrant project folder (`vagrant_lab`)

- Work on the guest (inside vagrant)

```
$ sudo mkdir -p /usr/lib/jvm

$ sudo tar zxvf /vagrant/jdk-7u45-linux-x64.tar.gz -C /usr/lib/jvm/

$ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.7.0_45/bin/java" 1
$ sudo update-alternatives --install "/usr/bin/javac" "javac" "/usr/lib/jvm/jdk1.7.0_45/bin/javac" 1
$ sudo update-alternatives --install "/usr/bin/javaws" "javaws" "/usr/lib/jvm/jdk1.7.0_45/bin/javaws" 1

$ java -version
java version "1.7.0_45"
Java(TM) SE Runtime Environment (build 1.7.0_45-b43)
Java HotSpot(TM) 64-Bit Server VM (build 24.0-b56, mixed mode)

$
```

Tomcat 6 (I)

- Work on the guest (inside vagrant)
  - Download

```
$ wget http://mirrors.ibiblio.org/apache/tomcat/tomcat-6/v6.0.37/bin/apache-tomcat-6.0.37.tar.gz
```

  - Install

```
$ sudo mkdir -p /usr/local
$ sudo tar zxvf apache-tomcat-6.0.37.tar.gz -C /usr/local/
```

  - Link

```
$ sudo ln -s /usr/local/apache-tomcat-6.0.37/ /usr/local/tomcat
```

  - Auto-start

```
$ sudo nano /etc/init.d/tomcat
```

Tomcat 6 (II)

```
#
# Startup script for the Tomcat server
#
# chkconfig: - 83 53
# description: Starts and stops the Tomcat daemon.
# processname: tomcat
# pidfile: /var/run/tomcat.pid
# See how we were called.
case $1 in
        start)
                export JAVA_HOME=/usr/lib/jvm/jdk1.7.0_40/
                export CLASSPATH=/usr/local/tomcat/lib/servlet-api.jar
                export CLASSPATH=/usr/local/tomcat/lib/jsp-api.jar
                export JRE_HOME=/usr/lib/jvm/jdk1.7.0_40/
                echo "Tomcat is started"
                sh /usr/local/tomcat/bin/startup.sh
        ;;
        stop)
                export JRE_HOME=/usr/lib/jvm/jdk1.7.0_40/
                sh /usr/local/tomcat/bin/shutdown.sh
                echo "Tomcat is stopped"
        ;;
        restart)
                export JRE_HOME=/usr/lib/jvm/jdk1.7.0_40/
                sh /usr/local/tomcat/bin/shutdown.sh
                echo "Tomcat is stopped"
                sh /usr/local/tomcat/bin/startup.sh
                echo "Tomcat is started"
        ;;
        *)
                echo "Usage: /etc/init.d/tomcat start|stop|restart"
        ;;
esac
exit 0
```

Tomcat 6 (III)

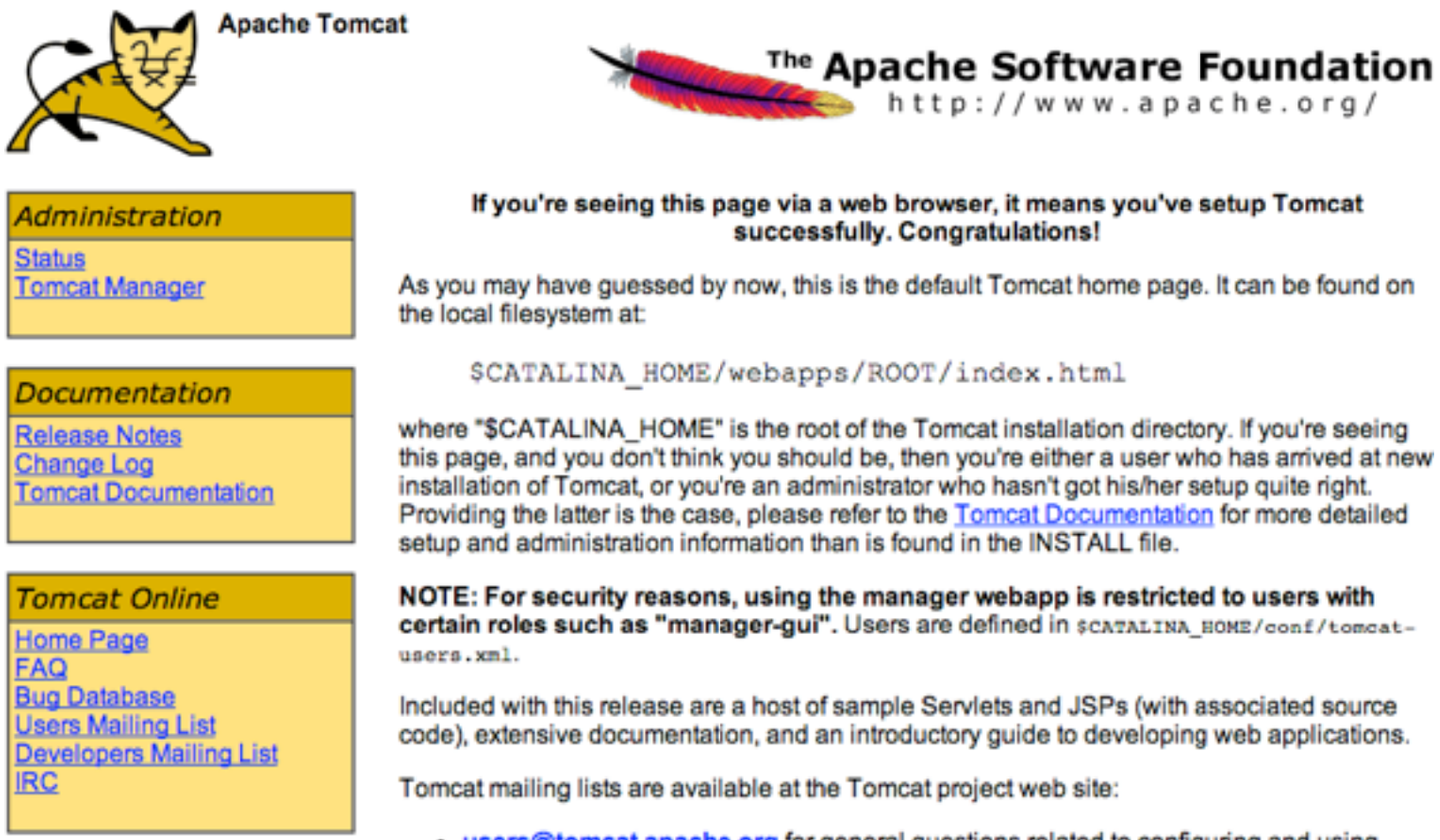
- Set Permissions

```
$ sudo chmod 755 /etc/init.d/tomcat
$ sudo update-rc.d tomcat defaults
```

- Start

```
$ sudo /etc/init.d/tomcat start
```

- Check (on host) at `http://localhost:8080`



ANT

- Apache Ant (http://ant.apache.org) is a tool to automatically compile Java programs and related tasks:
  - JAR creation
  - JUnit test execution
  - Javadoc creation
  - Application deployment
- Similar to Make but:
  - is pure Java
  - does not depend on shell
  - targets specifically Java code
- sudo apt-get -u install ant

ANT - Core concepts

- Ant is configured with an XML file with default name build.xml
- The configuration file describes a project
- A project can include one or more targets
- Each target includes a sequence of tasks

- From command line, the user specifies the configuration file and the target to run
  - if no configuration file is specified, using build.xml in current directory
  - if no target is specified, using the default target as specified in configuration

- Ant manages every dependencies among targets.

ANT Projects

- The root element of a build.xml file is <project> with its attributes
- The basedir attribute specifies the root of relative paths
- The default attribute specifies the name of the default target
- The name attribute specifies the name of the project

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="clean" name="Foo">

   <target name="clean">
   </target>

   <target name="compile">
   </target>

</project>
```

ANT Targets

- A target is defined with the element <target>
- The attribute name specifies the target's name
- The attribute depends lists the dependent targets
  - Dependent target are executed automatically before current target
  - Avoid cyclic dependencies
- In a target we specify the tasks composing the target

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="clean" name="Foo">

    <target name="compile">
    </target>

    <target name="build" depends="compile">
    </target>

</project>
```

- ant build will execute target compile, then target build.

ANT Tasks

- Each task is represented by an XML element
  - Attributes specify the task parameters
  - Some tasks require additional nested elements

Task javac

- The task javac compiles all java files in a given dir (including subdirs)
  - A java file is compiled only if more recent than the corresponding class file
  - Main attributes:
    - srcdir: source directory for java files
    - destdir: target directory for class files
    - classpath: to be used during compilation
  - Example:

```
<target name="compile" depends="init" description="Compile sources">
    <javac srcdir="src" debug="on" optimize="on" destdir="build"/>
</target>
```

Task java

- The task java executes a java class
  - Main attributes:
    - classname: name of the class containing the main
    - classpath: to be used during execution
    - fork: if true, will spawn a new JVM for execution
    - jvmarg: command line arguments to java command
    - arg: command line arguments to java class
  - Example:

```
<target name="run" depends="compile">
    <java classname="MainClass" fork="yes">
        <jvmarg line="-Xmx4G"/>
        <jvmarg line="-server"/>
        <arg line="args 3 12345"/>
    </java>
</target>
```

ANT Properties

- It is possible to define variables, called properties, using the <properties> task
  - Can be nested or not
- Main attributes:
  - name
  - value
- The value of a property is referenced with the notation ${name}
- Example

```
<property name="base.dir"    value="." />
<property name="jar.dir"     value="${base.dir}/jars" />
<property name="src.dir"     value="${base.dir}/src" />
<property name="classes.dir" value="${base.dir}/classes" />
```

Other ANT tasks

- <mkdir dir="name"/>
  - Create a directory
- <echo message="text"/>
  - Display a message
- <delete dir="classes"/>
  - Delete a directory (recursively)
- <delete file="foo.txt"/>
  - Delete a file

Specifying classpaths and filesets

```
<project name=“my-project" default="compile" basedir=".">

    <property name="jars"  value=“${basedir}/jars"/>
    <property name="build" value=“${basedir}/classes"/>
    <property name="src"   value="${basedir}/src"/>

    <path id="compile.classpath">
        <fileset dir=“${jars}"/>
    </path>

    <target name="init">
        <mkdir dir=“${build}"/>
    </target>

    <target name="compile" depends="init" description="Compile sources">
        <javac srcdir="${src}" debug="on" optimize="on" destdir=“${build}"
         classpathref="compile.classpath"/>
    </target>

    <target name="jar" depends="compile" description="Creates jar">
        <jar jarfile=“my-project.jar">
            <fileset dir="${build}"/>
        </jar>
    </target>

    <target name="clean">
        <delete dir="${build}"/>
    </target>
</project>
```

MAVEN

- Command line tool
- Manages dependencies like a package management system
- Performs builds, either stand-alone or via Ant integration
- Runs JUnit tests and generates reports
- http://maven.apache.org
- sudo apt-get -u install maven
- (sudo apt-get -u install curl)