FreeBSD Test Cluster Automation

Kamil Czekirda
Warsaw University of Technology

Abstract

"FreeBSD Test Cluster Automation" is a Google Summer of Code 2015 project for FreeBSD organization to create an infrastructure for automated tests building, installing and first booting process of FreeBSD.

The base of this project is iPXE - Open Source Boot Firmware, which is used for controlling nodes. A small webapplication written in python is a frontend for the database where information about nodes, current states and states of revisions are saved. The project is also using mfsBSD and bsdinstall extension for an automatic and non-interactive installation process, it was done during Google Summer of Code 2014.

On the server side the main part of the project is FreeNAS, it is used to provide shared storage and jails for applications. The ZFS filesystem with deduplication enabled on dataset for source code allows to save every tested revision of the source code with space saving.

The scope of the project was only infrastructure, without focusing on tests. During the project simple tests of building and installing FreeBSD were made, it's similar to https://jenkins.freebsd.org/ but on the bare metal infrustructure and it's possible to test all commits, not all commits from one period of time like in jenkins.

Another interesting application for this project is testing drivers, for example network card drivers. Inside testing cluster it's possible to build a driver after any commit, test it, measure and report.

The most important requirement during this project was as little intervention as possible.

1 iPXE port

iPXE is open source network boot firmware, it provides a full PXE implementation extended with additional features such as:

- boot from a web server via HTTP and HTTPS
- boot from iSCSI
- boot from wireless network

And the most important for this project is to control the boot process with scripts.

The firts stage of the project was creating iPXE port for FreeBSD. The port is ready for submition and has many possibilities for extensions.

2 Servers side

The Preboot eXecution Environment allows to boot from a network interface. Host broadcasts a DHCP discover a request and a DHCP server responds with a DHCP packet that includes PXE options (the name of a boot server and a boot file). The client downloads his boot file by using TFTP and then executes it. In this project it is iPXE loader and this is classical chainloading of iPXE. In the next step iPXE loads MEMDISK kernel with the location of modified mfsBSD iso file as its parameter and then nodes mount shared storage via NFS protocol.

As you can see, there are a lot of services to configure:

- DHCP server
- TFTP server
- HTTP server
- NFS server
- Management application

The first step of booting node from the network is DHCP service. DHCP server responds with a DHCP packet that included PXE options, in this case the name of TFTP boot server and a boot file.

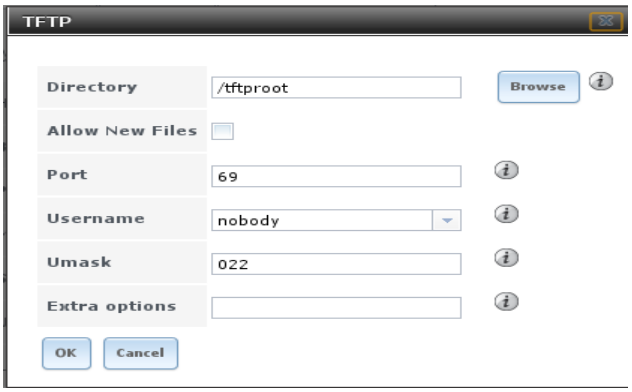
An example of the dhcp server configuration:

```
subnet 192.168.22.0 netmask 255.255.255.0 {
  range 192.168.22.10 192.168.22.50;
  option routers 192.168.22.1;
  option domain-name-servers 192.168.22.1;
  next-server 192.168.22.19;
  if exists user-class and ( option \\
   user-class = "iPXE" ) {
    filename "http://192.168.22.3/menu.ipxe";
  }
  else {
    filename "undionly.kpxe";
  }
}
```

In this case we can see, that TFTP server is located on 192.168.22.19 IP address, filename is different and depends on client user-class. iPXE image (filename "undionly.kpxe") is handed when the DHCP request comes from a legacy PXE client. In the next step request sends iPXE DHCP client with user-class iPXE and answer in filename option is the url with menu.ipxe script.

2.1 TFTP Server

Trivial File Transfer Protocol (TFTP) is a service used for transfer iPXE image compiled from the port. Nodes download the image from the TFTP server each time they boot. In my project I use FreeNAS and TFTP configuration screen shows default configuration and it is sufficient.



2.2 HTTP Server

HTTP server is used for serving iso image of custom mfsBSD and initial script: menu.ipxe. In my case it's apache in the jail on the FreeNAS box.

2.3 NFS Server

The NFS service is provided by FreeNAS. It's a storage for source code. If node have not enough RAM memory can also save obj files there. NFS export is stored on the ZFS filesystem. The dataset has enabled deduplication. This configuration allows to have access to every revision of the source code without switching beetween revisions in repository.

2.4 Management

The frontend of management application is written in python with bottle framework. Information about nodes and revisions is saved in the sqlite database. The management is the place, where the user can manage nodes and revisions and it works as http server. Application supports methods:

- / to provide default ipxe script
- /admin - it's main dashboard
- /admin/add_node
- /admin/edit_node/:id
- /admin/delete_node/:id
- /admin/add_task
- /admin/delete_task/:id
- /menu/:mac to send static ipxe script whose name is saved in the database
- /static/ to provide static files
- /admin/take_task/:mac to start environment preparing
- /admin/change_boot/:host/:new to change boot ipxe script
- /admin/change_task_status/:revision/:new_status
- /admin/change_node_status/:hostname/:new_status

Example screenshot you can see at Figure 1.

3 Client side

From client side there is only one thing I have to carry on - set network card as the first booting device. The iPXE uses script and decides which is the next step on booting is - its either hard drive or network.

4 mfsBSD configuration

mfsBSD configuration is very simple, because I added only these lines to mfsbsd/conf/rc.local.sample file:

```
sleep 10
mkdir /cluster
mount -t nfs -o nolockd 192.168.22.19:/mnt/tank \\
/freebsd/$(hostname)/cluster /cluster
sh -x /cluster/run.sh > /cluster/run.log 2>&1 &
```

Node mounts storage and runs cluster script, where other instructions are.

5 iPXE scripts

For control the boot process on nodes I have four ipxe scripts (take_task, wait, cluster and hdd). The first of them is take_task.ipxe:

```
#!ipxe

set www 192.168.22.3
set port 8080

chain http://${www}:${port}/admin/ \\
      take_task/${net0/mac}
```

In this script node sends request to management application and tells them that it is clean and it is ready to take new task. Very important parameter is mac address of the network card. The management uses this parameter to search which is the next one ipxe script (wait, cluster or hdd).

The second script is wait.ipxe:

```
#!ipxe
set www 192.168.22.3
set port 8080
set timeout 120000

:menu
menu Creating environment, please wait...
item next Please wait...
choose --timeout ${timeout} selected
goto ${selected}

:next
chain http://${www}:${port}/menu/${net0/mac}
```

This script is the infinite loop. Every 120 seconds node asks the management for new ipxe script. During this time the management is preparing environment (creating directories for revision, copying the source tree etc).

When server finishes preparing environment ipxe script for node changes to cluster.ipxe:

```
#!ipxe
set timeout 10000

set www 192.168.22.3
set iso mfsbsd_cluster.iso

sanboot --drive 0x81 --no-describe http://${www}/${iso}
```

and node boots from mfsBSD iso and do tests.

When all tests are fine cluster script changes node status (and ipxe script) to hdd:

```
#!ipxe
set timeout 10000

sanboot --drive 0x80 --no-describe
```

and node boots from HDD drive.

The last change is reseting node and set take_task.ipxe as a script to run.

6 Workflow

This is complete workflow for the node and revision.

6.1 The node

- the node starts netbooting from take task status
- in the first step of PXE booting node sends DHCP request and DHCP server responds with `next-server` and `filename` options and node knows what and where from to download.
- the node downloads iPXE loader binary by TFTP protocol and executes it
- iPXE sends DHCP request and gives an answer with a different filename option - url to iPXE starting script
- iPXE starting script asks the management for chainloading next script and authorizes itself by mac address
- management return `take_task.ipxe` file
- `take_task.ipxe` runs next chainloading and node waits for environment preparation, in this time on server side script `take_task.sh` prepares files (update svn, rsync to new src space)
- the node chainloads `cluster.ipxe` script and node starts mfsBSD
- the node mounts storage from NFS server and building process starts (`make buildworld`, `make buildkernel` and `make ftp`)

- after building the node tries to install the system on hard drive from files compiled and created during this task
- the node reboots and boots from hdd
- the node reboots and boots from the network like in first step

If any step from building, installing or booting stage fails then the node starts netbooting and takes a new task.

Diagram of states you can see at Figure 2.

6.2 Revision

- the first status of revision is NEW, in this status revision awaits for free node to take a task
- when the node starts netbooting and revision is first in the queue status changes to preparing
- in the next steps revision is tested by compilation, installation and boot
- revision is marked as success or failed and logs of every steps are available on the management server

Diagram of states you can see at Figure 3.

7 Urls

- https://wiki.freebsd.org/SummerOfCode2015/FreeBSDTestClusterAutomation
- https://svnweb.freebsd.org/socsvn/soc2015/kczekirda/
- http://ipxe.org/

Nodes

ID	host	mac address	ip address	boot	status		
1	node01		192.168.22.51	cluster.ipxe	buildworld	Editt	Delete
2	node02		192.168.22.52	cluster.ipxe	buildworld	Editt	Delete
3	node03		192.168.22.53	cluster.ipxe	rebooting	Editt	Delete
5	d430		192.168.22.54	cluster.ipxe	buildkernel	Editt	Delete

Add node

Tasks

ID	revision	host	status	
2	r286981	node01	done	Delete
3	r286982	node03	done	Delete
4	r286983	d430	buildkernel	Delete
5	r286984	node02	buildworld	Delete
7	r286986	node03	done	Delete
12	r286991	node03	running	Delete
13	r286992	node01	buildworld	Delete
14	r286985		new	Delete
16	r286995		new	Delete
17	r286997		new	Delete
18	r286998		new	Delete
19	r286993		new	Delete
20	r286999		new	Delete
21	r287000		new	Delete
22	r287001		new	Delete

Add task

Figure 1: Dashboard screenshot.

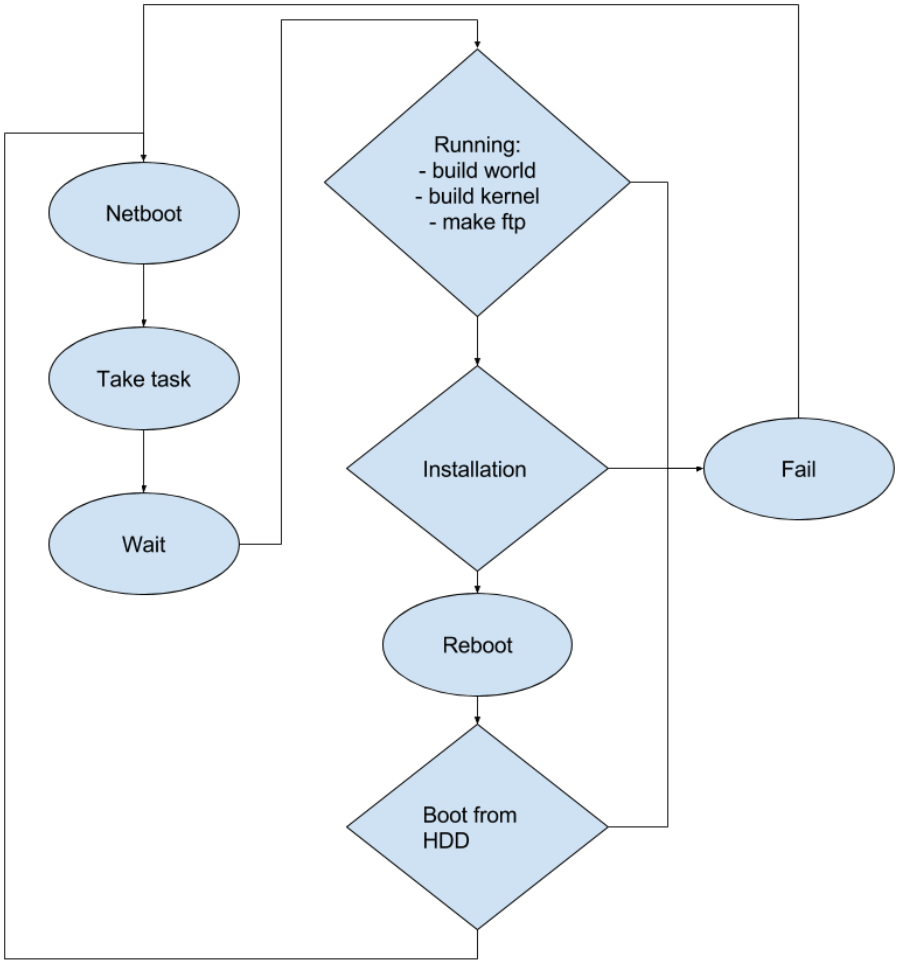


Figure 2: Nodes states diagram

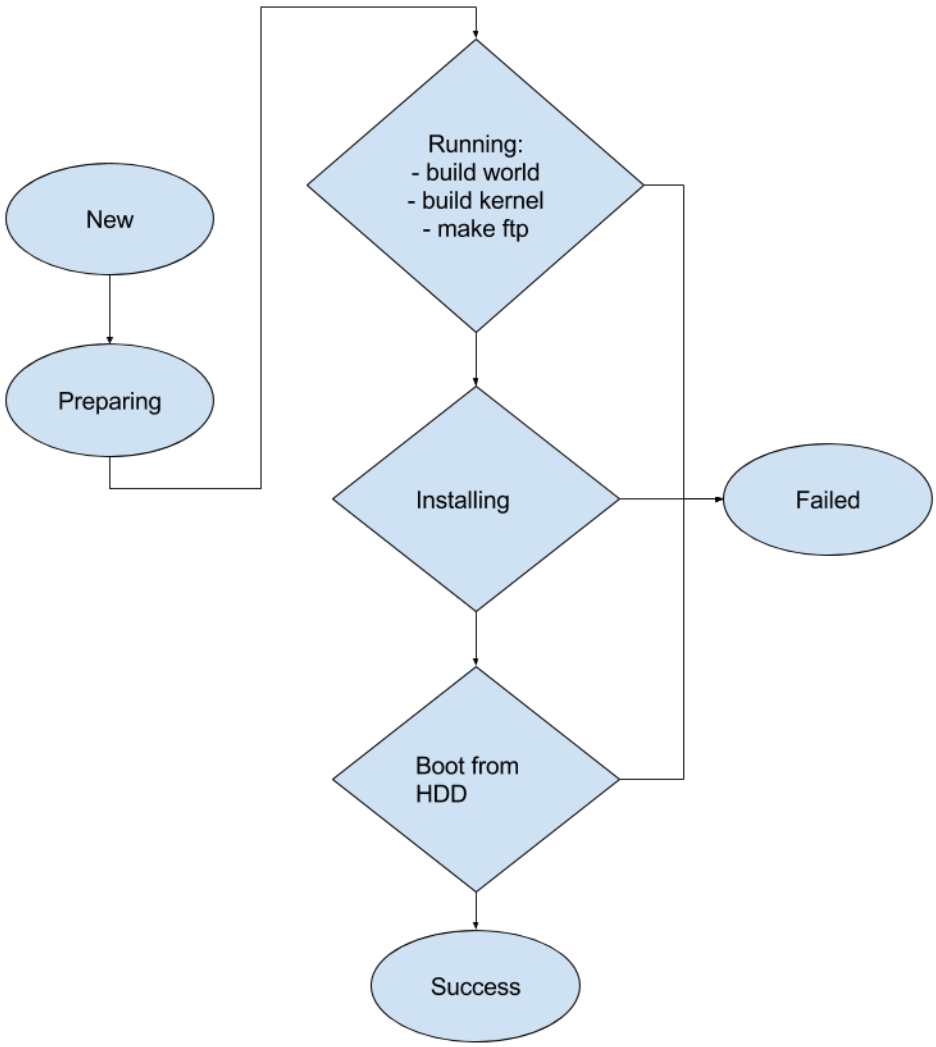


Figure 3: Revisions states diagram