Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

TI - HADOOP

Cluster Documentation

Department Maschinenbau und Produktion
Fakultät Technik und Informatik
Hochschule für angewandte Wissenschaften Hamburg

Verantwortlich:
Prof. Dr. rer. nat. Sarah Hallerberg
sarah.hallerberg@haw-hamburg.de

Administrator:
Tobias Schröder
tobias.schroeder@haw-hamburg.de

Stand: Januar 2020

Table of Contents

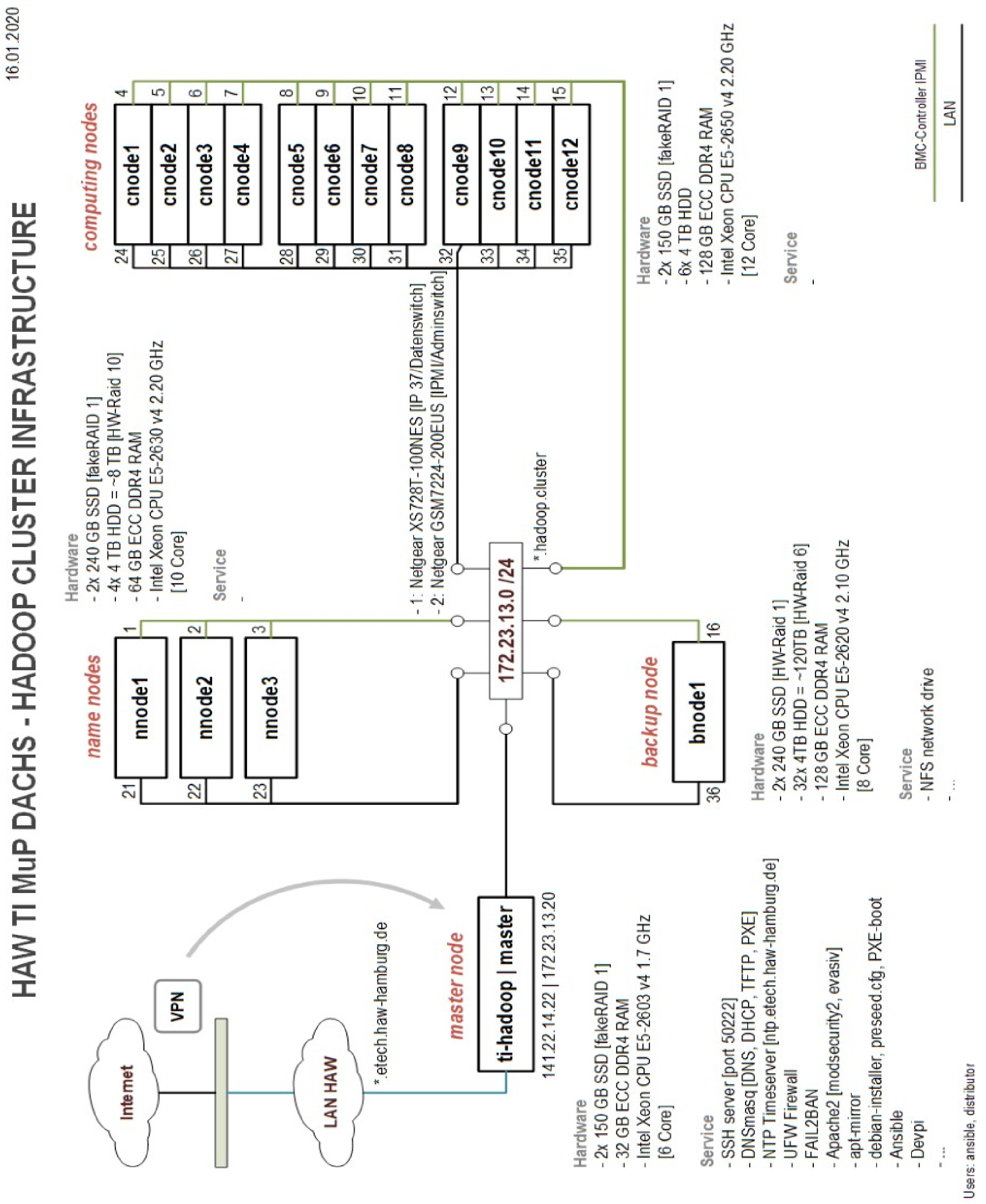
1 Changelog

2020/01/24	Documentation created

2 ToDo

2020/01/24	Installation of the Hortonworks Hadoop Distribution	

3 Architecture and Overview


16.01.2020
HAW TI MuP DACHS - HADOOP CLUSTER INFRASTRUCTURE
Internet
VPN
LAN HAW
*.etech.haw-hamburg.de
master node
ti-hadoop | master
141.22.14.22 | 172.23.13.20
Hardware
- 2x 150 GB SSD [fakeRAID 1]
- 32 GB ECC DDR4 RAM
- Intel Xeon CPU E5-2603 v4 1.7 GHz
[6 Core]
Service
- SSH server [port 50222]
- DNSmasq [DNS, DHCP, TFTP, PXE]
- NTP Timeserver [ntp.etech.haw-hamburg.de]
- UFW Firewall
- FAIL2BAN
- Apache2 [modsecurity2, evasiv]
- apt-mirror
- debian-installer, preseed.cfg, PXE-boot
- Ansible
- Devpi
- ...
Users: ansible, distributor
name nodes
nnode1
nnode2
nnode3
1
2
3
21
22
23
Hardware
- 2x 240 GB SSD [fakeRAID 1]
- 4x 4 TB HDD = ~8 TB [HW-Raid 10]
- 64 GB ECC DDR4 RAM
- Intel Xeon CPU E5-2630 v4 2.20 GHz
[10 Core]
Service
-
172.23.13.0 /24
- 1: Netgear XS728T-100NES [IP 37/Datenswitch]
- 2: Netgear GSM7224-200EUS [IPMI/Adminswitch]
*.hadoop.cluster
backup node
bnode1
16
36
Hardware
- 2x 240 GB SSD [HW-Raid 1]
- 32x 4TB HDD = ~120TB [HW-Raid 6]
- 128 GB ECC DDR4 RAM
- Intel Xeon CPU E5-2620 v4 2.10 GHz
[8 Core]
Service
- NFS network drive
- ...
computing nodes
cnode1
cnode2
cnode3
cnode4
cnode5
cnode6
cnode7
cnode8
cnode9
cnode10
cnode11
cnode12
4
5
6
7
8
9
10
11
12
13
14
15
24
25
26
27
28
29
30
31
32
33
34
35
Hardware
- 2x 150 GB SSD [fakeRAID 1]
- 6x 4 TB HDD
- 128 GB ECC DDR4 RAM
- Intel Xeon CPU E5-2650 v4 2.20 GHz
[12 Core]
Service
-
BMC-Controller IPMI
LAN

3.1 System Structure

- The master/ti-hadoop server serves as the gateway to the cluster infrastructure, for the system management and for general services.
- Inside the cluster network there are 3 ‘Namen-Node’ server (nnode1-3) which fulfill a special role in a *Hadoop*-Environment.
- There are 12 ‘Computing-Node’ server (cnode1-12) for the actual calculations and computations. Physically, there are always 4 nodes in one chassis. Only this Computing-Nodes should be handed to users. They can use them for calculations with e.g. *Python*, *C*,… or whatever software. (Name-Node, Backup-Node are not for users to work on.)
- Additionally there is a ‘Backup-Node’ (bnode1) with network drives for data and backups.
- The operating system (OS) on all hosts is the *Linux* distribution ‘*Debian*’ in the latest stable version 9.5 (*Stretch*).

3.2 Hardware

- <u>Master/TI-Hadoop (Management):</u>
  1x sysGen/SUPERMICRO SuperServer SYS-5018R-M
  - Mainboard Supermicro X10SRi-F with Intel Ethernet Controller Intel i350-AM2 Dual port GbE LAN, Realtek RTL8211E PHY (dedicated IPMI)
  - 1x Intel Xeon Broadwell-EP Series Processor E5-2603 v4, 1.70GHz, 6 core, Socket 2011-3, 15MB Cache, max. 1866MHz DDR4, QPI 6.4GT/s, 85Watt
  - 32GB (4x 8GB) DDR4 PC2133 ECC
  - System SSDs (SW-RAID1 [BIOS-RAID/FakeRaid] ~ 150GB)
    → 2x 150GB Intel SSD DC S3520 150GB SATA6Gb/s 2.5”
  - 19” Supermicro SuperChassis 813MTQ-350CB, 1U
- <u>Name-Nodes:</u>
  3x sysGen/SUPERMICRO SuperServer SYS-6028R-TRT
  - Mainboard Supermicro X10DRi-T with Intel X540 Dual port 10GBase-T, Realtek RTL8211E PHY (dedicated IPMI)
  - 2x Intel Xeon Broadwell-EP Series Processor E5-2630 v4, 2.20GHz, 10 core, Socket 2011-3, 25MB cache, max. 2133MHz DDR4, QPI 8.0GT/s, 85Watt
  - 64GB (8x 8GB) DDR4 PC2133 ECC
  - System SSDs (HW-RAID 1 ~ 240GB)
    → 2x 240GB Intel SSD DC S3520 240GB SATA 6Gb/s 2.5”
  - Data HDDs (HW-RAID 10 ~ 8TB)
    → 4x 4TB Toshiba Enterprise MG04ACA400E 512e 4TB SATA 6GB/s 3,5”

  - RAID-Controller LSI MEGARAID SAS9361-4I SGL 4-Port 12Gb/s SATA+SAS PCIe 3.0
  - 19” Supermicro SuperChassis SC825TQ-R740LPB, 2U

- Computing-Nodes:
  3x sysGen/SUPERMICRO FAT TWIN SYS-F628R3-RTBPTN+
  → 4 hot-pluggable nodes in a 4U chassis
  - 4x Mainboard Supermicro X10DRFR-NT with Intel X540 Dual port 10GBase-T, Realtek RTL8211E PHY (dedicated IPMI)
  - 8x Intel Xeon Broadwell-EP Series Processor E5-2650 v4, 2.20GHz, 12 core, Socket 2011-3, 30MB cache, max. 2400MHz DDR4, QPI 9.6GT/s, 105Watt
  - 4x 128GB (8x 16GB) DDR4 PC2400 ECC
  - System SSDs (SW-RAID 1 [BIOS-RAID/FakeRaid] ~ 150GB)
    → 8x 150GB Intel SSD DC S3520 150GB SATA 6Gb/s 2.5”
  - Data HDDs
    → 24x 4TB Toshiba Enterprise MG04ACA400E 512e 4TB SATA 6GB/s 3,5”
  - 19” Supermicro SuperChassis CSE-F424AS-R1K28B, 4U

- Backup-Node:
  3x sysGen/SUPERMICRO BTO Superserver
  - Mainboard Supermicro X10DRH-iT with Intel X540 Dual port 10GBase-T, Realtek RTL8211E PHY (dedicated IPMI)
  - 2x Intel Xeon Broadwell-EP Series Processor E5-2620 v4, 2.10GHz, 8 core, Socket 2011-3, 20MB cache, max. 2133MHz DDR4, QPI 8.0GT/s, 85Watt
  - 128GB (8x 16GB) DDR4 PC2400 ECC
  - System SSDs (HW-RAID 1 ~ 240GB)
    → 2x 240GB Intel SSD DC S3520 240GB SATA 6Gb/s 2.5”
  - Data HDDs (HW-RAID 6 ~ 120TB)
    → 32x 4TB Toshiba Enterprise MG04ACA400E 512e 4TB SATA 6GB/s 3,5”
  - RAID-Controller LSI MEGARAID SAS9361-8I SGL 8-Port 12Gb/s SATA+SAS PCIe 3.0
  - 19” Supermicro SuperChassis CSE-847BE1C-R1K28LPB, 4U

- Dataswitch:
  NetGear ProSAFE XS728T-100NES, managed, L2+/L3 Lite, VLAN, 24x 10GBase-T + 4x 10Gb/s SFP+, 560Gb/s

- IPMI/Admin Switch:
  NetGear ProSAFE GSM7224-200EUS, managed, L2, VLAN, 24x 1Gb/s + 4x 1Gb/s SFP, 48Gb/s

4 Network

The master/ti-hadoop server is the gateway/access point to the cluster. So it is physically connected to the *HAW*-network with a public IP-adress and to the clusters local area network (*LAN*) with a private class C subnet 172.23.13.x/24.

- ti-hadoop.etech.haw-hamburg.de ↔ 141.22.14.22 /22 [eth0] (*HAW*-network)
- master.hadoop.cluster ↔ 172.23.13.20 /24 [eth1] (private LAN)

The master server manages the *LAN* by *DHCP* and *DNS* services. So the clients get an *IP*-adress and a *DNS*-name automatically. The clients are connected to the clusters *LAN* with the interface eth0. The *DHCP*-service is configured to ensure that the clients get always the same *IP*-address and *DNS*-name. This is hard-coded by the clients *MAC*-addresses and a dedicated *IP*-address with an infinite lease time.

- nnode1-3.hadoop.cluster ↔ 172.23.13.21-23 /24
- cnode1-12.hadoop.cluster ↔ 172.23.13.24-35 /24
- bnode1.hadoop.cluster ↔ 172.23.13.36 /24

The network controllers inside the cluster are connected to the data switch *NetGear ProSAFE XS728T-100NES*, which is linked to the *IPMI*/Admin switch *NetGear ProSAFE GSM7224-200EUS*.

Hint: So far, there is no internet connection from inside the local *LAN* for the cluster clients. If it should be necessary one day, eventually a web-proxy on the master/ti-hadoop server is appropriate. Then it could be thought about if a *Debian* repository mirror and the *Devpi PyPi* caching mirror are still relevant.

4.1 IPMI 2.0 (Intelligent Platform Management Interface)

All cluster clients (nnodes, cnodes, bnode) have a second connection to the clusters network by their *IPMI*-Interface. *IPMI* is a standardized computer hard- and firmware interface to manage (remote control) and monitor servers on a hardware level, independent of the host system's *CPU*, firmware (*BIOS*/*UEFI*) and operating system (OS). Thus it allows to manage a server that may be powered off or is unresponsive by network connection or the OS. Another use case is the remote installation of an OS.

- nnode1-3 ↔ 172.23.13.1-3 /24
- cnode1-16 ↔ 172.23.13.4-15 /24
- bnode1 ↔ 172.23.13.16 /24

For the *IP*-addresses the same *DHCP* rules are applied (s. 4 Network). The *IPMI*-interfaces are connected to the *IPMI*/Admin switch *NetGear ProSAFE GSM7224-200EU*.

A management software is required to get access via the *IPMI*-interface. So far used are the tools *IPMIView* and *SMCIPMITool* of the server company *Supermicro*, which can be downloaded from: https://www.supermicro.com/SwDownload/SwSelect_Free.aspx?cat=IPMI

They are *Java* based tools and have to be run at the master/ti-hadoop server.

1. To use the *IPMI* Software with a *GUI* its mandatory to run the *SSH* connection to the master/ti-hadoop server with a graphical interface, e.g. `ssh -X <USER>@<HOST>`. (In Windows its necessary to install and configure e.g. *Xming* additional to *Putty*.)

2. Copy *IPMIView* from `/media/software` or download the newest version to the desired folder in your user folder and extract it `tar -xvzf <ARCHIVE>`.

3. Start e.g. *IPMIView* with `java -jar IPMIView20.jar`.

4. Scan for connected devices (magnifying glass), type in the desired *IP*-range or leave empty. Select all found devices (Strg+a) and save the devices, they should now be shown in the list '*IPMI* Domain'. Exit the scanning window and "Save Configuration" (Floppy). To open a device, double click on the *IP*-adress in the "*IPMI* Domain" list. User: ADMIN, Password: ADMIN

5. Use the Tabs at the bottom of the Screen to navigate between menus. To power up or down a device, navigate to the desired one. To monitor the screen output, go to the text console or establish a *KVM* (Keyboard-Video-Mouse) console and press start.

Note: The *KVM* console may not work properly with *IPMIView*. With a " `&`" symbol at the end of a command, the command will run in background. Don't forget to stop it if finished.

If that should be the case, try

```
sudo XAUTHORITY=/home/<USER>/.Xauthority java -jar IPMIView20.jar
```

or use *SMCIPMI* with a dedicated separate *KVM* console.

```
sudo XAUTHORITY=/home/<USER>/.Xauthority java -jar SMCIPMITool.jar <IPMI-IP>
<USER> <PASSWORD> ukvm.
```

Hint: It can take 'forever' to open a Java *KVM* console over the network, keep patient for around 10 – 15 min until it opens and is somehow 'usable'.

4.2 Access Restrictions and Security

So far the clusters network is only accessibly via *SSH* protocol through the master/ti-hadoop server. For security reasons the *SSH* server is listening on the non-reserved port 50222. Furthermore it can only be reached if a client owns a public *IP*-address of the *HAW*-network 141.22.14.x /22. Thus a *VPN* connection is necessary to get access from outside the *HAW*-network.

Additionally the master/ti-hadoop server is protected by a *Netfilter-firewall* managed with the command line based frontend *UFW (Uncomplicated Firewall)* for easier configuration. *UFW* uses the complicated *iptables* for configuration. All incoming traffic is denied, except the open *SSH* port 50222. (s. A.6 Master / TI-Hadoop)

To prevent brut force attacks, an intrusion prevention software *Fail2ban* is installed. This software can ban any host *IP*-address that makes too many login attempts or performs any other unwanted action within a defined time frame. It can be configured for different services, like e.g. *sshd*, *Apache2*, … and it uses as well *iptables*. (s. A.6 Master / TI-Hadoop).

5 Cluster management

The master/ti-hadoop server is dedicated for cluster management and general services. In this chapter the chosen concept of the cluster management will be explained.

An automated Linux Debian installation is configured and can be applied to all nodes (s. 5.2 Automated Linux Debian Installation).

For the cluster management (client and user) the open-source software *Ansible* is used, an automatization tool for computer orchestration and general configuration and administration (s. https://www.ansible.com/). The master/ti-hadoop server itself is not included as client to the management software.

5.1 Ansible

So far and to keep the complexity of the cluster as minimal as possible the powerful software *Ansible* is only used with playbooks and ad-hoc commands. That means, that for a repeating purpose there exists a playbook or has to be created, which then applies configurations from the control server to the managed nodes, if executed. A playbook is a script based configuration file written in *YAML* that provides instructions for what needs to be done in order to bring a managed node into the desired state with the help of *Ansible* modules performing specific tasks. (*Ansible* is by far more powerful than what is shown here, beside the shown commands, there exist several further commands.)

Ansible works agentless over *SSH*. For this purpose there exists a special user on the master/ti-hadoop server and on the managed nodes (clients). During the automated Linux *Debian* installation the user on the client side gets created and a related public *SSH* key established. The private *SSH* key remains in the users directory of the master/ti-hadoop server. This allows a password-less single sing-on.

- distributor ↔ master/ti-hadoop server
- ansible ↔ nodes (client side)

All playbooks can be found in the directory `/home/distributor/playbooks`. The directories and files are dedicated to the user “distributor” and only this user should have directory and file permission (e.g. directory = `drwxr-x--- distributor distributor`, file = `-rw-r----- distributor distributor`). Inside that directory exist different categorized subdirectories like ‘clientconfs’, ‘post-install’, ‘user_mgmt’.

Inventory

A file (by default, *Ansible* uses a simple *INI*-format) that describes hosts and groups in *Ansible*. Inventory can also be provided via an inventory script.

Role

Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules. A role is a group of variables, tasks, files, and handlers that are stored in a standardized file structure. In *Ansible*, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing complex playbooks, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.

Modules

The units of code *Ansible* executes. Each module has a particular use, from administering users on a specific type of database to managing *VLAN* interfaces on a specific type of network device. You can invoke a single module with a task, or invoke several different modules in a playbook. For an idea of how many modules *Ansible* includes, take a look at the list of all modules.

Modules are the units of work that *Ansible* ships out to remote machines. Modules are kicked off by either `/usr/bin/ansible` or `/usr/bin/ansible-playbook` (where multiple tasks use lots of different modules in conjunction). Once modules are executed on remote machines, they are removed, so no long running daemons are used.

Tasks

The units of action in *Ansible*. You can execute a single task once with an ad-hoc command. Playbooks exist to run tasks. Tasks combine an action (a module and its arguments) with a name and optionally some other keywords (like looping directives). Handlers are also tasks, but they are a special kind of task that do not run unless they are notified by name when a task reports an underlying change on a remote system.

Playbooks

Ordered lists of tasks, saved so you can run those tasks in that order repeatedly. Playbooks can include variables as well as tasks. Playbooks are written in *YAML* and are easy to read, write, share and understand.

Playbooks are usually **idempotent**. An operation is idempotent if the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening actions. For *Ansible* it means after 1 run of a playbook to set things to a desired state, further runs of the same playbook should result in 0 changes, if the playbook hasn’t been changed meanwhile. In simplest terms, idempotency means you can be sure of a consistent state in your environment.

Commands

The *Ansible* commands are quite complex. Thus some important commands to understand the concept will be explained, for more details see the *Ansible* documentation https://docs.ansible.com/ (or `ansible --help`, `ansible-playbook --help`).

Ad-hoc commands:

Its possible to perform a single task once with an ad-hoc command by invoking a single module. The standard module is ‘command’. Useful to perform rare tasks, e.g. reboot servers, copy files, manage packages.

Ad-hoc refers to running *Ansible* to perform some quick command, using `/usr/bin/ansible`, rather than the orchestration language, which is `/usr/bin/ansible-playbook`. An example of an ad-hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad-hoc can be accomplished by writing a playbook. Playbooks can also glue lots of other operations together.

```
ansible <pattern> -m <module> -a "<module options>"
```

```
ansible all -m ping -u ansible
```

→ All defined hosts get pinged to check availability. The module ‘ping’ is used and the remote user ‘ansible’ is used to connect as this user (otherwise it will default to run from the user account used).

```
ansible cnode1 -a "/sbin/reboot" -u ansible –become –ask-become-pass
```

→ Reboots `cnode1`. Instead of a single host, multiple hosts `cnode1:cnode2:cnode3` or a group (if defined, a group consists of several hosts assigned to a pool that can be conveniently targeted together, as well as given variables that they share in common) can be addressed. Here the command module is used and the command `/sbin/reboot` gets performed on the host. The user `ansible` is used to log-in and `--become` (does not imply password prompting) for privilege escalation by switching to the “root” (default user for privilege escalation). The option `--ask-become-pass` gives a password prompt.

https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html

Executing playbooks:

Playbooks are a completely different way to use *Ansible* than in ad-hoc task execution mode, and are particularly powerful. Simply put, playbooks are the basis for a really simple configuration management and multi-machine deployment system, unlike any that already exist, and one that is very well suited to deploying complex applications. Playbooks can declare configurations, but they can also orchestrate steps of any manual ordered process, even as different steps must bounce back and forth between sets of machines in particular orders. They can launch tasks synchronously or asynchronously. While you might run the main `/usr/bin/ansible` program for ad-hoc tasks, playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

```
ansible-playbook playbook.yml <options>
```

```
ansible-playbook install-base-packages.yml --ask-become-pass
```

→ Executes the playbook ‘install-base-packages.yml’ and gives a password prompt for privilege excalation. In comparison to the ad-hoc commands, everything else is defined inside the playbook.

```
ansible-playbook -l cnode10 pip_conf_for_devpi.yml --ask-become-pass
```

→ Usually the hosts, where the playbook is executed at, are defined inside the playbook. If you want to run the playbook instead only on one or multiple hosts, the `-l` limit-option makes it possible.

```
ansible-playbook playbook.yml --list-hosts
```

→ Outputs a list of matching hosts declared inside the playbook. Does not execute the playbook.

```
ansible-playbook playbook.yml --syntax-check
```

→ This will run the playbook file through the parser to ensure its included files, roles, etc. have no syntax problems.

Hint: *YAML* syntax is highly sensitive.

https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#playbooks-intro

5.2 Automated Linux Debian Installation

To (re-) install a new Linux *Debian* automatically on one of the several nodes or on a new node its recommended to do it over *IPMI*. So there is no need to go physically to the server room. For how to use the *IPMI* software, s. (4.1 IPMI 2.0 (Intelligent Platform Management Interface)).

(If a complete new server should be installed its wisely to first check the *DHCP* and *DNS* settings of the *DNSmasq* service for a fixed *IP* ↔ *MAC*-address combination and desired hostname. Otherwise the system will get a random *IP*-address out of the configured *DHCP*-pool.)

1. Establish a *KVM* connection over *IPMI* software to the desired device.
2. Boot or reboot the device.
3. The boot menu of the nodes is configured to always boot from the network with *PXE* (Preboot Execution Environment) as first option. After booting with *PXE* the node gets an IP-address and several other information from the master/ti-hadoop server by *DHCP* (dnsmasq service). Told by the *DHCP* information a small bootable *PXELinux* will be loaded from the told *TFTP* (Trivial File Transfer Protocol) server, showing the install menu.
4. The important entries in the install menu are:
   - Automatic Install *Debian* 9.5 (*Stretch*) → choose for automatic installation (uses a configuration file with predefined answers for the install process [preseed.cfg])
   - Install → choose for manual installation
   - Boot from local disk → standard selection if no interrupt (timeout after 300s)

After the automated installation some post installation procedures for system configuration have to be done, s. 5.3 Client Management.

5.3 Client Management

So far there exist several playbooks with different purposes to perform tasks on the nodes. Sometimes a playbook is dedicated to all nodes and performs the same task(s). Sometimes the playbook distinguishes between 'nnode' and/or 'cnode' and/or 'bnode' and performs similar but different tasks on the devices. Some playbooks may be specifically dedicated to one of them. This can be always seen inside the playbook as well as a short description of the playbooks purpose.

The existing playbooks are categorized as follows and lay in the home directory of the user 'distributor' and the directory permissions:

```
distriburor@ti-hadoop:/home/distributor/playbooks#
```

```
drwxr-x---  5 distributor distributor 4096 Dec 20 14:01 playbooks
```

The directory/file structure is as follows:

```
├── clientconfs
│  ├── bash.bashrc
│  └── pip.conf
├── install-base-packages.yml
├── package_nfs-client.yml
├── pip_conf_for_devpi.yml
├── post-install
│  ├── post-install-base-conf.yml
│  └── post-install-partitioning-datadrives.yml
├── script-automatic-update.yml
└── user_mgmt
    ├── adduser_self.sh
    ├── adduser.yml
    └── user.txt
```

- **clientconfs** → Folder contains preconfigured *Debian* configuration files which get copied to the clients by some playbooks. (For description and meaning of the config-files, s. A System Configuration (Files, Location) and further Services.

- **install-base-packages.yml** → Playbook with a list of packages of the *Debian* repository to be installed on the clients. This list can be extended with new packages or removed from ones not needed any more. Afterwards the playbook has to be executed.

- **package_nfs-client.yml** → Installs the *NFS*-client for access to the *NFS* network drives on the clients and adds mount-entries that they are automatically mounted.

- **pip_conf_for_devpi.yml** → Copies the config-file 'pip.conf' for python package download to the clients.

- **post-install** → This playbooks have to be run once after a new automated OS (re)install to finish a basic system configuration. (Because of the idempotency, if a new OS gets installed the playbooks can be run for all devices or only on that one where the new OS installation has been performed.)

  - **post-install-base-conf.yml** → Copies the config-file 'bash.bashrc' to the clients.

  - **post-install-partitioning-datadrives.yml** → Creates a filesystem on the *HDD* data disks and adds mount-entries that they are automatically mounted.

- **script-automatic-update.yml** → Playbook to perform a *Debian* package update, upgrade and `dist-upgrade`. Uses 'apt-get'. (Hint: Use wisely, test before on one client. Not that a `dist-upgrade` corrupts the system.)

- **user_mgmt** → s. 5.4 User Account Management

5.4 User Account Management

Ansible is as well used for the user account management so far, s. https://docs.ansible.com/ansible/latest/modules/user_module.html for options of the user module and https://www.mydailytutorials.com/working-ansible-user-module/ for examples.

Till now there is no need for a complex user account management and or an integration of the cluster to an existing one planned. Due to this, the user account management is kept simple, existent out of two interdependent scripts, a bash-script and a co-dependant *YAML*-playbook.

Because of, that the master/ti-hadoop server itself is not included as a client to the management software, its necessary to create a new user account first locally on the master/ti-hadoop server and then distribute that user account to the clusters clients. Till now, user accounts are created on all cluster clients nnode1-3, cnode1-12 and bnode1 so far, s. playbook `adduser.yml`.

The directory/file structure is as follows:

- **user_mgmt** → Folder containing user account management scripts.
  - **adduser_self.sh** → Self written interactive bash-script to create a local user account on the master/ti-hadoop server. With the given user credentials, the local user account will be created by using the standard 'adduser' command. Afterwards the user credentials are automatically appended in a predefined 'user section' to the `user.txt` file. This 'user section' can be directly used in the `adduser.yml` playbook.
  - **adduser.yml** → This playbook is the heart and the final step of the user account creation process. It contains the 'user sections' of all the user accounts created so far and needs to be extended if a new user account is created. Every 'user section' comprises a task which creates the user account on the clients and a task which creates a private user folder on bnode1 for the *NFS* network drives.
  - **user.txt** → Contains the predefined 'user sections' for the `adduser.yml` playbook. It gets extended by a new 'user section', if a new local user account on the master/ti-hadoop server is created with the `adduser_self.sh` bash-script. This is only an intermediate step to prevent mistakes, to keep overview and for history/changelog purpose. Therefore every 'user section' gets an additional timestamp of its local user account creation on the master/ti-hadoop server. (E.g. can changes to 'user sections' in the playbook – e.g. deactivation of a user account – later be compared with the original 'user section' of the txt-file.)

Because of the *NFS* network drives and correct folder/file permissions, its **very important** to keep the specific *UID* number (User Identification) and primary *GID* number (Group Identification) of every user identical and synchronized at all clients. Because of that reason, if a new local user account is created with the `adduser_self.sh` script on the master/ti-hadoop server, the *UID* of that user account gets copied to the according 'user section' of the `adduser.yml` playbook to ensure it has the same *UID* on the cluster clients.

Important (keep in mind): A mess in the *UID* numbers between the master/ti-hadoop server and the cluster clients should be always prevented. Thus it can be very difficult to solve it again. As well no one wants that user X gets maybe file permissions of user Y on the network drives, cause user X has somehow the same *UID* as user Y. So think about or test what could happen, if you delete e.g. a user on the master/ti-hadoop server but not on the cluster clients and following you create a new user on the master/ti-hadoop server but not on the cluster clients and so on.

(With no guarantee: The *Debian* standard behaviour assigns ascending *UID* numbers from 1000 on to new user accounts. It should be like, that if there exist users with the *UID*s from 1001 to 1007 on the system and you delete e.g. the users with *UID* 1002 and 1007, a new added user gets *UID* 1008 independent of the newly available *UID*s 1002 and 1007.)

Note: For every created user an identical primary group (same name) gets created on the systems. The users primary group is the standard group, which gets always assigned automatically to new created directories/files (standard *Debian* behaviour). With ‘groups’ later on are secondary groups meant.

How to create a new user account:

1. Run the `adduser_self.sh` bash-script as ‘distributor’ with sudo or as root. Together with the user fill out the user credentials. Then the local user account is created on the master/ti-hadoop server and the predefined ‘user section’ is appended to the `user.txt` file. Check the section for correctness.

2. Copy the appended ‘user section’ from the txt-file to the end of the playbook `adduser.yml` manually (without the timestamp) or e.g. with `tail -n 46 user.txt >> adduser.yml` (timestamp already excluded). The ‘tail’ command works only, if one user account is created or if strictly one after another is created following this procedure. But it would be as well possible to create e.g. five user accounts in a row by calling five times the `adduser_self.sh` bash-script and then copy manually the last five ‘user sections’ from `user.txt` to `adduser.yml` and remove the timestamps.

3. If everything looks correct and the playbook has been extended with the new user account(s), run the playbook as ‘distributor’ by executing `ansible-playbook adduser.yml –ask-become-pass`.

How to delete a user:

So far this hasn't been the case. There are two options (maybe more):

- If a user account should be finally removed from the whole cluster. Delete the user account on the master/ti-hadoop server by `deluser [--remove-home] <USER>`, if wanted with its home directory. Change the options 'state' and if you want to remove the home directory 'remove' of the according 'user section' in the `adduser.yml` playbook to '`state: absent`' and '`remove: yes`'. Then run the playbook. (Afterwards the 'user section' could be removed from the playbook, but maybe its wise to keep it.)
- If you think a user account is only 'temporarily' not needed anymore, but could be needed later on, I would suggest not to delete this user account, rather just set it inactive on the master/ti-hadoop server. This can be done e.g. by expiring the user account with `chage -E0 <USER>`, for other possibilities see for example https://www.thegeekdiary.com/unix-linux-how-to-lock-or-disable-an-user-account/.

How to manage existing users:

So far this hasn't been the case. The concept is as follows:

If an existing user account should be changed, e.g. new password, group membership, first change the options of the user account on the master/ti-hadoop server manually with standard Debian commands to the desired state. Afterwards search the appropriate 'user section' in the playbook `adduser.yml`, change the identical options inside the 'user section' (maybe extend the 'user section' by further options if necessary, s. https://docs.ansible.com/ansible/latest/modules/user_module.html) and call the playbook to distribute the changes to the cluster clients.

Two examples:

- Password change: If its desired to change the password of a user account, change it on the master/ti-hadoop server with `passwd <USER>` (as the user or su/sudo), copy the new password-hash from `/etc/shadow` to the according 'user section' in `adduser.yml` and replace the old one. If you call now the playbook, the password on all clients should be changed to the new one.
- Group membership to already existent groups: Add the user account to the desired and already existent group with e.g. `usermod -a -G <GROUP1,GROUP2,GROUP3> <USER>` on the master/ti-hadoop server. Search for the according 'user section' in `adduser.yml` and add the identical groups to the option 'groups'. After running the playbook, the user account should be a member of the existing groups on the clients. (Hint: Inside the playbook, check how the option 'append: yes/no' affects the group membership.)

5.5 Group Management

So far it hasn’t been necessary to establish groups. If groups should be needed one day, I definitely recommend to keep the user account and group creation process separate. That means to not create groups or join a user to existing groups in one step during the user creation process. Create first the groups and join afterwards the users in a separate step.

Important: The same as for the *UID* number counts for the unique *GID* number of a group. Identical groups have to have the same *GID* number on the master/ti-hadoop server as on the cluster clients.

On the master/ti-hadoop server its quite simple to create a group with e.g. `groupadd <GROUP>`. Then join the desired user(s) to the group(s) with e.g. `usermod -a -G <GROUP> <USER>`.

I recommend to create a playbook for the group creation on the cluster clients similar to the `adduser.yml` playbook. For the *Ansible* group-module s. https://docs.ansible.com/ansible/latest/modules/group_module.html.

As on the master/ti-hadoop server, create the similar group(s) with a playbook on the cluster clients. With the option ‘*GID*’ of the *Ansible* group-module, it can be ensured that the *GID* on the cluster clients gets the same as on the master/ti-hadoop server.

After the group(s) is/are established on the master/ti-hadoop server as well as on the cluster clients, go back to the `adduser.yml` playbook and add the group(s) to the option ‘groups’ of the according user section(s).

6 Fileserver – Network Drives and Backup

The Backup-Node ‘bnode1’ is the clusters central file/storage server for data. The server has a disk capacity of 32 *HDD*s á 4TB. These are connected to a hardware *RAID*-Controller and combined to one huge drive of around 120TB within a *RAID*6 array (max. 2 *HDD*s are allowed to fail without data loss). This huge drive is formatted with a high-performance *XFS* filesystem and is available over the network by *NFS* as network drives.

6.1 Network Drives

Network drive for user data (available at all clients, except bnode1 of course):

Hint: Use the network drive to store and share files between the cluster clients and the master/ti-hadoop server. It’s probably wise to not use it for calculations with heavy read-write operations. Therefore use the local *HDD* data disks.

The directory structure and permission is as follows:

```
├── mnt                 drwxr-xr-x  4 root root
  ├── fileserver        drwxr-xr-x  5 root root
  | ├── public          drwxrwxrwt 31 root root
  | ├── users           drwxr-xr-x  9 root root
  | └── groups          drwxr-xr-x  2 root root
  ├ master
    ├──...
```

The directory `fileserver` is thereby the mount and entry point of the *NFS* network drive. Whereas the directories `public`, `users`, `groups` are just ordinary subdirectories. As it can be seen by the directory permissions, only root is allowed to create files and folders inside the directories `fileserver`, `users`, `groups`.

The concept of the predefined file/directory structure and permissions are as follows. But in the end every user is itself responsible to set the correct permissions to the files/directories created inside the following directories. And of course the user itself can change the permissions of owned files/directories if he wants to.

- **public** → Public directory for all users to share data among. For deletion protection of files/directories by other users, the permission ‘sticky bit (t)’ is set to the `public` directory. That means, even if all users have write access, only the owner can delete the files/directories (similar to `/tmp`).

- **users** → Directory contains for every user an identical named directory for private user data (only the user has permissions), e.g. `drwx------ 3 tobi tobi 32 Apr 18 2019 tobi`. These user directories are established during the user creation process, s. `adduser.yml` playbook.

- **groups** → Directory contains for every group an identical named directory for private group data, where just members of the group and the directory owner should have permissions. (So far there don’t exist any group directories, because it wasn’t necessary yet.)

Network drive for system data (available only to the master/ti-hadoop server):

The directory structure and permission is as follows:

```
├── mnt                 drwxr-xr-x  4 root root
  ├── fileserver        drwxr-xr-x  5 root root
  | ├──...
  ├── master            drwxr-xr-x  5 root root
    ├── backup          drwxr-x--- 2 root root
    ├── devpi_cache     drwxr-xr-x 5 devpi devpi
    └── repository      drwxr-xr-x 5 apt-mirror apt-mirror
```

The directory `master` is thereby the mount and entry point of the *NFS* network drive. Whereas the directories `backup`, `devpi_cache`, `repository` are just ordinary subdirectories. As it can be seen by the directory permissions, only root is allowed to create files and folders inside the directories `master` and `backup`. Whereas the directories `devpi_cache` and `repository` belong the system users devpi and apt-mirror.

- **backup** → Directory for backups (only writeable by root).
- **devpi_cache** → Directory used to cache *Python* packages from the official *PyPi* (*Python Package Index*) repository by the *Devpi* service (s. 7 Package Management (Debian and Python)).
- **repository** → Directory used for a local *Debian* package mirror, which is synced with an official *Debian* repository.

6.2 Backup

So far, there is only a very rudimentary backup strategy implemented. More or less, only system configuration files of the master/ti-hadoop server are backed up, but none user files or client data. Till now this is done only manually by the administrator after he changed or added new system configurations. In future it should be thought about doing regular backups automatically, e.g. implementing a cronjob, or implementing a ‘real’ backup strategy with other tools.

The command and files for backup on the master/ti-hadoop server are right now:

```
tar -zcvf /mnt/master/backup/YYYY_MM_DD_etc_srv_home_distributor_backup.tar.gz
/etc/ /srv/ /home/distributor/ /var/www/html/clientconfs/
```

Additionally a copy of the `/etc/exports` file on ‘bnode1’ (*NFS* exports) to `/mnt/master/backup/YYYY_MM_DD_nfs_export_bnode1` should be done if changed.

If required, set the permission and ownership of the files inside `/mnt/master/backup/` to: `-rw-r----- 1 root root ...` (e.g. `chmod 640 <FILE>`).

Important: Keep in mind, that a backup to the Backup-Node ‘bnode1’ is due to the *RAID*6 array relatively safe for data loss (two disks can fail). But its not safe of data/(file-)system corruption or user failures.

7 Package Management (Debian and Python)

Because of the fact, that there is no internet connection from inside the clusters private network, a local repository (mirror) for *Debian* packages and a package cacher (*Devpi*) for *Python* packages from the official *PyPi* repository is established on the master/ti-hadoop server.

7.1 Debian Packages

A local *Debian* repository is available to the cluster clients by a running *Apache* web-server. The repository is a local mirror of an official *Debian* repository and regularly automatically synchronized by the package `apt-mirror`.

Further is nothing special to consider. On the cluster clients, *Debian* packages can be installed in the same way as usual. The only difference is, that the package repository is the master/ti-hadoop server.

So it would be possible, if users need dedicated computing nodes and have different requirements, to install different Debian packages on different clients. But its recommended to keep the packages synchronized to all cluster clients, s. 5.3 Client Management. In the end its the consideration and decision of the administrator, dependant of the wanted package/software and if a package is relevant to all clients/users or e.g. only needed on one Computing-Node, with e.g. a new automated OS reinstall in the end.

7.2 Python Packages

Python packages are made avlailable to the cluster clients by *Devpi*, used as a local *PyPi* caching mirror (http://pypi.org/). The software is a powerful *PyPI*-compatible server and a complementary command line tool to drive packaging, testing and release activities with *Python*. So it can be used for own *Python* package development. That means uploading, testing and staging packages in private indexes with different users and an integrated user management. Even a web interface and a search plugin can be installed, to navigate and search through private indexes. https://devpi.net/docs/devpi/devpi/stable/%2Bd/index.html

Concept for Python packages

In general its recommended for users to work with their own virtual *Python* environments (*venv*) created on the Computing-Nodes, e.g. one for every dedicated project. (Should be possible to copy a once created *venv* to different servers.)

If a wanted *Python* package is available in the *Debian* repository, the *Python* package is installed system-wide on all cluster clients with the `install-base-packages.yml` playbook, s. 5.3 Client Management.

If its not available, the user has to create and work with his own virtual *Python* environment. Then the user can install and use desired *Python* packages available on *PyPi*.

How to create a virtual Python environment and install packages

1. Create a virtual environment to work with (creates the *venv* directory at current path, otherwise specify a full path)

   ```
   python3.5 -m venv [</PATH/TO/>]<VENVNAME>
   ```

2. Activate virtual environment

   ```
   source [</PATH/TO/>]<VENVNAME>/bin/activate
   ```

3. PIP install package,, search packages

   - install package(s) from *PyPi (or rather from the local cache)*

   ```
   pip3.5 install <PACKAGE>
   ```

   - install packages according to a requirements.txt file

   ```
   pip3.5 install -r [/PATH/TO/]<REQUIREMENTS>.txt
   ```

   - search packages

   ```
   pip3.5 search <PACKAGE>
   ```

4. Work with *Python* in virtual environment

   ```
   python3.5 (-i)
   python3.5 (-i) [/PATH/TO/] <YOURSCRIPT>.py
   ```

   (-i : inspect interactively after running a script; forces a prompt)

5. Deactivate virtual environment

   ```
   deactivate
   ```

Notes:

If you use screen to run python scripts, don’t forget to activate the environment after running screen command

- `screen` (creates session) → now activate the virtual environment
- `screen -r <PID>` (resume to session. if multiple sessions exist, find out first to resume)
- `screen -ls` (find the running sessions and its)
- inside session type `Strg + A + D` (detaches the running session)
- inside session type `Strg + A + \` (kills and terminates the running session)

8 Hadoop (Disbribution Hortonworks)

Now its time to start with the installation of *Hadoop*. It was decided to choose the *Hadoop* distribution of *Hortonworks*, which is open source (only support costs). The reason is, its supposed to be very difficult and complex, to keep all the different components in compatible software versions together, in a way that they function well. Cause its a whole distribution, in that case *Hortonworks* cares about the interoperability of the components different software versions together.)

https://stackoverflow.com/questions/44955010/hortonworks-vs-apache-projects

https://tecadmin.net/set-up-hadoop-multi-node-cluster-on-centos-redhat/

https://www.tutorialspoint.com/de/hadoop/hadoop_multi_node_cluster.htm

https://www.dummies.com/programming/big-data/hadoop/master-nodes-in-hadoop-clusters/

https://www.bigdata-insider.de/was-ist-hortonworks-a-623004/

https://blog.codecentric.de/2013/08/einfuhrung-in-hadoop-was-ist-big-data-hadoop-teil-1-von-5/

9 Appendix

Table of Contents

A System Configuration (Files, Location) and further Services

Usually copies of the original configuration files has been created, so the changes to the default can be shown, e.g. by the difference via `diff <FILE1> <FILE2>`.

A.1 Basic Linux Settings

- **bash** [playbook: ‘post-install-base-conf.yml’]

  `/etc/bash.bashrc` → bash completion in interactive shells

  `/root/.bashrc` → colorized `ls` output in interactive shells as user root

- **file system table (fstab)**

  `/etc/fstab` → mounting system partitions and network drives

- **grub** [preseed.cfg]

  `/var/www/html/clientconfs/grub` → ‘grub’ gets downloaded during automated *Debian* installation from http://master.hadoop.cluster/clientconfs/grub and copied to `/etc/default/grub`; option ‘GRUB_DISABLE_LINUX_UUID’ is set to true, because of problems with UUID recognition of Fake-/BIOS-RAID

- **keyboard** [preseed.cfg]

  `/var/www/html/clientconfs/keyboard` → ‘keyboard’ gets downloaded during automated *Debian* installation from http://master.hadoop.cluster/clientconfs/keyboard and copied to `/etc/default/keyboard`; to obtain a functioning German keyboard layout

- **ssh and sshd**

  `/etc/ssh/ssh_config` → *SSH* client conf

  `/etc/ssh/sshd_config` → *SSH* daemon, server/service conf (especial configuration on master/ti-hadoop server)

  https://www.techgrube.de/tutorials/ssh-login-mit-public-private-key-authentifizierung

- **network time protocol (ntp)** [preseed.cfg]

  `/var/www/html/clientconfs/timesyncd` → ‘timesyncd.conf’ gets downloaded during automated *Debian* installation from http://master.hadoop.cluster/clientconfs/timesyncd and copied to `/etc/systemd/timesyncd.conf`

  `/etc/systemd/timesyncd.conf` → systemd-timesyncd (NTP-client) config to use local server ‘ntp.hadoop.cluster’; automatically configured on the custer clients during the automated *Debian* installation

  `timedatectl status` → request status of timesyncd

  https://wiki.ubuntuusers.de/systemd/timesyncd/

  https://feeding.cloud.geek.nz/posts/time-synchronization-with-ntp-and-systemd/

A.2 All Clients (nnode’s, cnode’s, bnode)

- **client and user management (ansible)** [preseed.cfg]

  user ‘ansible’ → ‘distributor’s’ opponent user on the cluster clients

  `/home/ansible/.ssh` → for password-less ssh authentication of user ‘distributor’ as user ‘ansible@...’ on the clients, ssh-keys are established; the user and its public-key are automatically established on the clients during the automated *Debian* installation

- **debian packages (apt-get)** [preseed.cfg, playbook: ‘post-install-base-conf.yml’]

  `/etc/apt/sources.list` → self-defined repository entries, use the local *Debian* mirror on the master/ti-hadoop server as standard repository on the cluster clients; automatically configured on the clients during the automated *Debian* installation and a playbook

- **python packages (pip)** [playbook: ‘pip_conf_for_devpi.yml’]

  `/etc/pip.conf` → local private repository (master.hadoop.cluster) for *Python* packages used by pip

- **network drives (nfs-common; not on bnode1!)** [playbook: ‘package_nfs-client.yml’]

  nfs-common → *NFS*-client

  `/etc/fstab` → mounting *NFS* network drives with ‘x-systemd.automount’

  `systemctl daemon reload` and `systemctl restart remote-fs.target` → commands to run once, to activate ‘x-systemd.automount’

A.3 Name-Node (nnode1 – 3)

A.4 Computing-Node (cnode1 – 12)

A.5 Backup-Node (bnode1)

- **data storage**

  `/dev/sdb1` → data storage partition (HW-RAID6 array) with *XFS* filesystem

  `/backup/` → mount point of `/dev/sdb1`

- **network drives (nfs-kernel-server, nfsv4)**

  NFSv4 exports exist in a single 'pseudo filesystem', where the real directories are mounted with the '--bind' option.

  NFSv4 no longer has a separate 'mount' protocol. Instead of exporting a number of distinct exports, a *NFSv4*-client sees the *NFSv4* servers exports as existing inside a single filesystem, called the *NFSv4* 'pseudo filesystem'. On the current linux implementation, the 'pseudo filesystem' is a single real filesystem, identified at export with the 'fsid=0' option.

  `/backup/nfs/` → real filesystem for the 'fileserver' export

  `/backup/nfs_master/` → real filesystem for the 'master' export

  `/etc/fstab` → bind mount entries, to not export directly the real filesystem, instead create a 'pseudo filesystem' structure for *NFS*-exports

  `/backup/nfs /srv/nfs/fileserver` none bind 0 0

  `/backup/nfs_master /srv/nfs/master` none bind 0 0

  `/srv/nfs/` → exported root directory (fsid=0) of pseudo filesystem, content of that directory is readable to all *NFS*-clients in the local *LAN*

  `/srv/nfs/fileserver` → exported read-writeable directory (pseudo filesystem), available to all cluster clients

  `/srv/nfs/master` → exported read-writeable directory (pseudo filesystem), only available to the master/ti-hadoop server

  `/etc/default/nfs-kernel-server` → conf for *NFS*-server; *NFSv2*/*v3* disabled to use only *NFSv4*

  `/etc/exports` → access control list for filesystems which are exported to remote hosts (*NFS*-clients); definition of paths and export options; the service has to be reloaded `systemctl reload nfs-kerne-server` after changes

  `exportfs -s` → display current export list; in general a command to maintain table of exported *NFS* file systems, normally the master export table is initialized with the contents of `/etc/exports` `systemctl status nfs-kernel-server` or `service nfs-kernel-server status` ⟶ check status of ‘nfs-kernel-server’ service (e.g. start, stop, restart, reload)

https://wiki.debianforum.de/NFSv4-Mini-Howto

Hint: Its said that *NFSv4* is supposed to have a new innovations. One of them should be, that its not mandatory that users and groups need to have identical numerical *UID*s and *GID*s on the server and the clients. As long as they belong to the same domain, an automatic translation takes place.

It has been tried to configure it like that. All devices belong to the same domain, but it was not possible to realize it. Instead an identical user with different UIDs on the devices led to malfunction of automatic translations. Conclusion was, that it seems like, that its only possible, if *NFSv4* is secured with ‘Kerberos’. Therefore, users and groups need to have identical numerical *UID*s and *GID*s on the server and the clients!

https://wiki.debianforum.de/NFSv4-Mini-Howto

https://serverfault.com/questions/812813/nfsv4-user-mapping

A.6 Master / TI-Hadoop

- **network interfaces**

  `/etc/network/interfaces` → configuration of the two network interfaces, external public *HAW* and internal private *LAN*

  https://askubuntu.com/questions/824376/failed-to-start-raise-network-interfaces-after-upgrading-to-16-04

  https://unix.stackexchange.com/questions/390307/startup-debian-9-error-failed-to-start-raise-network-interfaces

  https://debianforum.de/forum/viewtopic.php?t=169838

- **network drives (nfs-common)**

  nfs-common → *NFS*-client

  `/etc/fstab` → mounting *NFS* network drives with ‘x-systemd.automount’

  `systemctl daemon reload` and `systemctl restart remote-fs.target` → commands to run once, to activate ‘x-systemd.automount’

- **network time protocol server/service (ntpd)**

  ntp → *NTP* server daemon and utility programs

  user ‘ntp’ → dedicated system user for the service, automatically created

  `/etc/ntp.conf` → external *NTP* server (ntp.etech.haw-hamburg.de) to obtain the current time from as further configurations, e.g. access restriction

  `systemctl enable ntp.service` → enable *NTP* as a systemd service

  `systemctl status ntp.service` or `service ntp.service status` → check if *NTP* daemon is running (e.g. start, stop, restart, reload)

  `ntpq -p` → print a list of the configured *NTP* peers

  `hwclock --systohc` → writes the current system time to the hardware clock

  https://community.hetzner.com/tutorials/install-and-configure-ntp/de?title=Uhrzeit_synchronisieren_mit_NTP

  https://linuxconfig.org/how-to-setup-ntp-server-and-client-on-debian-9-stretch-linux

- **local debian repository mirror (apt-mirror)**

  apt-mirror → tool to mirror apt sources

  user ‘apt-mirror’ → dedicated system user for the service, automatically created

  `id apt-mirror` → `uid=117(apt-mirror) gid=121(apt-mirror) groups=121(apt-mirror)`

  `/etc/apt/mirror.list` → external official repository, base path of local mirror,...

  `/mnt/master/repository/` → base path for local repository mirror, with the permissions `drwxr-xr-x 5 apt-mirror apt-mirror 59 Apr 3 2019 repository`

  `apt-mirror` → command to start synchronization of the local mirror with an official repository

  `/etc/cron.d/apt-mirror` → cronjob for daily automatic sync of the local mirror

  That the cluster clients can use the local mirror, it has to be accessible to the cluster clients via *HTTP*, s. web-server (apache2).

  https://www.anleitungen.rrze.fau.de/betriebssysteme/linux/ppa-spiegeln-mit-apt-mirror/

  https://www.tecmint.com/setup-local-repositories-in-ubuntu/

  https://wiki.ubuntuusers.de/apt-mirror/

  https://www.howtoforge.de/anleitung/wie-man-einen-lokalen-debianubuntu-spiegel-mit-apt-mirror-erstellt/

  https://www.tobanet.de/dokuwiki/debian:debmirror

- **dhcp | dns | tftp (dnsmasq)**

dnsmasq → a lightweight, easy to configure *DNS* and *DHCP* server with *BOOTP*/*TFTP*/*PXE* functionality

user ‘dnsmasq’ → dedicated system user for the service, automatically created

`/etc/dnsmasq.conf` → main conf with *DNS*, *DHCP*, *TFTP*, *PXE* sections; *DHCP* is static with infinite lease time, that means every MAC-address is a dedicated IP-address assigned

`/etc/hosts` → names are resolved inside the private *LAN* according to that file

`systemctl status dnsmasq` or `service dnsmasq status` → check status of ‘dnmasq’ service (e.g. start, stop, restart, reload)

resolvconf → dnsmasq is configured in that way, that it uses the ‘resolvconf’ package, to forward unknown *DNS*-requests to external *DNS*-servers and cache them

`/etc/network/interfaces` → `dns-*` options are implemented by ‘dnsmasq’ `dns-nameservers 141.22.13.12 141.22.13.15` and `dns-search etech.haw-hamburg.de`

`/etc/resolvconf/resolv.conf.d/` → conf files of the ‘resolvconf’ package

`/etc/resolvconf/resolv.conf.d/original` → Copy of the `/etc/resolv.conf` file before the ‘resolvconf’ package was installed. This file has no effect on the functioning of ‘resolvconf’; it is retained so that `/etc/resolv.conf` can be restored to its original state if the ‘resolvconf’ package is removed.

`/etc/resolv.conf` → symlink to `/etc/resolvconf/run/resolv.conf`, dynamic file generated by ‘resolvconf’ package, contains `nameserver 127.0.0.1` and `search etech.haw-hamburg.de`

`/var/run/dnsmasq/resolv.conf` → location of the ‘resolv.conf’ used by ‘dnsmasq’ (if package ‘resolvconf’ is installed) with the “real” external *DNS*-server `nameserver 141.22.13.12` and `nameserver 141.22.13.15`

tftp → trivial file transfer protocol, used for the automated *Debian* network installation with *PXE*

`/srv/tftp/` → base path for *TFTP* with permissions `drwxrwxr-x 3 dnsmasq root 4096 Jan 21 15:39 tftp` and all files for the *PXE* boot and the *Debian* network installation

http://manpages.ubuntu.com/manpages/trusty/man8/resolvconf.8.html

https://wiki.ubuntuusers.de/Dnsmasq/

- **automated debian installation (pxe boot)**

  `/srv/tftp` → base path for *TFTP*

  Download the *Debian* network installer from https://www.debian.org/distrib/netinst#netboot and unpack ‘netboot.tar.gz’ to `/srv/tftp`.

  `chown -R dnsmasq:root /srv/tftp/` → files/directories belong user ‘dnsmasq’

  `/srv/tftp/preseed.cfg` → answer file for the installer; includes the predefined ‘answers’ for the automated *Debian* installation; some configuration files are downloaded by ‘wget’ from http://master.hadoop.cluster/clientconfs/ (s. web-server) [Hint: Difficult is to detect the device numbers of the *Fake-/BIOS-RAID*]

  `/srv/tftp/debian-installer/amd64/ boot-screens/syslinux.cfg` → pxelinux conf, e.g. timeout 300, ontimeout bootlocal

  `/srv/tftp/debian-installer/amd64/boot-screens/txt.cfg` → self added and defined ‘automatic’ menu entry ‘Automatic Install Debian 9.5 (Stretch)’ as install option (others are ‘Install’ or ‘Boot from local disk...’)

  https://wiki.debian.org/PXEBootInstall?action=show&redirect=DebianInstaller%2FNetbootPXE

  http://www.thurnhofer.net/pxe-server-unter-linux-installieren/

  https://wiki.debian.org/DebianInstaller/SataRaid

  https://salsa.debian.org/installer-team/netboot-assistant/tree/master/config

  https://www.debian.org/releases/stretch/s390x/apb.html.de

  https://www.syslinux.org/wiki/index.php?title=SYSLINUX#LOCALBOOT_type

  https://debiananwenderhandbuch.de/fai.html

- **web-server (apache2)**

  user ‘www-data’ ⟶ dedicated system user for the web-server, automatically created

  `/etc/apache2/apache2.conf` ⟶ main configuration file; contains settings that are global to *Apache2*

  `/etc/apache2/ports.conf` ⟶ houses the directives that determine which *TCP* ports *Apache2* is listening on

  `/var/www/html/` ⟶ base path for websites (specified ‘DocumentRoot’ in `/etc/apache2/sites-available/000-default.conf`); permissions should always be set, that all accessible directories and files inside that path belong to user and group ‘www-data’; then only the owner ‘www-data’ gets `rw-` for files and `rwx` for directories, the group and others get only `r` for files and `r-x` for directories, e.g. like `drwxr-xr-x 3 www-data www-data 4096 Jan 21 18:49 html`.

  `/var/www/html/index.html` ⟶ *HTML*-file or shown website if http://master.hadoop.cluster is requested

  `/etc/apache2/conf-available/` ⟶ contains available configuration files

  `/etc/apache2/conf-enabled/` ⟶ holds symlinks to the files in `/etc/apache2/conf-available`; when a configuration file is symlinked, it will be enabled the next time *Apache2* is restarted

  `a2enconf <CONF>` ⟶ creates symlink in `/etc/apache2/conf-enabled`

  `a2disconf <CONF>` ⟶ removes symlink in `/etc/apache2/conf-enabled`

  `/etc/apache2/mods-available/` ⟶ configuration files to both load modules and configure them; not all modules will have specific configuration files, however

  `/etc/apache2/mods-enabled/` ⟶ holds symlinks to the files in `/etc/apache2/mods-available`; when a module configuration file is symlinked, it will be enabled the next time *Apache2* is restarted

  `a2enmod <MODULE>` ⟶ creates symlink in `/etc/apache2/mods-enabled`

  `a2dismod <MODULE>` ⟶ removes symlink in `/etc/apache2/mods-enabled`

  `/etc/apache2/sites-available/` ⟶ configuration files for *Apache2* ‘VirtualHosts’; ‘VirtualHosts’ allow *Apache2* to be configured for multiple sites that have separate configurations

  `/etc/apache2/sites-enabled/` ⟶ like mods-enabled, sites-enabled contains symlinks to the `/etc/apache2/sites-available` directory; similarly when a configuration file in sites-available is symlinked, the site configured by it will be active once *Apache2* is restarted

`a2enconf <SITE>` → creates symlink in `/etc/apache2/sites-enabled`

`a2disconf <SITE>` → removes symlink in `/etc/apache2/sites-enabled`

`systemctl status apache2` or `service apache2 status` → check status of 'apache2' service (e.g. start, stop, restart, reload)

https://help.ubuntu.com/lts/serverguide/httpd.html

http access for automated Debian installation:

`/var/www/html/clientconfs/` → predefined configuration files used during the automated *Debian* installation; accessed by 'wget' via *HTTP* from http://master.hadoop.cluster/clientconfs

http access for local Debian repository mirror:

```
ln -s /mnt/master/repository/mirror/ftp.de.debian.org/
/var/www/ftp.hadoop.cluster
```
→ created symlink to access the local *Debian* repository via *HTTP*

```
ln -s /mnt/master/repository/mirror/security.debian.org/
/var/www/security.hadoop.cluster
```
→ created symlink to access the local *Debian* repository via *HTTP*

`/etc/apache2/sites-available/mirror-ftp.conf` → 'VirtualHost' with 'ServerName' ftp.hadoop.cluster and 'DocumentRoot' `/var/www/ftp.hadoop.cluster`

`/etc/apache2/sites-available/mirror-security.conf` → 'VirtualHost' with 'ServerName' security.hadoop.cluster and 'DocumentRoot' `/var/www/security.hadoop.cluster`

- **uncomplicated firewall (ufw)**

  `/etc/ufw/ufw.conf` → main conf

  `ufw status verbose` → the firewall is configured to block by default incoming traffic and allow outgoing traffic; on ethernet controller ‘eno1’ (public *HAW* network) only the *TCP*-port 50222 for *SSH* is allowed for incoming traffic; on ‘eno2’ (private *LAN*) all ports are opened for incoming traffic, additionally *UDP*-port 67 is opened for *DHCP*-requests from *UDP*-port 68

Status: active		
Logging: on (low)		
Default: deny (incoming), allow (outgoing), disabled (routed)		
To	**Action**	**From**
Anywhere on eno2	ALLOW IN	172.23.13.0/24
50222/tcp on eno1	ALLOW IN	141.22.0.0/16
67/udp on eno2	ALLOW IN	68/udp

  `ufw enable/disable` → activate or deactivate the ‘ufw’ service

  `systemctl status ufw` or `service ufw status` → check status of ‘ufw’ service (e.g. start, stop, restart, reload)

  https://wiki.ubuntuusers.de/ufw/

  https://www.cyberciti.biz/faq/ufw-allow-incoming-ssh-connections-from-a-specific-ip-address-subnet-on-ubuntu-debian/

  https://www.linode.com/docs/security/firewalls/configure-firewall-with-ufw/

- **fail2ban**

  Fail2ban scans log files and bans IPs that show malicious signs, like too many password failures, seeking for exploits, etc. (e.g. DoS/DDoS attacks). Generally *Fail2Ban* is then used to update firewall rules to reject the IP addresses for a specified amount of time.

  There exist default ‘*.conf’ configuration files, which are created during installation and could be overwritten by updates.

  `/etc/fail2ban/fail2ban.conf`

  `/etc/fail2ban/jail.conf`

  `/etc/fail2ban/action.d/*conf`

  `/etc/fail2ban/filter.d/*.conf`

Thus, changes should only be done in corresponding '*.local' configuration files. '*.local' files and their options have priority over the '*.conf' files. Thus, only options which differ from the default, are necessary in the '*.local' files. The remaining default options are then taken from 'jail.conf'.

`/etc/fail2ban/fail2ban.d/fail2ban.local` → basic configuration settings, usually the default is appropriate (that's why there exists so far no 'fail2ban.local')

`/etc/fail2ban/jail.d/jail.local` → combines for different services 'Actions' and 'Filters' to a 'Jail', e.g. *SSHd*, *Apache2*, etc.; here the monitored services and all individual settings which differ from 'jail.conf' can be set

`/etc/fail2ban/action.d/*.local` → files which contain the commands to block/unblock IP-addresses

`/etc/fail2ban/filter.d/*.local` → files which contain the regular expressions to analyse the according log-files

`fail2ban-client set <JAIL> unbanip <IP>` → command to unban a banned IP for dedicated service <JAIL>, e.g. ssh

`systemctl status fail2ban` or `service fail2ban status` → check status of 'fail2ban' service (e.g. start, stop, restart, reload)

https://wiki.ubuntuusers.de/fail2ban/

https://www.thomas-krenn.com/de/wiki/SSH_Login_unter_Debian_mit_fail2ban_absichern

- **client and user management (ansible)**

  `/etc/ansible/ansible.cfg` → main *Ansible* conf

  `/etc/ansible/hosts` → defined hosts included for orchestration/management

  user ‘distributor’ → special dedicated user to manage the clients

  `/home/distributor/` → directory for playbooks, config-files,…

  `/home/distributor/.ssh` → for password-less ssh authentication of user ‘distributor’ as user ‘ansible@...’ on the clients, ssh-keys are established

  https://wiki.ubuntuusers.de/Ansible/

  https://www.admin-magazin.de/Online-Artikel/Konfigurationsmanagement-mit-Ansible

  https://www.tutorialspoint.com/ansible/index.htm

  https://medium.com/@abhijeet.kamble619/10-things-you-should-start-using-in-your-ansible-playbook-808daff76b65

  https://serversforhackers.com/c/how-ansible-vault-works

  https://www.howtoforge.com/tutorial/setup-new-user-and-ssh-key-authentication-using-ansible/

  https://github.com/ansible/ansible-modules-core/issues/4756

  http://www.mydailytutorials.com/the-basics-of-ansible-variables/

  https://docs.ansible.com/ansible/2.6/modules/list_of_system_modules.html

  https://docs.ansible.com/ansible/latest/modules/modules_by_category.html

  https://docs.ansible.com/ansible/devel/user_guide/playbooks_variables.html

  https://docs.ansible.com/ansible/devel/user_guide/intro_inventory.html

  https://docs.ansible.com/ansible/2.5/user_guide/playbooks_loops.html#looping-over-the-inventory

  https://docs.ansible.com/ansible/latest/modules/file_module.html

  https://docs.ansible.com/ansible/latest/modules/filesystem_module.html

  https://docs.ansible.com/ansible/latest/modules/parted_module.html#id5

  https://docs.ansible.com/ansible/latest/modules/apt_module.html#apt-module

  https://docs.ansible.com/ansible/latest/modules/copy_module.html#copy-module

  https://docs.ansible.com/ansible/latest/modules/apt_repository_module.html#apt-repository-module

  https://docs.ansible.com/ansible/latest/modules/mount_module.html#mount-module

  https://github.com/AutoLogicTechnology/autologic-users

- **private local pypi caching mirror for python packages (devpi)**

  Devpi is a *Python* based software, so it comes as a *Python* package. It has to be run in an own environment and with a dedicated system user ‘devpi’ (for security reasons).

  Inside the environment exist two main *Python* scripts, which are usable like *Debian* commands, to work with the *Devpi* server. ‘devpi-server’, to e.g. initialize, start, configure, etc. the devpi server/service. And ‘devpi’ (a *Devpi* client, command-line tool), to e.g. add users, change passwords, list projects, manage and use own python indexes and packages, upload, test and install own packages, etc.

  Preconfiguration:

  `adduser --system --home /srv/devpi/ --group devpi` → create a dedicated ‘devpi’ system user

  `id devpi` → `uid=122(devpi) gid=126(devpi) groups=126(devpi)`

  `mkdir /mnt/master/devpi_cache` → create directory for the mirror cached *PyPi Python* packages, on *NFS* network drive

  `chown devpi:devpi /mnt/master/devpi_cache/`

  `chmod 755 /mnt/master/devpi_cache`

  `ls -la /mnt/master/` → `drwxr-xr-x 5 devpi devpi 176 May 10 2019 devpi_cache`

  `su devpi -s /bin/bash` → switch to user ‘devpi’ with a shell (necessary cause its a system user)

  `mkdir /srv/devpi` → base directory for the *Devpi* software, with the permissions `drwxr-xr-x  4 devpi devpi 4096 Jan 18 20:54 devpi`

  `python3.5 -m venv /srv/devpi/devpi-venv` → create virtual *Python* environment

  `source /srv/devpi/devpi-venv/bin/activate` → use virtual *Python* environment

  Installation:

  `pip3.5 install devpi` → download and install the *Python* package ‘devpi’ in the virtual environment

  `cd /srv/devpi/devpi-venv/bin/` → change directory for the two main *Devpi* scripts

  ```
  devpi-server --host=master.hadoop.cluster --init --port=3141
  --restrict-modify root --passwd root --serverdir=/mnt/master/devpi_cache/
  ```
  → run once to initialize the *Devpi* server/service and set password for root

  ```
  devpi-server --host=master.hadoop.cluster –-gen-config --port=3141
  --serverdir=/mnt/master/devpi_cache/ --restrict-modify root
  ```
  → run once to generate predefined system configuration files (e.g. for ‘systemd’)

`su root` → switch to root

`cp /srv/devpi/gen-config/devpi.service /etc/systemd/system` → copy the created ‘devpi.service’ config to ‘systemd’ directory; change the user inside the service-config to ‘user=devpi’

`chown root:root /etc/systemd/system/devpi.service`

`chmod 644 /etc/systemd/system/devpi.service`

`systemctl enable devpi` → enable *Devpi* as a service for autostart and management

`systemctl start devpi` → start the *Devpi* server/service

Management:

Manage the *Devpi* server/service with systemd as root:

```
systemctl [start|stop|restart|reload|status] devpi.service
```

Manage the *Devpi* server as user ‘devpi’ (`su devpi -s /bin/bash`) and activated virtual *Python* environment (`source /srv/devpi/devpi-venv/bin/activate`) with the script ‘devpi-server’ `/srv/devpi/devpi-venv/bin/devpi-server`.

`devpi-server -h` → command help

`devpi-server --log --serverdir=/mnt/master/devpi_cache/` → show logfile content

```
devpi-server --serverdir=/mnt/master/devpi_cache/ --restrict-modify root
--[start | stop | status]
```
→ different way to start/stop the *Devpi* server/service

Hint: Where appropriate, the options ‘serverdir’ and ‘restrict-modify’ have to be specified if working with the ‘devpi-server’ command/script, e.g. to start the server.

Manage *Devpi* ‘content’ with the *Devpi* client (command-line tool). Activate the virtual *Python* environment (`source /srv/devpi/devpi-venv/bin/activate`) to use the client/script ‘devpi’ `/srv/devpi/devpi-venv/bin/devpi`. Following, some expamples:

`devpi -h` → command help

`devpi use http://master.hadoop.cluster:3141` → *Devpi* server to connect to

`devpi login root` → login to this *Devpi* server as user root

```
devpi user -c packages email=packaging@company.com password=packages
```
→ create a user ‘packages’ and set password

`devpi user -l` → list all existent users

how to use the local devpi server to download and install python packages:

```
pip3.5 search --i http://master.hadoop.cluster:3141/root/pypi/
--trusted-host master.hadoop.cluster <PACKAGE>
```
→ search for a *Python* package on *PyPi* over the local *Devpi* server

```
pip3.5 install --i http://master.hadoop.cluster:3141/root/pypi/+simple/
--trusted-host master.hadoop.cluster <PACKAGE>
```
→ install a *Python* package over/from the local *Devpi* server

This commands can be used on the cluster clients and the master/ti-hadoop server (alternatively, on the master/ti-hadoop server packages can be directly downloaded and installed from the *PyPi* repository, e.g. `pip3.5 install <PACKAGE>`).

For the cluster clients and the user, to not type that unhandy commands all the time and to make the local *Devpi* server the standard repository used for *Python* packages, a predefined ‘pip.conf’ is copied with *Ansible* to the cluster clients by ‘pip_conf_for_devpi.yml’ playbook. Then a `pip3.5 install <PACKAGE>` uses the local *Devpi* server on the cluster clients.

https://pseudoscripter.wordpress.com/2016/03/07/running-a-pypi-mirror-on-your-laptop/

https://docs.python.org/3/tutorial/venv.html

https://phabricator.wikimedia.org/T114871

https://devpi.net/docs/devpi/devpi/stable/+d/userman/devpi_commands.html#devpi-command-reference-server

https://stefan.sofa-rockers.org/2017/11/09/getting-started-with-devpi/

https://www.dabapps.com/blog/introduction-to-pip-and-virtualenv-python/

https://github.com/jhermann/devpi-enterprisey/tree/master/debianized-devpi

B Useful Linux Commands

https://ryanstutorials.net/linuxtutorial/cheatsheet.php

- **ssh**

```
ssh-keygen -f /home/<USER>/.ssh/known_hosts -R <HOST>
```

```
ssh-keyscan <HOST> >> /home/<USER>/.ssh/known_hosts
```

```
ssh -i .ssh/id_rsa <USER>@<HOST>
```

```
ssh-copy-id -i id_rsa.pub <USER>@<HOST>
```

- **scp**

```
scp -P <PORT> </FILE/TO/SEND> <USER>@<HOST>:</WHERE/TO/PUT>
```

```
scp </FILE/TO/SEND> <USER>@<HOST>:</WHERE/TO/PUT>
```

```
scp <USER>@<HOST>:</FILE/TO/SEND> </WHERE/TO/PUT>
```

http://www.hypexr.org/linux_scp_help.php

- **vim**

`Shift + Insert` $\longrightarrow$ to copy text into vim out of shell

https://ryanstutorials.net/linuxtutorial/cheatsheetvi.php

- **tar**

```
tar -zcvf <TAR-ARCHIVE>.tar.gz <SOURCE-DIRECTORY>
```

```
tar -zxvf <TAR-ARCHIVE>.tar.gz
```