PROCEEDINGS OF SCIENCE

The Automatic Installation And Configuration ProcedurE for the data acquisition system of KM3NeT

The KM3NeT Collaboration[‡*]
[‡] *full author list at PoS(ICRC2019)1177*
E-mail: emidio.giorgio@infn.it

The usage of tools for the automation of IT procedures has been increasingly spreading in the last years. The advantages of this approach include the traceability and the reproducibility of the performed steps, which implies reduced delivery times, possibility of roll-back as well as replicate configurations. Automation also minimizes errors, as human intervention is generally limited to the description of the general properties of the infrastructure components, like IP addresses, MAC address and so on, while configuration commands are coded, minimizing the chance of failures. A specialized and complex context like the data acquisition system of KM3NeT can largely benefit from the automation of tasks, in particular the reproducibility of configuration steps that allows to harmonize setups across different sites. This contribution describes the Automated Installation And Configuration procedurE (AIACE) developed for the online DAQ of KM3NeT, motivating the choices at the base of the current implementation and sketching possible evolutions.

Corresponding authors: Ronald Bruijn[†1], Tommaso Chiarusi[2], Emidio Giorgio[3]
[1] *University of Amsterdam, Institute of Physics/IHEF, PO Box 94216, Amsterdam, 1090 GE The Netherlands - Nikhef, Science Park 105, 1098XG Amsterdam, The Netherlands*
[2] *INFN - Sezione di Bologna, V.le Berti-Pichat 6/2 40127 Bologna, Italy*
[3] *INFN - Laboratori Nazionali del Sud, Via Santa Sofia 62 95123 Catania, Italy*

36th International Cosmic Ray Conference -ICRC2019-
July 24th - August 1st, 2019
Madison, WI, U.S.A.

[*] for collaboration list see PoS(ICRC2019)1177
[†] Speaker.

PoS(ICRC2019)898

1. Introduction

KM3NeT is a distributed neutrino observatory installed in the abisses of the Mediterranean Sea. It comprises two telescopes, ARCA and ORCA, 80 km off the Sicily coast (Italy) at the depth of 3500 m and 40 km off Toulon (France) at the depth of 2500 m, respectively [1]. Both of them are based on a grid of thousands Digital Optical Modules (DOMs), interconnected to shore stations via an electro-optical seafloor infrastructure. The DOMs are organized in vertical structures, the Detection Units (DUs), each one provided with a Base-module for controlling the power supply and optical amplification of the attached devices. Exploiting a custom FPGA-based White Rabbit kernel with Ethernet connectivity, the DOMs and Base-modules are submarine nodes of the global Layer 2 optical networking infrastructure [2], with two significant characteristics that make it unique with respect to the other DAQ networking installations: the asymmetry of the connections between the station and the detector and a custom implementation of the White Rabbit (WR) protocol [3] for synchronizing the detector DOMs and Base-modules, which exploits a hybrid layout of the onshore switching infrastructure. Beside shore stations, the DAQ context is, partially or entirely, replicated also in integrations sites, where DOMs are assembled in DUs, as well as test sites, where general functionality checks are performed. It is clear how it's important to have IT setups that are as much as possible each other similar, thus reducing the differences between production and test environments and allowing to test hardware and software components in a realistic scenario.

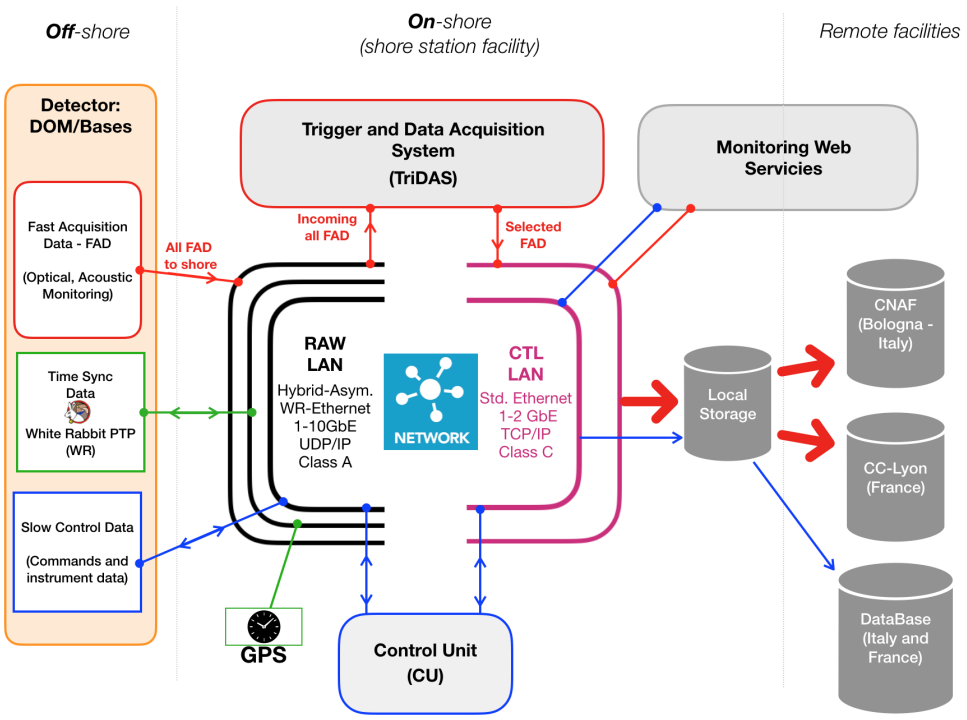


Figure 1: Outline of KM3NeT DAQ components

2. KM3NeT Trigger and Data Acquisition System

A detailed, and updated description of KM3NeT TriDAS can be found in Refs [2, 4]. However, it is worth to resume quickly its main components and their interactions. Offshore and onshore elements are connected through the *RAW LAN*, a specialized network which takes into account White Rabbit Protocol requirements and related customizations [3]. Onshore elements are connected also through another network, called *CONTROL LAN* as it is used to manage nodes and to reach them from external network. The *Control Unit* is a suite of processes that orchestrate the off-shore Detector and the DAQ facilities in order to optimize data-taking and preserve Detector efficiency [8]. The *Trigger and Data Acquisition System* identifies the group of computing resources that manipulate optical and acoustic data coming from DOMs. Filtered data is saved on local storage cache, and on a daily basis saved remotely to Tier 1, where they are made available for offline analysis. Besides these core DAQ components, it is worth mentioning *SDN controller* and *monitoring*. The SDN controller is needed because of the asymmetry of DAQ connections and the hybrid switching layout described here [3, 5]. The *monitoring* is a set of python scripts, exposed through a web server, that watches over data taken, permitting a real time evaluation of detector performance and allowing to spot, for instance, DOM failures or disruptions.
Finally, there is a node, the so called *bastion-host* that although not strictly playing a DAQ role it acts as a front-end with external network for the nodes, and it is normally used to provide utility functions, as virtual machines hypervisor or dhcp server for the control network.

3. Ansible

In the Information Technology scenario there is nowadays plenty of tools that allow software provisioning and configuration management, like Ansible, Puppet and Chef [6] just to mention the most widely known. A detailed comparison of such tools is out of the scope of this paper. However, with regards to Ansible, it can be observed that the most common of its deployments features a central node that sends commands via ssh to controlled nodes; this scenario matches quite well the typical outline of KM3NeT DAQ, with the bastion host that can be used to inject commands on the managed DAQ nodes.

3.1 Inventory, tasks, playbook and roles

A task is a command to be executed on a resource. Tasks are typically organized in playbooks, that can be seen therefore as a list of steps targeting a given state or a given configuration. It is mandatory to define one inventory file, at least. This file will contain the list of hosts that are managed by Ansible, and the related properties, as IP address or MAC address. Properties are expressed in terms of association *{key:value}*. Actually the choice of properties that can be assigned is free, so the list of properties used to describe the infrastructure can be modeled as needed. Properties are used by tasks to finalize hosts configuration. Hosts within an inventory can be organized in groups: this allow to apply tasks and roles only to group of hosts, as well as defining properties just for the subset of the hosts being managed. For instance, if we have a host that acts both as web server and dhcpd server, can be defined two groups, web and dhcpd, with web related properties set in the web server group, and dhcpd ones in the other. Web related tasks will

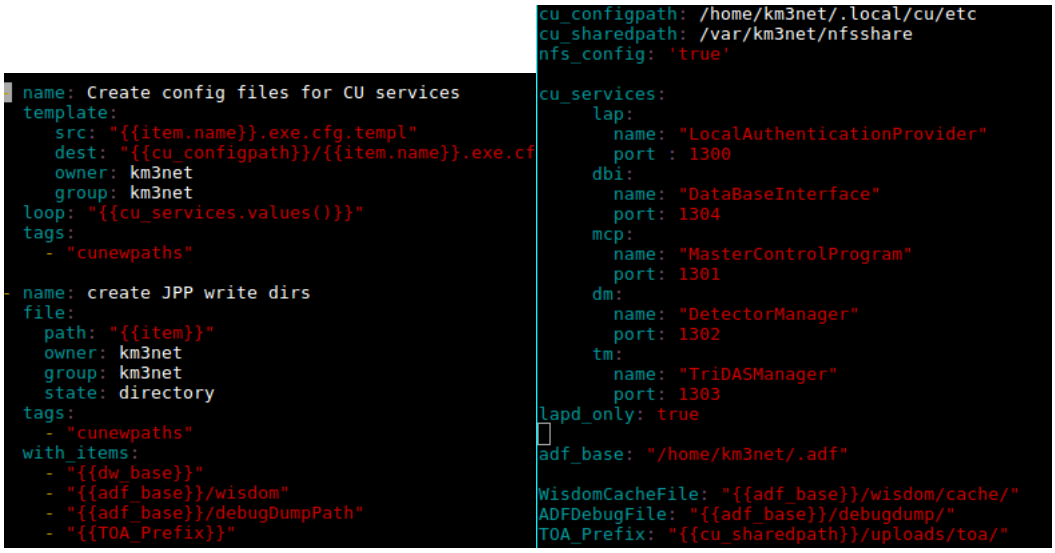
refer only to hosts in the web server group, and the same will do dhcpd related tasks. Tasks can be also organized in roles, that allow to reuse the same role in different contexts, invoking it from different playbooks. For instance, the role for software experiment installation is to be applied to more than one host group. So, instead of writing a playbook for installing software experiment, it is more convenient to define a role, and recall it from every playbook that needs it.

```
# INVENTORY FILE FOR ARCA STATION - 181210

arca-bastion-02 hostname=arca-bastion-02 ctl_ip='192.168.0.50' ctl_mac='00:1e:67:a4:04:1a'  vncport='20050'
arca-cu-02 hostname=arca-cu-02 raw_ip='10.10.102.2' ctl_ip='192.168.0.119' ctl_mac='00:1e:67:a3:fe:61' raw_mac='00:0f:53:20:66:d0'
   vncport='20119'
arca-dq-03 hostname=arca-dq-03 raw_ip='10.10.103.2' ctl_ip='192.168.0.123' ctl_mac='00:1e:67:a4:00:b4' raw_mac='00:0f:53:20:66:6c'
   vncport='20123'
sdn-controller-02 hostname=sdn-controller-02 ctl_ip='192.168.0.240' ctl_mac='08:00:27:14:27:f3' vncport='20240'
scsf ofid="openflow:542010846977778" cu_port="229" daq_port="155" scbd_uplink_port="235" raw_dhcp_port="229"
scbd ofid="openflow:550366641185744" scsf_link_portid="79" wrsb_ports=["77","78"]

[all:vars]
ctl_netmask= '255.255.255.0'
ctl_subnet= '192.168.0.0'
raw_netmask= '255.0.0.0'
raw_subnet= '10.0.0.0'
local_ntp_server='192.168.0.200'
production_site='true'
km3netsiteid=A00073795

[fen]
arca-bastion-02 publ_ip='172.16.65.150' publ_mac='00:1e:67:a4:04:1b' ctl_gw='192.168.0.50' ctl_static_ip='true'
     dns_ext_server='192.84.151.3'      os_install_server_path='http://ct.mirror.garr.it/mirrors/CentOS/7/os/x86_64'
     dhcpd_start='false' vbox_hyperv='true' skipvmcreation='true'

[control]
# Multiple nodes can be here, but only one can be active -> first
arca-cu-02 raw_bootmode='static' lapd_only='false' #dhcpd_start='false'
[daq]
# Multiple nodes can be here, but only one can be active -> first
arca-dq-03 raw_bootmode='static'

#
[sdnctl]
sdn-controller-02 karaf_version="0.7.3"

[doms]
# to add new doms, just add a row here, following
# the syntax below, and then run play_dhcpd_raw
# DOMNAME    DOM_RAW_IP           DOM_MACADDRESS
# CHANGEME     raw_ip="CHANGEME"    raw_mac="CHANGEME"
# EXAMPLES

du1base raw_mac="08:00:30:30:43:2b" raw_ip="10.0.1.100"
du1dom01 raw_mac="08:00:30:38:3f:5b" raw_ip="10.0.1.101"
du1dom02 raw_mac="08:00:30:37:fa:50" raw_ip="10.0.1.102"
du1dom03 raw_mac="08:00:30:38:4a:99" raw_ip="10.0.1.103"
```

Figure 2: An extract from ARCA inventory: on top, catch-all zone, followed by global variables, and then FrontEnd, Control, Daq, SDN CTL and doms group

3.2 Inventory, roles and playbook implemented for KM3NeT DAQ

These instruments have been used in the KM3NeT DAQ context to implement both installation and configuration scenarios. For each site, the inventory contains the list of infrastructure hosts, either physical or virtual. Since each host can play different roles (which is very common in small sites), each host is defined in a catch-all part, where common properties of the host, like control IP and related MAC address. Then there are groups for each logical set, like SDN controller, Control Unit, TriDAS, DOMs and so on. These allow to identify which host belong to which group, and apply playbooks selectively. Definition of hosts within such groups contains also the related specific property: for instance, TriDAS nodes have a boolean property that forces manual configuration of raw ip addresses avoiding dhcp invocation. Inventory contains definition also of some entities not managed via Ansible, for instance DOMs, whose mac addresses are used to configure the raw dhcpd, and SDN switches, whose definition are used for the SDN controller.



```
- name: Create config files for CU services
  template:
    src: "{{item.name}}.exe.cfg.templ"
    dest: "{{cu_configpath}}/{{item.name}}.exe.cf
    owner: km3net
    group: km3net
  loop: "{{cu_services.values()}}"
  tags:
    - "cunewpaths"

- name: create JPP write dirs
  file:
    path: "{{item}}"
    owner: km3net
    group: km3net
    state: directory
  tags:
    - "cunewpaths"
  with_items:
    - "{{dw_base}}"
    - "{{adf_base}}/wisdom"
    - "{{adf_base}}/debugDumpPath"
    - "{{TOA_Prefix}}"
```

(a) Some of the CU tasks, involving creation of directory and JPP binary path

```
cu_configpath: /home/km3net/.local/cu/etc
cu_sharedpath: /var/km3net/nfsshare
nfs_config: 'true'

cu_services:
    lap:
      name: "LocalAuthenticationProvider"
      port : 1300
    dbi:
      name: "DataBaseInterface"
      port: 1304
    mcp:
      name: "MasterControlProgram"
      port: 1301
    dm:
      name: "DetectorManager"
      port: 1302
    tm:
      name: "TriDASManager"
      port: 1303
lapd_only: true

adf_base: "/home/km3net/.adf"

WisdomCacheFile: "{{adf_base}}/wisdom/cache/"
ADFDebugFile: "{{adf_base}}/debugdump/"
TOA_Prefix: "{{cu_sharedpath}}/uploads/toa/"
```

(b) Default directory paths and binaries ports for CU role

Figure 3: Control Unit role : tasks (a) and default values (b). Tasks defined in role can be imported in any playbook. Some of the meta variable have default values set in a specific file

Few playbooks have been implemented, each of which aiming to the configuration of the main actors : FrontEndNode (Bastion), ControlUnit, DAQ node, SDN controller. Instead of putting directly tasks for each playbook, roles have been defined instead, in order to facilitate re-use. For instance, for the installation of experiment software, a dedicated role (*km3_sw_env*) has been written, that configures the repository and then downloads and installs software packages. This role is invoked both from CU and DAQ, avoiding to maintain two different set of tasks performing the same operations in two different playbooks. Another example is the configuration of *raw* interface, a role needed by all nodes that communicate offshore (DAQ and CU). Another reason to use roles is that they naturally imply the use of templates. Although it is not mandatory, when the role implies generation of configuration file it is very convenient to use templates, which are transformed in real configuration files reading the inventory and replacing the template placeholder value with the actual value for the node. It also allowed to set predefined values, while a fatal error is generated if a placeholder can't be expanded. A role that largely exploits this template feature is the PXE server role. A PXE server is a service used to bootstrap nodes and kickstart the operating system installation. Therefore the server needs to know the MAC address and related IP for the nodes being the installed. The PXE role include template files that are transformed in real PXE configuration files reading IP and MAC address related values within the inventory file. The FrontEndNode invokes the largest number of roles indeed. It also plays as control dhcpd server, but provides also a local kickstart server that can be used for the nodes installation, and acts also as NAT, tunneling external connections for the DAQ nodes. Although these duties are generally played by a single node, yet they have been implemented as a role, and not within playbooks, because this facilitates re-usage in other contexts: for instance it could be needed to move the dhcpd server to another host: in a role-scenario it is enough to apply the role to the desired node, while a playbook implementation

```
# CU specific configuration tasks

- name : Configure Control Unit
  hosts :        # the inventory group targeted
    - control
  roles:         # invoked roles
    - km3_rawif
    - km3_sw_env
    - km3_cu
  vars:
     debug: "false"

  handlers:
    - name: restart network
      service: name=network state=restarted

  tasks :      # the following tasks will be executed after roles
     - name: check script pushing static arps
       stat: path=/etc/NetworkManager/dispatcher.d/11-static-arps
       register: static_arps
       tags: arp

     - name : Create static arp
       lineinfile:
          create: yes
          path: /etc/ethers
          state: present
          line: "{{ hostvars[item]['raw_mac']}} {{ hostvars[item]['raw_ip'] }}"
       with_items:
          - "{{ groups['doms'] }}"
```

Figure 4: An extract from Control Unit playbook. It begins from the inventory group targeted by the playbook, then roles invoked (raw_if, sw_env, km3_cu) and finally few specific tasks that, differently from those defined in roles, cannot be reused

would require a deeper editing.

4. Ansible in action

This section describes the whole procedure implemented by AIACE in order to set up the KM3NeT DAQ context. The first node being installed is the FrontEndNode (FEN). This node is a plain CentOS 7, better if installed through kickstart in order to predefine some settings. When the FEN is installed, AIACE scripts are downloaded from a git repository, and the first step to be taken is the compilation of the inventory. Then the first playbook can be executed, that actually configures the FEN. The major tasks carried out by the FEN playbook are the creation of a local kickstart server for the other TriDAS nodes being installed, the definition of a dhpcd server for the CONTROL network and the configuration of the interface in order to provide a gateway for the external network.
Once that FEN configuration is completed, set up of the other nodes can follow. The procedure is similar for all of them. If the FEN playbook has been run successfully, the node is exposing a PXE/kickstart server, filled with entries related to the other nodes to be installed. Similarly, the DHCP server for the CONTROL network is already setup, so it is enough to reboot the nodes to be installed, in order to trigger the OS installations. When all the nodes are installed, they can be configured through their respective playbooks. Although there is not a specific order, it

is convenient to start from the SDN controller: this playbook, differently from ControlUnit and DAQ, does not install KM3 software, so *km3_sw_env* role is not linked from the sdn playbook. It is rather invoked *km3_sdn role*, which downloads Karaf [7], the runtime environment upon which the SDN controller chosen, OpenDayLight, is built. The inventory file is sourced in order to discover the switch ports where TriDAS nodes are attached, and coherently build the SDN rules that are eventually pushed to the controller [5]. If everything has gone fine, raw network connectivity is now fully enabled, and only ControlUnit and TriDAS nodes are left. Two different playbooks have been prepared for this purpose, ControlUnit and DAQ, that actually do not differ very much: they spend both the largest part of their time executing *km3_sw_env* role, which performs installation and configuration of experiment software (mainly JPP), and *km3_cu* role, which installs the ControlUnit software, that so is present in CU and DAQ nodes. It is convenient to start from the ControlUnit, as this normally carries a DHCP server for raw network, and complete with the TriDAS nodes. When the CU software is installed, a boolean value on the inventory determines if the node has to behave as ControlUnit for the detector, or just act as an agent for the main ControlUnit process. However, at any time roles can be changed exploiting ControlUnit Dynamic Provisioning [8], that also provides failover strategies in case one of the binaries in the main CU process dies unexpectedly. The last steps performed in the role is the creation of configuration files, starting from predefined templates and the creation of work directories for CU binaries and JPP, where detector data will be stored.

5. Next evolutions

The work here described has been used in KM3NeT production sites since January 2019, when ARCA detector was restarted after a downtime period. In the following weeks AIACE was used to install KM3NeT DAQ context in the Catania test site, Bologna Common Infrastructure and Strasbourg integration site. Production usage stressed several areas for improvements. For instance, the playbooks are all oriented to the installation and configuration of software, with directories and configuration files continuously recreated. While this feature has been extremely useful during detector "commissioning" (when detector whole configuration is being tuned), it became quite uncomfortable when the detector was fully operative: at this point, reconfigure a whole stack just to correct a package configuration was really inefficient, also because some post-configuration aspects could be lost. This behavior was already adjusted but will be more enforced in the next versions of AIACE, with playbooks able to reconfigure software or perform package upgrades preserving as much as possible the existing setup. Another useful extension would be some playbook that just perform configuration checkup without any executive action. Another change will be led by the fact that RPM packages, current format used for the deployment of experiment software in AIACE, will be no longer the supported format for next major releases, which will adopt container instead. This will obviously impact the role deploying KM3NeT software, but probably also the configuration of related nodes, with some aspects that will be embedded in the container and will then disappear, while others, like the IP address of the node, will have to be moved on a different step. Finally, some more component will be included: monitoring, but also the configuration of White Rabbit and SDN switches (currently, AIACE manages just the controller installation and configuration).

References

[1] S. Adrían-Martínez et al. (KM3NeT Collaboration) *Letter of Intent for ARCA and ORCA* in *J. Phys. G: Nucl. Part. Phys.* 43 (2016), 084001

[2] C. Pellegrino and T. Chiarusi et al., *The Trigger and Data Acquisition System for the KM3NeT neutrino telescope* in *EPJ Web of Conferences* 116 (2016) 05005

[3] M. Bouwhuis, *Time synchronization and time calibration in KM3NeT* in proceeding of *ICRC 2015* PoS(ICRC2015)1170(2016)

[4] T.Chiarusi and R. Brujin, *KM3NeT DAQ and Trigger* in proceeding of *this ICRC conference*

[5] T. Chiarusi, L. Chiarelli, E. Giorgio, S. Zani, S. Celli and on behalf of the KM3NeT Collaboration, *The Software Defined Networks implementation for the KM3NeT networking infrastructure*, in proceeding of *ICRC 2017* PoS(ICRC2017)940(2018)

[6] Ansible Home, Chef Home, Puppet Home

[7] Karaf Home

[8] C. Bozza, T, Chiarusi *The Control Unit of KM3NeT detectors*, in proceedings of *this ICRC conference*