JOINT INSTITUTE FOR NUCLEAR RESEARCH
Laboratory of Information Technologies

FINAL REPORT ON THE SUMMER STUDENT PROGRAM

Development of continuous integration and deployment pipeline for the GNA project and reengineering of Cloud monitoring system

Supervisor:
N. Balashov

Student:
Igor Kuprikov, Russia
Dubna State University

Participation period:
July 01 – August 23

Dubna, 2019

Contents

Abstract

In this paper, I describe the tasks that I completed related to JINR projects. I participated in 2 projects and came across tasks such as Restructuring the cloud monitoring system, writing pipeline for automatic assembly, testing and loading the GNA framework into the Docker registry, debugging the OpenNebula [1] platform scripts, deploying the Icinga monitoring system and building monitoring dashboards in Grafana [2].

Introduction

During this JINR summer student program I have learned a lot of new technologies, including software frameworks and complex software platforms. I took part in two projects: “JINR Cloud” and “Development of continuous integration and deployment pipeline for the Global Neutrino Analysis (GNA) project”. The JINR Cloud is a computing infrastructure providing users with virtual machines. It has two goals: the first is to simplify user access to physical computing resources by virtualizing them; the second is to optimize resource consumption. The monitoring system is one of the critical components of the JINR Cloud, but its database (DB) schema did not allow to built some important plots due to initial design flaws. My assignment was to analyze the current DB schema and come up with the new one that would eliminate major design flaws. The GNA is a framework for a statistical data analysis for neutrino neutrino experiments. The main features of the framework are dataflow computations, high numerical performance, flexibility and extensibility. The framework is developed by neutrino physics group of DLNP JINR and is being adopted by external users from JUNO experiment. My task in this project was to develop a Continuous Integration and Deployment pipeline (CI/CD) to automate testing the framework and building a distribution for end-users. Such a system improves the development process by reducing the debugging time and potentially leading to better code quality, and the automatic distribution production simplifies the installation process for the end-users.

As a side task I also volunteered in an IT-School which was based in Dubna University.

Development of continuous integration and deployment pipeline for the GNA project

The goal of my first assignment was to automate the build and test process of the GNA framework. The framework needed to be assembled in a docker container, the purpose of which was to facilitate the process of software distribution. Since the use of the container facilitates the installation task, the container is a lightweight copy of the operating system in which only the necessary dependencies for the target program are installed. The GNA framework is built in a container in 4 stages:

1) *On the first stage a docker image is built containing the ROOT framework with all the necessary dependencies and is published in a docker registry manually. This image is used as base for the final GNA docker image built in later stages;*

2) *Next, the docker image with the GNA is built automatically using the built-in CI/CD system of the git.jinr.ru service;*

3) *The final GNA image is automatically tested to verify the framework functions as expected;*

4) *If the tests succeed, the GNA image is the automatically published in the registry, otherwise the pipeline fails and developers are provided with the debugging data;*

To do this, it was necessary to write a .ci-yml file with the instructions for gitlab-runner to build the framework. At the initial stage of the assignment, a stable version of GNA and tests were provided. The scheme of the resulting pipeline is illustrated on Figure 1.

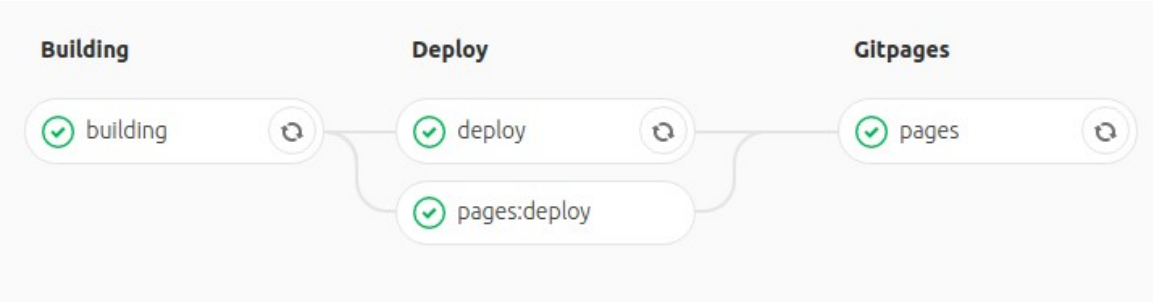

Figure 1 – GNA assembly pipeline structure

As a result, the developed pipeline eliminates the need of running the tests manually by the developers. It also automatically builds the framework distribution which greatly simplifies the installation process for the users: they only need to download and run the docker image.

Reengineering of Cloud monitoring system

My second assignment was related to the Monitoring system of the JINR Cloud. Its goal is to facilitate the Icinga 2 as a network monitoring system; it provides a large number of binary files with ready-made polls for hosts and virtual machines. Monitoring can take place with or without agents. Agents are installed on the interrogated machines, and the main machine that interrogates interrogates these agents. Without agents, a monitoring host can only check the state of the machine (on / off, whether it listens on the port, etc.) and interrogate some information from the inside through special sockets. My goal was to verify the compatibility of the old “Nagios” monitoring system with the newer “Icinga”. During my practice I installed and configured Icinga2 platform and developed some additional polls. The intention was to prepare the working test-bed to be later used by the JINR system administrators to migrate from the old Nagios system (which is planned to be decommissioned) to a newer Icinga2 system.

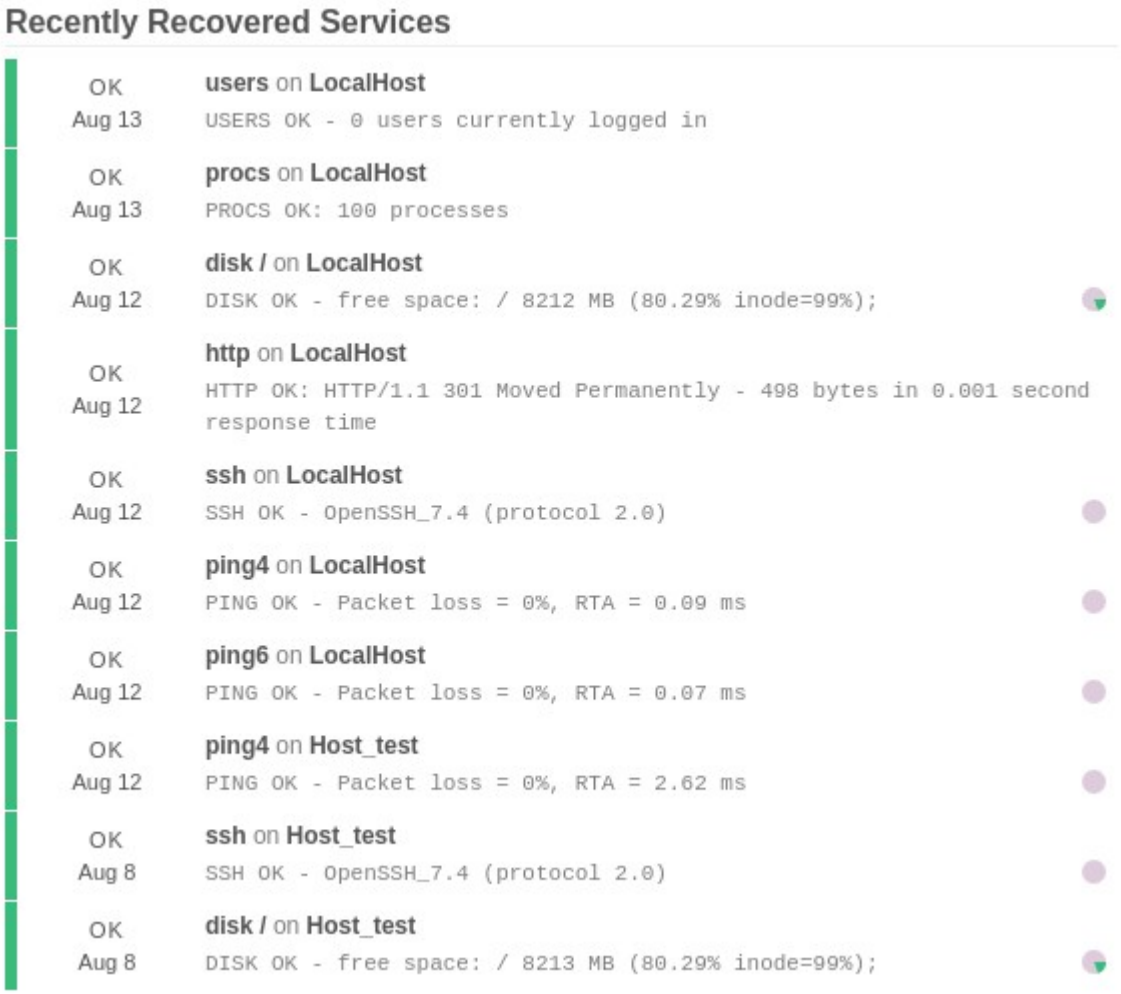

Figure 2 – Example of hosts asking from Icinga

Current task was connected with the OpenNebula and Grafana platforms, as well as InfluxDB [3], a database with temporary stamps. OpenNebula (ONE) is a platform that allows you to configure cloud computing, consisting of clusters, hosts, virtual machines, configure virtual networks for communication between clouds, as well as track physical and virtual computing objects. You can get it in many ways, but the most suitable method was using XML-RPC requests. Using the Ruby API for XML-RPC [4], data about hosts is requested in the form of XML templates, with which you can find everything you need for template development. Graphic data for building graphs and tables, for more convenient monitoring of the cloud system. OpenNebula survey structures can be completely different, and the current survey structure does not require graphing restrictions; my task should be rewritten using reference clouds, which can be from a large number of metrics. The monitoring data collector works in 3 stages:

1) The process of querying the cloud and getting simple XML templates with values;

2) Formation of data structures from XML templates;

3) Writing data to InfluxDB;

After data sending to InfluxDB, this data will use on Grafana. Grafana helps to monitor systems like ONE. Grafana can create graphs and other kind of panels which helps with data visualization.

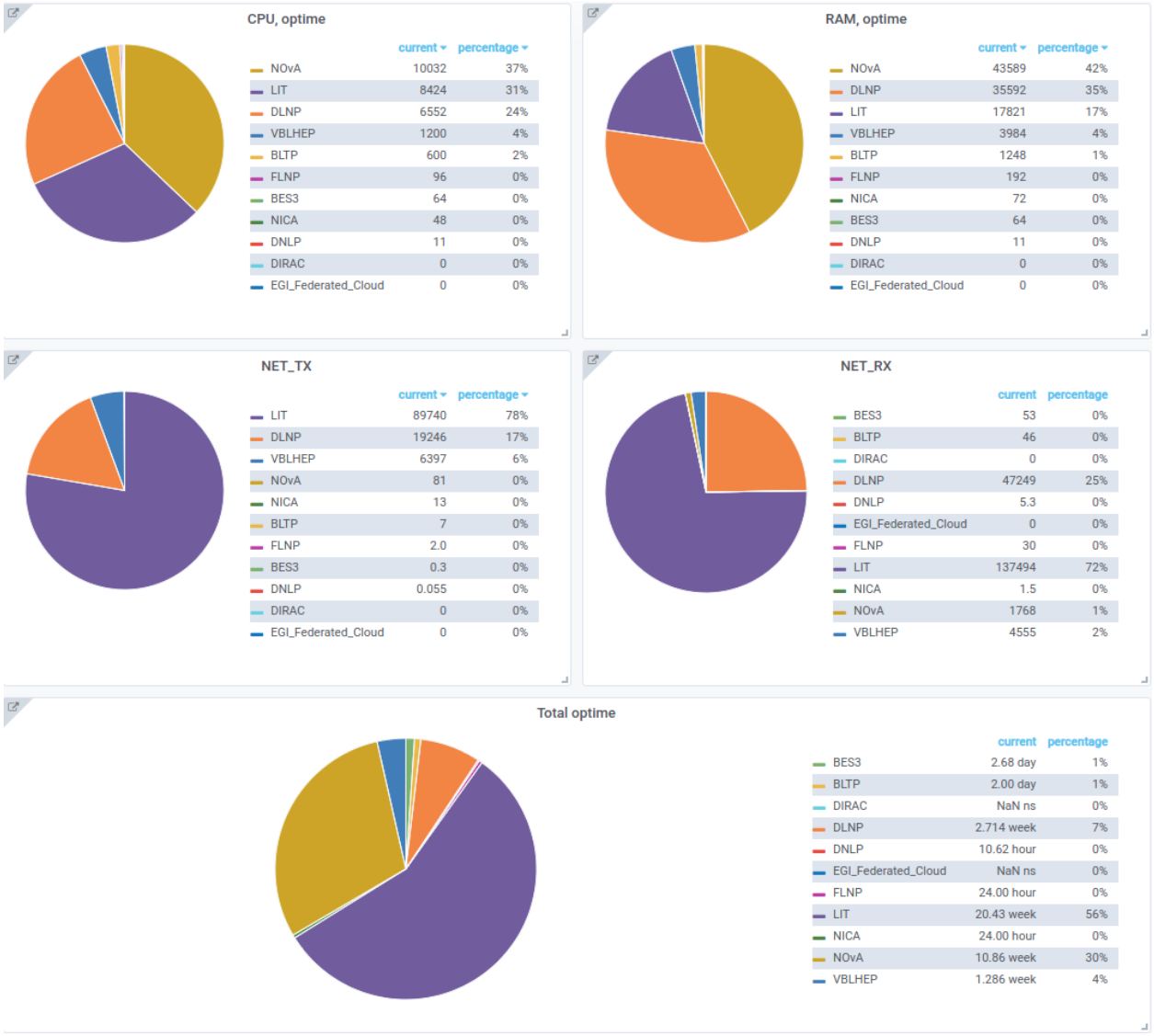

Figure 3 – Example of data visualization in Grafana

This is mine panel which visualize accounting data per project, about resources which used by laboratories.

The next assignment was also related to the OpenNebula (ONE) platform which is the base of the JINR Cloud. In the process of updating the cloud, the database structure, which contained all the necessary information of the cloud, could change. Specially for this, the script called onefsck exists, which task is to check the database and, if possible, correct the inconsistensies in it. When updating the ONE 5.8.1 to a new version 5.8.4, the script generated invalid fields in the database at startup. It was required to find and fix the error. Using the debugging method, a part was found in which this error arose and some changes were made. After the changes, the script stopped generating extra fields.

References

1. OpenNebula Release Notes 5.8 / OpenNebula Project

URL: http://docs.opennebula.org/5.8/intro_release_notes/release_notes

2. Icinga2 – Platform which provides useful network monitoring

URL: https://icinga.com/docs/icinga2/latest/doc/01-about/

3. Grafana - Beautiful Metrics, Analytics, dashboards and monitoring

URL: http://grafana.org/

4. InfluxDB – Time-Series Data Storage / InfluxData

URL: https://influxdata.com/time-series-platform/influxdb/

5. Class XMLRPC Client (Ruby 2.3.1)

URL: http://ruby-doc.org/stdlib-2.3.1/libdoc/xmlrpc/rdoc/XMLRPC/Client

Acknowledgement

I would like to express my appreciation to my practice supervisor, Director of University Centre of Joint Institute for Nuclear Research, for giving me an opportunity to participate in Student Summer Program. I also wish to acknowledge the help provided by Nikita Balashov and Aleksandr Baranov for consulting me in my work. I would like to thank to Elena Karpova for excellent organization of our practice, It’s my first and the greatest experience at working in team, thanks for everyone who was helping me.