Open Source Cloud Costing Framework

Author: Anastasia Voronkova, vvanast@yandex.ru
Supervisor: Luis Fernandez Alvarez, luis.fernandez.alvarez@cern.ch

CERN Openlab Summer Student Programme 2021

Abstract
In the last years the amount of resources that are needed to conclude experiments at CERN grew rapidly, and some computations began to be performed in clouds. To control them we need some tools for cloud costing. The goal of this project is to deploy and test one of these tools -Koku- for future use it in the Batch Service with Microsoft Azure cloud platform.

Keywords
CERN Summer Student Report, cloud cost, optimization, Microsoft Azure

1 Introduction

The Batch Service has been provisioning since 2019 resources in Microsoft Azure to provide compute capacity. During this time the control and visibility of costs has been a key area to manage, exploring different solutions, from Azure built-in tools to open-source alternatives via Prometheus. A new project, called Project Koku, could simplify and provide a long-term solution for these tasks.

Project Koku is an open source solution for cost management of clouds, containers, and hybrid cloud environments. It is designed to identify and report cost data associated with various sources, including Microsoft Azure.

The aim of this summer project is to study existing cloud costing tools and deploy and evaluate Project Koku in the context of the Batch Service. The main points to evaluate will be its features as a tool to visualise and analyse the resource usage and expending done in Microsoft Azure by the different compute capacity deployed: batch and HPC.

1.1 Review of existing tools

Nowadays the task of managing cloud costs is very popular, so there are a lot of tools to do it. Most of them are integrated into cloud platforms or are paid services, so in fact there aren't many open source projects that we can consider. Exploring the open source ecosystem, we have identified the following list of tools for cloud costing control: Project Koku [2], Kubecost [3], Infracost [4] and Cyclops [5]. There were revealed several requirements for such a tool - it should support Microsoft Azure, provide detailed information on all groups of cloud resources, show visualizations for costs and usage, forecast future costs, warn about too high costs, it should be possible to deploy it to existing system and it should not require dependencies that are hard to embed to the system.

1.1.1 Project Koku

Koku is basically an open source tool that is used by Red Hat to provide the cost management functionality in the Red Hat Insights portal [1]. It also supports the major cloud providers as data sources including Microsoft Azure, and allows visualization of the resource usage and costs, predict future spending and optimize them. Moreover, unlike other tools it can gather metrics from all cloud elements.

Koku consists of 3 main parts -Koku API (reporting and query API service), Koku UI (web interface) and Masu (data ingestion service)-. The API is written in Python using Django, and the frontend

is written in JavaScript using React. There are two components in Koku API: web service and database. PostgreSQL is the default database engine and can be replaced by Trino (formerly Presto) with MinIO for object storage. Database query monitoring can be done with Grafana.

There are several dependencies:

- Docker
- PostgreSQL or Trino and MinIO
- Redis
- pgAdmin
- RabbitMQ
- Prometeus
- Grafana

1.1.2 Kubecost

Kubecost has open-source and commercial version. The first has rather limited functionality compared to the second, although both versions support visibility of Kubernetes' current and historical expenses and resource allocation on different levels from nodes and pods to global perspective. Although it doesn't support cost visibility for services that are outside of Kubernetes or other cloud resources. Kubecost can also send alerts of anomalous spending and recommend resource optimization.

Kubecost retrieves resource usage data from Kubernetes and prices from the chosen cloud computing platform, calculates costs of resources, caches it and returns to Prometheus. Then it can showed in Grafana or its own UI. Also in cases when it is needed to expand system to a large number of nodes, Thanos can also be used, but probably only in commercial version.

There are several dependencies:

- Kubernetes version 1.8 or higher
- Prometheus
- kube-state-metrics

1.1.3 Infracost

Infracost shows cloud cost estimates for infrastructure-as-code projects such as Terraform in command line interface, as well as in pull requests and CI/CD. The tool supports a lot of different Terraform resources in Microsoft Azure. Although there is no any visualization or prediction of costs for now, but results can be exported in HTML or JSON format.

The tool consists of command line interface and Cloud Pricing API. CLI parses Terraform plan JSON file for supported resources and Cloud Pricing API returns the prices to CLI and it calclulates monthly costs.

There is only one dependency in this case - Terraform.

1.1.4 Cyclops

Cyclops is a dynamic rating-charging and billing solution for cloud providers. It measures resource usage and generates dynamic rates according to different factors and calculates bills for users. Finally it visualizes usage and costs data in charts and statistics in dashboard for administrators. Also it can predict usage of cloud resources using ARIMA models based on activity of one user, group of users or all users. Currently Cyclops supports only CloudStack and OpenStack (Events and Ceilometer) and it was planned

to add support for more cloud services, but it looks like authors have stopped development of the project in 2019.

There are 3 main micro services: UDR micro service (usage data collection), CDR micro service (pricing generation) and Billing micro service (invoicing and discounting). UDR micro-service collects usage data from cloud computing software and save harmonised data to Time Series Based Database. CDR uses rule engine to generate dynamic rates based on environmental and other factors, while Billing micro service generate invoices.

There are several dependencies:

- Java 8 and Maven 3
- PostgreSQL 9.6
- RabbitMQ 3.6

1.2 Tool selection

In conclusion, Kubecost supports gathering of cost data only from Kubernetes and Infracost does not have any visualization or prediction of costs. Cyclops provides wider opportunities, but does not support Microsoft Azure and as it looks like an abandoned project, there is no much hope that it can change. On the contrary, Project Koku supports data collection from multiple cloud sources, provides visualizations and predictions of future spending and it is actively maintained.

2 Koku installation and usage

As Koku is a big project, we need to do two separate installations of different parts of the project. In the documentation of the project there is a rather good description, but some steps were actually missed [6]. Firstly, we need to install PostgreSQL and Docker and then we can install Koku API and Masu simply using Docker Compose as it is written in README of github project [7]. To install Koku UI we need to install firstly NodeJS and Yarn, then clone the github project [8] and install it using Yarn. Next we should configure Microsoft Azure for Koku access and then add to Koku as a source of data [9].

During the summer program Koku was deployed in one of virtual machines hosted on the CERN Openstack Infrastructure. As operating system was chosen CentOS 7, since it is one of the most popular operating systems in CERN, so it is important to test new tools in it. Although previously I didn't have experience with it, so it took some time to start to consider this operating system convenient.

2.1 Koku installation problems

As it was found during the study of Koku UI, it uses React and it has deep connections to Red Hat Insights and it uses several Red Hat cloud services components for web applications. This fact turned out a rather big obstacle in standalone deploying of Koku. It was shown that there is a way to run Koku UI on localhost without Red Hat services [10], but it is not suitable for actual deployment. After discussion this problem with developers of Koku it became clear that they don't have plans to separate Koku UI from Red Hat dependencies yet, although the project is positioned as suitable for several other cloud computing platforms. At least probably they will describe a method of standalone deploying of the application. For now we haven't found a method to do it by ourselves due to the lack of expertise in JavaScript and the complexity of refactoring all the dependencies. Actually as the one that is not working is the frontend part of application, it was investigated further the most important part - the functionality of the backend of project.

3 Koku evaluation

3.1 Current costs and usage

Koku collects cost data from Microsoft Azure via the Masu component. This data is processed and then an aggregated version is provided in the web interface - Figure 1 or it can be also obtained through API. We can get JSON with total costs like this:

`GET http://localhost:8000/api/cost-management/v1/reports/azure/costs/`,

inventory data like this:

`GET http://localhost:8000/api/cost-management/v1/reports/azure/instance-types/`

or storage inventory data:

`GET http://localhost:8000/api/cost-management/v1/reports/azure/storage/`.

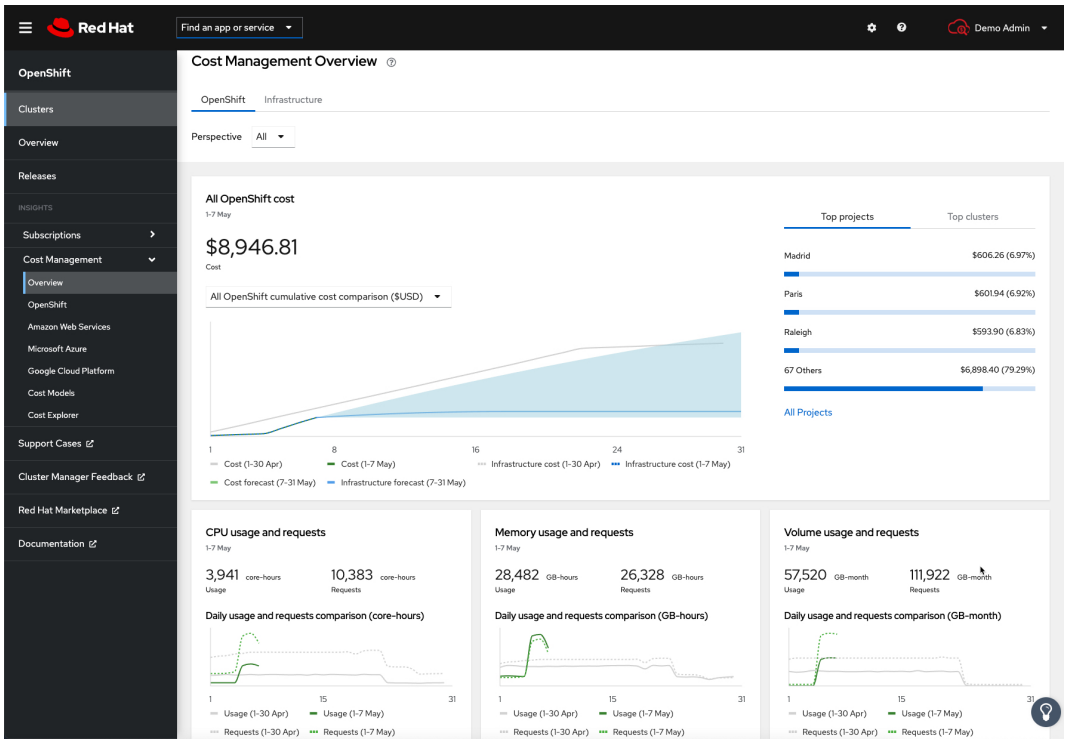

Figure 1: Web interface of Koku

In all cases we can also use grouping or filtering by name of project. Also we can add several users with different rights in Koku, so that they can see the cost statistics associated to their group or role.

Koku defines several types of costs - infrastructure costs and supplementary costs. The first are costs reported by cloud provider and the second are costs not directly related to infrastructure. So in most cases total costs and infrastructure costs are the same. As we can see from JSON, there are also some other fields. Raw costs are just costs from cloud provider, markup is the part of cost that was removed from bill after applying markup or discount. Usage cost is the part of cost, calculated from hourly or monthly price and usage and then total cost is the sum of costs from other fields. More can be read in Documentation for Red Hat Cost Management Service [11]. Here is an example of JSON with costs for 24th of August:

```
...
"data": [
        {
```

```
            "date": "2021-08-24",
            "values": [
                {
                    "date": "2021-08-24",
                    "source_uuid": [
                        "04ba4cfc-f662-4ab1-882b-9de2783799a1"
                    ],
                    "infrastructure": {
                        "raw": {
                            "value": 521.896673715,
                            "units": "USD"
                        },
                        "markup": {
                            "value": 0.0,
                            "units": "USD"
                        },
                        "usage": {
                            "value": 0.0,
                            "units": "USD"
                        },
                        "total": {
                            "value": 521.896673715,
                            "units": "USD"
                        }
                    },
                     "supplementary": {
                     ...
                    }
                }
            }
        ]},
        ...
    ]
```


3.2 Predictions in Koku

Another important function of Koku is cost forecasting. We also have access to it through API:

`GET http://localhost:8000/api/cost-management/v1/forecasts/azure/costs/`

Koku forecast costs using very simple linear model - Ordinary Linear Squares model from popular statsmodels [12] library. It is a model where we step by step optimize the squared difference of the true value and prediction. It is a simple, but powerful model, although in Koku is used only the total cost as feature. And the prediction is also total cost. At the same time there can be other parameters that can lead to significant changes in costs - like day of week, use of expensive resources or very rapid growth in popularity of less expensive resources. Also for better prediction there need to be used a lot of data, but Koku uses only data for the current month or for the previous if in current month the amount of days is less then 8. Obviously this can be not enough for reliable predictions.

Despite rather simple model everything is done from the point of statistics - removing outliers using interquartile range, calculating the quality of prediction using 2 methods. First is coefficient of determination - proportion of variation of dependent variable that can be explained from independent variable, and the second is using confidence intervals.

On Figure 2 is showed a chart of historical costs (in dollars) in blue and predicted costs in orange. Historical costs are given for 9 days, they seems rather unstable, and Koku predicted a stable upward trend for next 15 days. This looks too ideal and probably Koku has problems with prediction because of

short period of gathering historical data and it's unstable manner.

Another thing that should be mentioned here that Koku doesn't have its own system of notifications if some costs became too big, but it is desirable in practice.

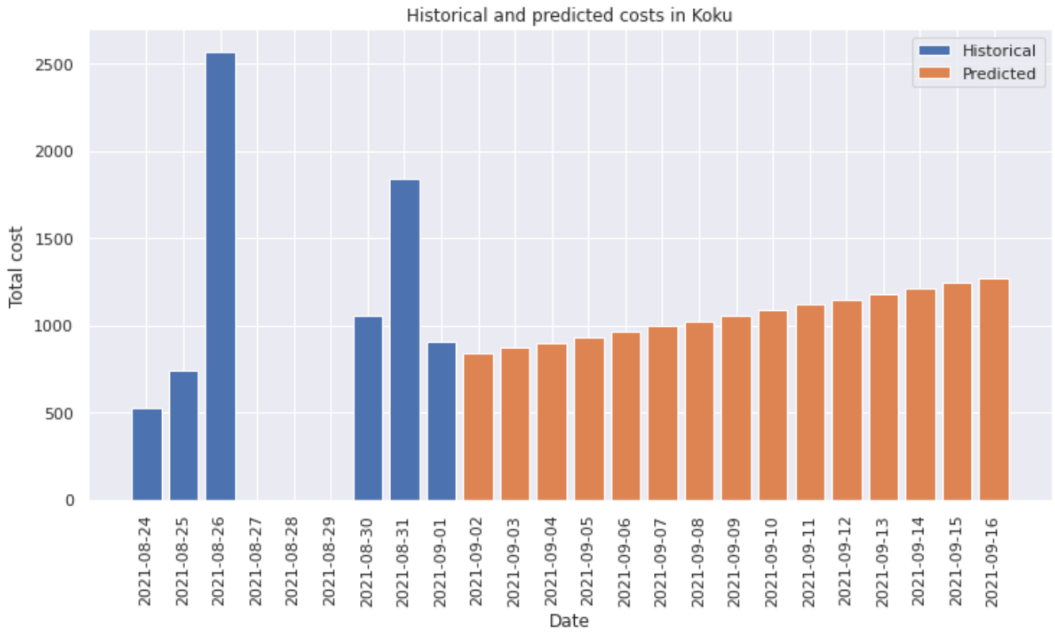

Figure 2: Historical costs and predicted by Koku

3.3 Discussion

As it was mention in the previous chapter forecasting mechanism in Koku is rather simple and can not predict all the fluctuations in costs. Although due to its simplicity it works more stable. Currently the problem of time-series prediction is rather popular and there are a lot of different tools for it, although in this task we have only daily data, so there are many points to consider. At the same time this limitation didn't stopped authors of Cyclops from using popular model for time-series - ARIMA model. However the problem of having not so much data can be also solved with using not only cost data, but also quantity of units of different groups of resources, day of the week and some other factors.

For example we can look on Figure 3 - a chart with costs from Microsoft Azure for August (in CHF) with splitting to type of resource. It is clearly seen that costs for virtual machines are the most expensive sector and their costs grow drastically in the end of the month. At the same time other groups cost much less with exception of ExpressRoute. If we look at Figure 4 with chart of quantity of different types of resource, it can be easily seen that the amount of used resources grow for all groups. So we can suppose that for prediction of costs the most important groups would be the most expensive - virtual machines and ExpressRoute.

4 Conclusion

All in all, through the internship were reviewed several tools for cloud cost management, was tested one of them -Koku-, because it satisfied most of the criteria of such a tool and also were proposed ideas to make a better prediction model than that is used in Koku. Through the work were identified several problems with Koku - it really depends on Red Hat ecosystem and it requires more time to investigate how to eliminate these dependencies. Also it was found that Koku's forecasting function may return not very accurate predictions for short periods of time, so there were described some ideas of improvement.

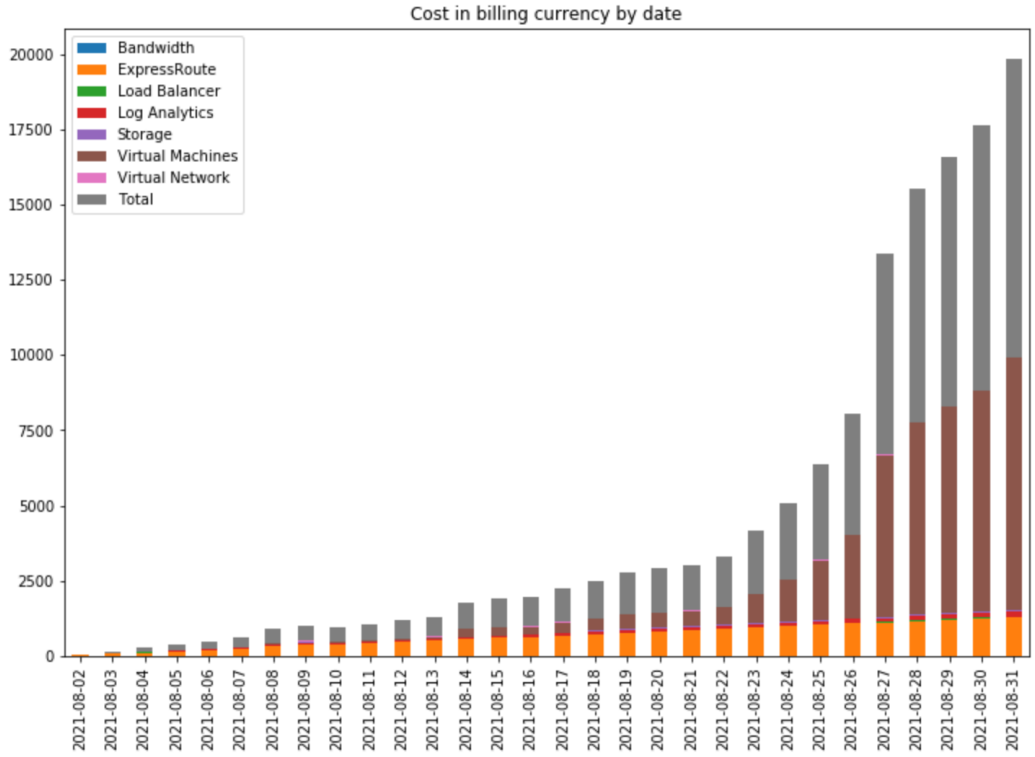

Figure 3: Cost in CHF by date and by group

Unfortunately, due to time constraints, it wasn't possible to test these ideas on real data. Finally, we should mention lack of notification system in Koku, but it should be not so hard to add it. Koku is a great tool, but for now it requires some additional work, so further investigation should be done to decide whether investment on its modification would be valuable over exploring other alternatives or custom tools.

Acknowledgements

Openlab Summer Student Programme was a great experience - I learned a lot of new things about Linux and cloud resource management, program brought deeper understanding of international culture and it was interesting to look at how top organisations like CERN change their working processes during COVID-19. I got into an amazing team of specialists and although team specialization was very new to me, they warmly accepted me in team and it was interesting to listen to the discussions on meetings. And most of all I want to give thanks to my supervisor, who shared with me his great experience and helped with all hard problems I faced, especially in the beginning, when everything seemed to be so mysterious and unclear.

Bibliography

[1] Red Hat Cost Management Tool `https://cloud.redhat.com/learn/topics/cost-management`

[2] Project Koku, `https://project-koku.github.io/`

[3] Kubecost, `https://www.kubecost.com/`

[4] Infracost, `https://github.com/infracost/infracost`

[5] Cyclops, `https://icclab.github.io/cyclops/`

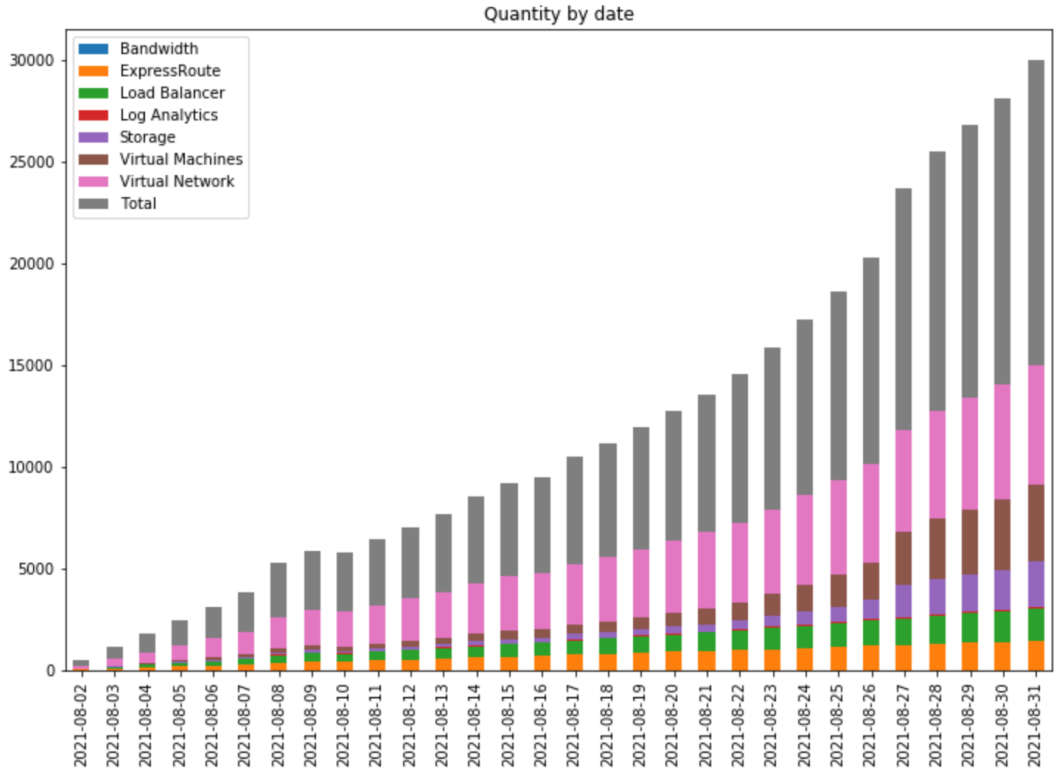

Figure 4: Quantity by date and by group

[6] Project Koku Documentation, `https://koku.readthedocs.io/en/latest/`

[7] Koku project, README `https://github.com/project-koku/koku`

[8] Koku UI, `https://github.com/project-koku/koku-ui`

[9] Project Koku Documentation, Adding an Azure Account `https://koku.readthedocs.io/en/latest/sources.html#adding-an-azure-account`

[10] Frontend-components, README - standalone running of application `https://github.com/RedHatInsights/frontend-components/tree/master/packages/config#standalone`

[11] Red Hat Customer Portal, Documentation for Cost Management Service `https://access.redhat.com/documentation/en-us/cost_management_service/2021/html-single/using_cost_models/index`

[12] Statsmodels, `https://www.statsmodels.org/stable/index.html`