EASE MAPE System
Framework for automatic management of cloud computing resources

Valentin Laquit
Intern at Department of Computer Engineering and Software Engineering
Polytechnique Montreal
Montreal, Canada
Email: valentin.laquit@gmail.com

***Abstract*—The EASE MAPE System project is a framework design project for the automatic management of IoT resources. It is part of a development process within the EASE group. It is intended to be reused according to different needs.**

The system is based on a MAPE architecture: Monitoring, Analysis, Planning and Execution. The MAPE architecture is in the form of a loop. We can link the IoT application we want to manage.

In order to provide an example of a MAPE loop, the project also contains an implementation for IoT services deployed with Docker Compose. This concrete version of the loop allows you to get some information about the application's performance. Using a threshold comparison algorithm, the MAPE loop is able to size the number of hosting containers.

The system is therefore something extensible and must be implemented according to the use cases.

I. Introduction

In devops work, automatic management systems (AMS) is a very important aspect of project development. In order to facilitate the development of this type of type of system, and avoid the situation when all the develop of a group create their own little system, it is important to create something extensible and understandable like a framework.

It is in this context that the project has been integrated. Develop a framework based on a MAPE architecture, allowing you to develop your own automatic management system for cloud computing resources.

A. What is a MAPE architecture?

MAPE is the acronym for **Monitoring Analysis Planning Execution**. MAPE loop is a concept introduce by IBM in 2005 to enable an optimized automatic management system for cloud computing resources.

Concretely the loop will automatically manage the system by connecting to the client application or system, monitor, analyze, plan and execute adaptive action.

In our case it will follow something like the following diagram:

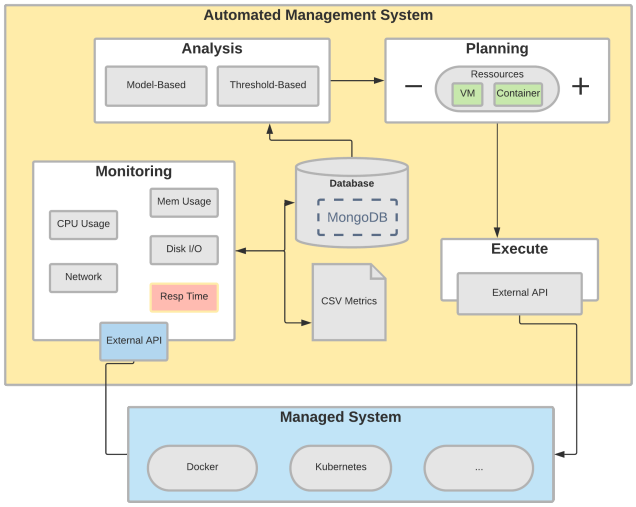


Fig. 1. EASE MAPE System architecture

B. Framework components

- **Monitoring**
  Monitoring needs to be connected to a client (like docker environment) and a database. It is supposed to get different data from the client like CPU, memory, disks I/O and network throughput of components, and store them into the NoSQL database in JSON document format. It can also get and store current date and for example the number of containers running on the client.
  The main point is that the monitoring needs to run independently of the other modules. Because

if something happens in other parts of the system the monitoring should continue to get information from the client.

- **Analysis**
  Analysis needs to be connected to the database because he will periodically read the new information stored. Because of the threshold-based used in our experiments, the analysis module should concentrate his analyze on the CPU and compare it with the threshold of our choice. It should return a list of information to the planning module. But the analysis can implement any type of model, corresponding to user needs.
- **Planning**
  Planning is a bit similar to the analysis but he only has to get information from analysis and plan the adaptive action. For example, add or remove containers in the case of Docker monitoring. And then send the decision to the execution module.
- **Execution**
  The execution module receives decisions from the planning module and execute the decision. He has to use the corresponding API and make a break in the MAPE loop until the system stability is reached.

II. Development process

A. Goal and constraint

1. The main goal of this project was to build a framework. Being a framework, we want the components of Monitoring, Analysis, Planning, Execution need to be extendable to allow for plugging services from external providers. More specifically, the framework components need to be abstract and simply define an interface to access the functionality of the individual component.

2. Secondly, we had to provide a concrete Docker implementation with Docker Compose scaling. This implementation will provide a ”guideline” for the development.

3. The second point was to provide a complete user manual. Because the system is a framework, the manual should contain a development guideline. But it must contain the tutorial for running the existing implementation.

B. Tools

Pycharm: PyCharm is a JetBrain Company IDE that allows a lot things in Python projects. This was a very powerful IDE to develop this project because it allows us to run different programs and debug them. It is also connected to GitHub in order to commit modifications. It provides auto completion, code refactorings, inspections, error highlighting and quick fixes.

MongoDB: MongoDB is a general purpose, document-based, distributed NoSQL database. MongoDB is a document database, which means it stores data in JSON-like documents. It is very useful when you need to store data with lots of information like a monitoring algorithm can do.

In our work it was very simple to use this database because it can be built locally with containers. The query in Python with Pymongo modules are very easy to understand.

Docker: Docker is the well know container services that provide a lot of functionality for running containers.This is a very large service used everywhere.

Then in our case we used docker to deploy test application of IoT simulation and database deployment. There is multiple way to interact with docker containers, but the remote API is a pretty good way because it provides JSON information.

The main point is that to have a deployment service, you cannot run containers independently. That why we used Docker Compose, to run multiple containers.

Docker Compose: As mentionned just below, Docker Compose is a tool for defining and running multi-container Docker applications.

”With Compose, you use a YAML file to configure your applications services. Then, with a single command, you create and start all the services from your configuration.” [1]

We used docker compose to deploy the test app and also to scale our application with the command line.

PyBuilder: ”PyBuilder is a software build tool written in pure Python which mainly targets Python applications. It is based on the concept of dependency based programming but also comes along with a powerful plugin mechanism that allows the construction of build life cycles similar to those known from other famous build tools like Apache Maven.” [2]

Pybluider was integrated in the project in order to installs dependencies automatically by using this build tool.

PowerAPI: PowerAPI is a tool under development to estimate the energy consumption of a machine. More specifically, the CPU usage.

We used PowerAPI to provide us with one more data to monitor. Energy consumption in Watt.

This can allow, for example, to provide an analysis based on something other than just use in

III. Docker client implementation

In order to provide a concrete version of the MAPE loop, we decided to implement a Docker client management system.

This concrete version will help developers to implement their own version of the loop by following the framework and get inspired by the existing one.

A. Docker remote API

Docker is a service which provides a lot of possibilities. As a lot of services he gives access to a Rest API. This API is available in multiple languages, and in our case the Python one is what we needed.

B. CPU threshold analysis

The threshold analysis method is the easiest way to manage the CPU utilization.

The thresholds are defined in the configuration file based on the needs of the user. The analysis algorithm is connected to the database where the monitoring data are stored. It will query the database every 10 seconde for example and compare the CPU of this last information with the thresholds. If something CPU reached one of the thresholds, the analysis method will call the planning part.

This planning part is nearly the same as the analysis part but it will decide how many containers to scale. Just after the decision is taken, the planning algorithm call the execution and then the execution execute the decision by calling bash command.

IV. Experiments

After developing the Docker concrete implementation, it was necessary to test it with a complete simulation of Docker environment with workload generators.

A. Using IoT sensors simulation

To test the application we used an IoT simulation. This simulation is launched with Docker Compose. When you run the Docker Compose command there are multiple services starting:

- Local MongoDB database
- HAproxy, proxy to load balance requests
- Web services container to receive requests
- IoT simulator to provide a scheduled number of requests per second

So the container to monitor is the web container because this is the one who receives the more request and he is likely to receive the most requests.

B. Results

So when the MAPE loop is connected to the docker environment where the IoT simulation is running, we could expect the autoscale working.

Here is an example of a simulation:

Fig. 2. Evolution of CPU utilization and a number of containers for the web service

As we can see, in orange we have the CPU utilization for the web service and the blue represent the number of containers.The threshold was fixed to 60% upper and 15% lower. It is not the most optimal thresholds but this is correct for the experiment.

The number of containers is not scaling every time the CPU is reaching one the threshold because the analysis takes a short section of time can calculate an average of the CPU utilization. That why sometime nothing appends.

Another example of thresholds autoscaling:

Fig. 3. Evolution of CPU utilization and a number of containers smoothed

This result seems much more satisfying. Indeed, the analysis algorithm has been modified in order to compare the last 5 CPU values in order not to cause a bounce. This results in a much more stable container auto scale.

V. Conclusion

This work has therefore led to several things. First of all, the idea of providing an extensible framework to be reused was reached. Indeed, a guideline has been defined in order to have a basis in the development of an automatic management system.

Secondly, the objective of providing an implementation example by following the framework was also achieved. There is therefore an implementation for the Docker environment that can be used as a guide.

Then, since the project will have to be further developed, the documentation provided had to allow this. This documentation therefore includes a tutorial and a user manual to build your own version of the MAPE loop.

In the future the objective is to continue to improve the framework, and to provide new versions of automatic management via different analysis algorithms, for example.

Acknowledgment

Thank you to Marios Fokaefs, teacher researchers at Polytechnique Montreal, for bringing the idea for this project to us. To have supervised, reviewed and ensured the credibility of the work provided.

Thanks to MohammadReza Rasol, for collaborating in the design and testing of the framework.

References

[1] https://docs.docker.com/compose/
[2] http://pybuilder.github.io/

Appendix A
User manual

EASE MAPE System

User manual

version : 0.2

author : Valentin LAQUIT

date : 2019/11/5

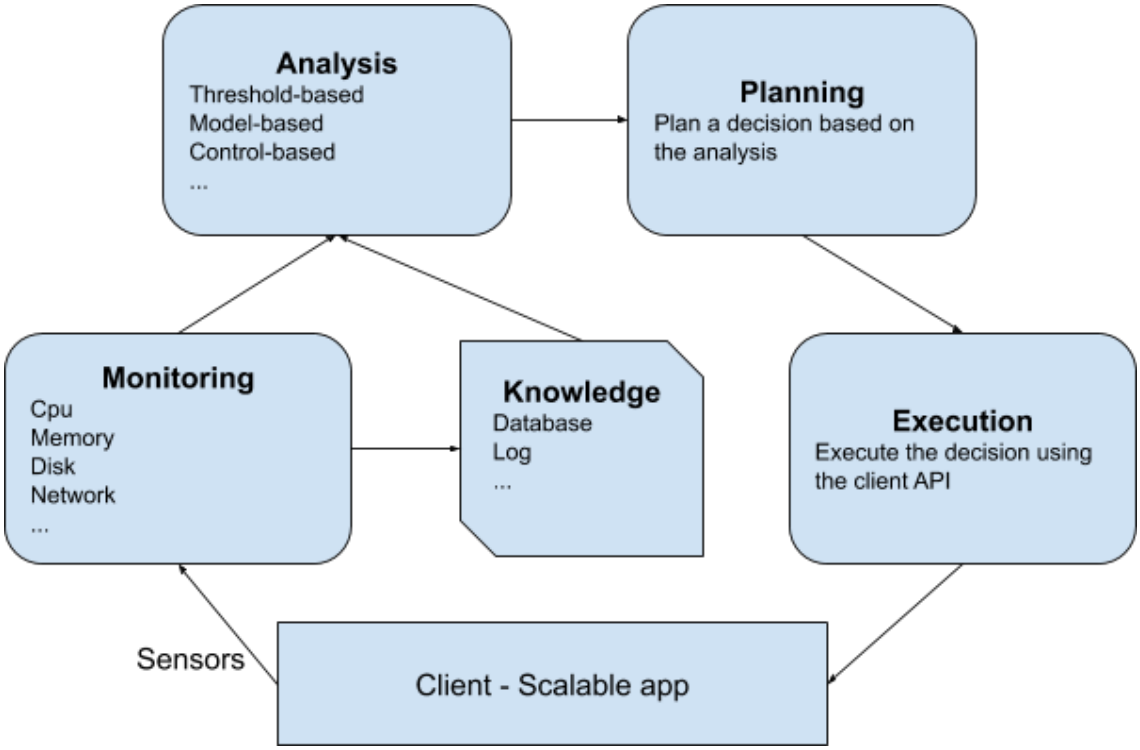
TABLES OF CONTENTS

INTRODUCTION

The framework is developed in Python 3.6

Only Docker deployment is available in the system for the moment, and it only can be used with Mongodb NoSQL database. But you can develop your own deployment for custom clients.

What is a MAPE architecture ?



Framework components

1. **Monitoring**

Monitoring needs to be connected to a docker client (like docker environment) and a database. It is supposed to get different data from the client like CPU %, memory %, disk I/O and network throughput of components, and store them into the NoSQL database in JSON document format. It can also get and store current date and for example the number of containers running on the client.

T**he main point is that the monitoring needs to run independently of the other modules.** Because if something happens in other parts of the system the monitoring should continue to get information from the client.

2. **Analysis** *(Threshold analysis)*

Analysis needs to be connected to the database because he will periodically read the new information stored. Because of the threshold-based, the analysis module should concentrate his analyze on the CPU and compare it with the threshold of our choice. It should return a list of information to the planning module.
Example :

- return 1 if CPU > upper_threshold
- return 2 if CPU < lower_threshold
- return 0 in other case

3. **Planning**

Planning is a bit similar to the analysis but he only has to get information from analysis and plan the adaptive action. For example, add or remove containers in the case of Docker monitoring. And then send the decision to the execution module.

4. **Execution**

The execution module receives decisions from the planning module and execute the decision. He has to use the corresponding API and make a break in the MAPE loop until the system stability is reached.

Analysis, Planning and Execution need to run together.

PROJECT DESCRIPTIONS

Project hierarchy :

```
EASE-MAPE-System
├──── build.py
├──── .env
├──── requirements.txt
├──── db
│     └──── docker-compose.yml
├──── mape
│     ├──── dockercompose_autoscale.py
│     ├──── analysis
│     │     ├──── analysis.py
│     │     └──── docker_threshold_analysis.py
│     ├──── execution
│     │     ├──── docker_execution.py
│     │     └──── execution.py
│     ├──── monitoring
│     │     ├──── docker_monitoring.py
│     │     ├──── monitoring.py
│     └──── planning
│           ├──── planning.py
│           └──── threshold_planning.py
└──── scripts
      ├──── autoscale.sh
      ├──── db.sh
      ├──── docker_monitoring.sh
      ├──── powerapi_monitoring.sh
      └──── powerapi.sh
```

⚠ *This is not the complete list of files and folders in the GitHub repository, but these are the important and necessary files for the project to work.*

All requirements are indicated in requirement.txt. Before you run anything, you should install all requirements with `pip3 install` or `pip install` command*.
** Using of PyBuilder build tool is in developpement.*

Files descriptions

`/Build.py`
This file is the PyBuilder* setup file. You should use it for installing dependencies.

`/.env`
This text file set some environment variable. It can be used as a configuration file for maybe, the database URI, path or threshold of your choice. (This file is hidden, do `ls -a` to see it)

`/requirements.txt`
All requirements are indicated in requirement.txt. Before you run anything, you should install all requirements with `pip3 install` or `pip install` command*. Or use the PyBuilder module and use `pyb` to install dependencies.

`/db/docker-compose.yml`
It is a docker-compose file to create a local mongodb database (optional)

`/mape/dockercompose_autoscale.py`
This python file is the main program to run Analysis, Planning and Execution module. It instantiate 3 objects and begins a while True loop to execute the analysis.

`/mape/monitoring/monitoring.py`
This file contains one abstract class **Monitoring.** In this class you have the `__init__()` method that get parameter for the client to monitor and the database client. And you have a `database_insertion()` method that you can call in your program to insert data into the database.

`/mape/monitoring/docker_monitoring.py`
This is the concrete implementation of the monitoring module for Docker Client. You have a class with several methods of calculation. The most important is that this class use Docker remote API to get information about containers running in the client environment.

`/mape/analysis/analysis.py`
This file contains one abstract class **Analysis.** There is several method, including one abstractmethod `update`. Notice that you have *Observer relation.

`/mape/analysis/docker_threshold_analysis.py`
This is the concrete implementation of the analysis module for threshold based analysis. There is an *Observer relation with Planning. The goal is to call Planning method when the analysis detect something.

/mape/planning/planning.py
This file contains one abstract class **Planning.** There is several method, including one abstractmethod `update`. Notice that you have *Observer relation.

/mape/planning/threshold_planning.py
This is the concrete implementation of the planning module. There an *Observer relation with Analysis and Execution. When he analysis call Planning method, the planning will plan some decision and call the Execution module.

/mape/execution/execution.py
This file contains one abstract class **Planning.** There is several method, including one abstractmethod `update`.

/mape/execution/docker_execution.py
The **DockerExecution** class use `os.system()` method to send bash command to the system to scale up and down the number of containers. There is an *Observer relation with Planning.

/scripts/autoscale.sh
This bash script run the autoscaling.

/scripts/db.sh
This bash script create a MongoDB database and a database manager.

/scripts/docker_monitoring.sh
This bash script run the docker monitoring.

** Using of PyBuilder build tool is in developpement.*

/scripts/powerapi_monitoring.sh
This bash script run the powerapi monitoring.

/scripts/powerapi.sh
This bash script deploy powerapi sensor.

HOW TO RUN EXISTING MAPE LOOP

Docker client

Before starting :

You need a sample application to test this system. It requires a web application and a workload generator to simulate a change in workload.

- Install docker :

Official docker documentation: https://docs.docker.com/

- Make sure you have a MongoDB database running and the good URI mentioned in the program.

Official mongoDB manual: https://docs.mongodb.com/manual/

- Make sure you have mentioned the path where the *docker-compose.yml* is located.

Install docker compose: https://docs.docker.com/compose/install/

- For the threshold analysis, please choose your thresholds in the .env file.

Install PyBuilder : http://pybuilder.github.io/

- You should read about the build.py file and the command pyb to use PyBuilder.

Get started :

1. Configure your **.env** file (It is an hidden file so use a terminal to edit it)
2. Install all requirements from requirements.txt. (via: Pybuilder, .sh file or pip command)
3. Run the **docker_monitoring.sh** file into a terminal with
   `sh docker_monitoring.sh`
   a. You should see "*Monitoring V2 \n Using: Docker remote API*"
   b. Wait few seconds until the monitoring is done
   c. You should see "Time to insert into the database *… "*
      (If not the monitoring will return an error.)
   d. Wait … sec and it will redo until you stop it

As mentioned previously monitoring run independently than the other modules of the MAPE loop so you can run only the **docker_monitoring.py** file to just monitor your client.

4. Run the **auoscale.sh** into another terminal
   a. You should see "*Analyse : …*"
   b. After some informations will be printed and the planning decision.
      (If nothing is printed, this mean that the analysis does not detects something wrong)

c. And in case of a scale the docker-compose prompt will appears
d. Wait … sec and it will redo until you stop

Warning : You cannot run the autoscale if there is no data into the database. Wait for the first insert before launch this -APE loop.

Debugging :

- If the docker client is on your computer, you can launch docker stats command to have a stream of your current environment and see if changes well done.
- Maybe you can use MongoDB Charts for plotting some data.

You can follow this tutorial :
https://www.mongodb.com/blog/post/visualizing-your-data-with-mongodb-charts

BUILD YOUR OWN IMPLEMENTATION

It is recommended to use PyCharm or an equivalent IDE.

Monitoring

If you want to implement a monitoring for custom clients, you have to create a subclass of the abstract class **Monitoring** into a new file in the same folder. The next step is to implement all method you need.
The `__init__()` method initiate the client with the client_to_monitor parameter and the MongoDB client with the mongo_client parameter. It also initiate lot of variable that you can use if you need for your monitoring's needs.
THe idea is to connect the algorithm to the client with an API. For example a REST API. The best way to implement methods is to give them a JSON or dictionary parameter taken from the client's API.
Remember that if you want our **MAPE** loop works you must regroup all your data into a dictionary and store it into the database. Because the analysis will always read it in the same way.

Analysis

The threshold analysis is quite simple to implement because you only need to read the database where you stored the cpu data and compare them to the thresholds. In order to do that you need to enter the same mongo client as the monitoring and get the last update data from it. The best way to connect the analysis to the planning is to establish an *Observer relation with the Planning. So the Analysis will call the Planning when it is necessary.

Planning

The `__init__()` method of the **Planning** class needs to observe the Analysis and plan decision. The decision in our case is the information which will be send to the execution module.

Execution

The `__init__()` method of the **Execution** class needs to observe the Analysis, in order to get the decision and execute it. In other words, the execution will scale up or done if needed for the client. The only method to implement is the `scale()` method which is the link between the **MAPE** loop and the client.

Main

In the **/mape/main.py** file you will instantiate all objects you need like the analysis, planning and execution and you will call all method you need from those three classes. In order to run it continuously, you can use a `while True` loop.

Dependencies

To manage dependencies, you must list all the necessary packages in the requirements.txt file.

ERROR YOU MAY ENCOUNTER

- Maybe you will have trouble installing dependencies with Pybuilder. This may be due to an access or python version problem.

Appendix

Observer relation