Intelligent Orchestration of Containerized Applications in Cloud Infrastructures

Julie Farkouh, Audrey Ahoukeng Donwoung, Nazim Agoulmine

Intelligent Orchestration of Containerized Applications in a Cloud Infrastructure (Short Paper)

Julie Farkouh[1], Audrey Ahoukeng Donwoung[1], and Nazim Agoulmine[1]

Paris Saclay University - Université of Évry Val d'Essonne - IBISC Laboratory
julie.farkouh@etd.univ-evry.fr, audrey.ahoukeng-donwoung@etd.univ-evry.fr,
nazim.agoulmine@univ-evry.fr

1 Introduction

Resource contention and performance interference between co-located applications can be a serious problem in a multi-tenant infrastructure. This problem is difficult to address by existing orchestrators and may cause applications performance downgrade, extra maintenance costs, as well as service-level agreement (SLA) violations. Therefore, a refined and robust container orchestration system is the key factor in controlling overall resource utilization, energy efficiency, and application performance. However the highly dynamic nature of modern applications, diverse features of cloud workloads and environments to take into account considerably raise the complexity of orchestration mechanisms[3]. One particular orchestration problem is the deployment of containers in appropriate hosts with the right required resources as resources play a major role in the containerized applications performance. This problem is not fully taken into account by modern orchestrator since the type of application and the deployment mode are not known to the orchestrators and therefore it is not able to determine the appropriate required resources.

Deploying Applications or Websites that are in production mode is an important concern for many companies willing to externalize their Applications and services in the Cloud Computing. Since allocated resources play a major role in their performance and the end-users satisfactions, determining the most appropriate level of resources is of paramount importance. Today mainstream deployment is using containers and orchestrator such as Kubernetes. The concerns is the difficulty to identify the required level of resources to allocate and maintain to these applications during their life cycle.

To overcome this problem, we propose to artificial intelligence and more precisely Decision Tree to predict the applications types and modes to take into account this information during orchestration. We adopt in this work a use case where the deployed applications are containerized Web Applications (CWA) and containerized DataBase Applications (CDA) which could be in test mode or production mode. The orchestrator that receives this information can deploy therefore deploy them in the most appropriate nodes with the most appropriate resources (CPU/Memory/Network) and achieve better resource allocation decision and performances.

Some related works on this issues are presented here. In [1], Ishak et al. proposed an intelligent scheduler in Kubernetes environment using ML based on the execution prediction of the applications and the nature of the processed data in order to better select the appropriate device, CPU or GPU. In [2] , Jose et al. proposed Network-Aware scheduler for container-based applications in Smart City deployments that added the parameters of network quality to the calculation score of the Kubernetes scheduler in order to take it into account when selecting the appropriate deployement node.

2 Problem Statement

Kubernetes which is one of the most popular orchestrating tools for containers. It is an open source scheduling platform that automates Linux container operations: Distribution, Replication, Load-balancing, Availability, Higher-level interfaces to composition features and Docker which is an open source containerization platform - with standardized executable components that combines application source code with the operating system(OS), libraries and dependencies that are required to run that code in any environment.

Kubernetes is doing a great job in scheduling, deploying and provisioning containers based on the user-requested RAM and CPU for their containers in proportion to the free RAM and CPU available in the infrastructure. Unfortunately, those are not only the resources that should be taken into account when deploying containers as other resources may play an important role in the application performance based on what it really needs such as Network resources and Disk Input and Output.

Eventually, these addition requirements should be taken into account during orchestration and appropriate related resources should be computed based on the type of application container service and the context in which they are deployed. If applications are deployed in production mode, more attention to all resources needs to be addressed to cope with the potentially large amount of service requests. Whereas deploying an application in a development mode for a single user developer is not that important as there is no online demands and as a result, high performance requirements are not necessary. Unfortunately, Kubernetes Scheduler that it is responsible for scheduling and deploying the containers into the nodes does not take into account these parameters (e.g. Network Quality , I/O ..) and the mode in which applications are deployed (development, stage, production). As a result, the deployment solution could be inappropriate resulting in bad performances.

3 Proposed Solution

As a solution, we introduce a model which uses ML to automatically predict the type of containerized applications to execute and their corresponding mode for deployment. Once predicted, the solution triggers the appropriate deployment of the applications in the appropriate nodes of the containerized infrastructure using an orchestrator (in our case Kubernetes).

In our model, we have addressed two types of applications: Containerised Web-based Applications (CWA) and Containerised Database Applications (CDA) in production mode. The rational behind the selection of these two applications is that there are the most commonly deployed industry grade applications.

Therefore, the ML Model will predict the Container Type and Mode and based on these predicted values, the model will assign the container to the appropriate computing node. This means that if the ML Model predicts that the container is for a Web application Service in a production mode, it will automatically assign and deploy it into one of the nodes with the highest network speed interface and if the predicted value is database container in production mode, then it will assign it to one of the nodes with the highest available RAM capacity.

During the prediction phase, the model reads the desired containers configurations and

extracts the features that correspond to their type and deployment mode. These features are then introduced in a previously trained ML Model that uses classification learning to classify the containers based on their type and mode.

Regarding the deployment phase, the proposed solution first searches for the nodes with the highest network interface speed from all the available nodes in its cluster to host the Web-application containers (since in production mode, this feature is one of the most important). Similarly, it searches for the node with the highest available RAM capacity to host to the Database Server containers.

- **Container Image** and **Port** that the container will use – these two features are the most important indicators to predict the application type and the ones commonly used. The used images are the ones available in the docker hub - official website of the dockers images - and widely known for web services like Apache, Nginx or IIS and the relevant ports would be also the generally the most used one: 80, 443, 8080 . For Data Base Service, the name of the images that are the most used are Mysql, Oracle or Postgress Sql and corresponding ports are usually 3306, 1521.

- **Name** of the Container, as in IT, we should always use as a best practices in naming process, names that indicate clearly the service to help maintenance such as, logging, debugging, auditing, etc. For the web application, it could vary between the following names "Websites", "Front ends", "Webs". Similarly, for the Database application, names could be "Database", "MyDataBase", "DB" in repeated way like for web services.

  RAM we assume in our work the RAM would start from 256/512 M bytes of RAM for the development mode to up to 4 G bytes of RAM for applications in the production mode..

These are the features we took into account to train our ML Model to classify them into two types of outputs (Web or database) and two predicted environment mode outputs (Production or Development).

4 Implementation and Experimentation

For our Deployment, we first trained our model in Matlab Classification App – Analysis Trees on simple Dataset of the Web and Database services properties previously introduced based on containers' image, Port, Name indicating reserved RAM. We then exported our Model into Matlab Production Server in order to use it for production. The provided interface is a Rest API.
Next, we used Kubernetes configured into one master node and two Worker nodes with different RAM Capacities and Network Interface Speed.

Worker Node1 = 8 Gg RAM , 10,000 Mb/s Traffic Speed
Worker Node2 = 10 Gg RAM, 1000 Mb/s Traffic Speed

After training the ML Mode, the sequence of performed tasks are the following:

1. Our Model first finds the node wirh the highest speed interface which is worker-node 1 and label it "prodweb" as for Webservices in Production.

2. It also finds the node with the highest RAM Capacity Node which is worker-node 2 and labels it "proddb" for DataBase Production

3. Then it reads the configuration file of the pod configuration containing the container specifications (YAML File) and extracts the features of the containers and send them via a HTTP Request API to our trained ML Model, The predicted values of the ML Model would be one of three cases:

   - **Containerised Web-based Applications (CWA)** in **Production Mode**.
   - **Containerised Database Applications (CDA)** in **Production Mode**.
   - **CWA** or **CDA** in **Development Mode** or Other application types in **Production / Development** Mode but we didn't follow-up thi czse in the deployment case we did not handle the deployment, just the first two first.

4. The pod configuration file is then updated to launch the the deployment in the corresponding node.

We tested the accuracy of our ML Model. Since our data was simple and the output is only composed of three possible values, the model achieved 93% accuracy.

We tested also the robustness of the prediction with general values in the dataset. For example, we changed the names to some unrelated strings (that do not contain the pattern) to the application while keeping the other values as it in the datase (e.g. the Ram to a lower/higher values than from the ones in our dataset , Changing two features like the Name and Image. Despite these changes the ML was still able to predict correctly the type and mode, thanks to the Decision Tree (DT) approach.

5 Limitations and Future Works

In our work we presented a simple Model which is able to predict the containerized application type and mode based on features contained in the configuration file. Based on this prediction, the scheduler aims to deploy these applications in the most appropriate nodes in the cluster. The prediction worked well with the type and mode, but we aim to extend it to address other type of services like Domain Name Service (DNS), Mail Services, Virtual Private Networks services (VPN), etc. However, this work has several limitations we plan to overcome. Our scheduler state is considered in an initial status with no consideration for other potentially running containers competing for the same computing and networking resources. Indeed, the solution identifies the nodes with the highest network interface and RAM capacity without considering the current bandwidth or RAM utilization. A possible approach would be to combining our model with the native Kubernetes scheduler score tool that it used in node selection i.e. using the result as factor to computing the node selection in a pod, similarly to the work presented in Jose et al.[2]. we envision also to expand our dataset to predict several more containerized applications types and have more real accuracy results for better orchestration mechanisms based on the container application type.

6 Conclusion

In this work, we presented a ML based scheduler which relays on "Type" and "Mode" Awareness of containerized applications for better resources allocation and node selections decisions. With

this solution, it is possible to detect the nature of applications: Web Containerized Applications and Data Base Containerized Applications and their production mode (test, production). This information is used to improve the allocation of resources (Network/RAM). We used two popular open-source projects that are Docker and Kubernetes, to validate our model which helps to identify additional needed resources beyond RAM and CPU (that are used in the original Kubernetes scheduler). We implemented a poc of our model a separate scheduler. Our future works is to integrated our model with Kubernetes scheduler and allow is to be aware of more information to perform a better scheduling and deployment .

References

[1] Ishak Harichane, Sid Ahmed Makhlouf, and Ghalem Belalem. A proposal of kubernetes scheduler using machine-learning on cpu/gpu cluster. In *Computer Science On-line Conference*, pages 567–580. Springer, 2020.

[2] Jose Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards network-aware resource provisioning in kubernetes for fog computing applications. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 351–359. IEEE, 2019.

[3] Zhiheng Zhong, Minxian Xu, Maria Alejandra Rodriguez, Chengzhong Xu, and Rajkumar Buyya. Machine learning-based orchestration of containers: A taxonomy and future directions. *arXiv preprint arXiv:2106.12739*, 2021.