A PLATFORM-AGNOSTIC DEPLOYMENT STRATEGY LEVERAGING REAL-TIME HOST METRICS FOR OPTIMIZED RESOURCE UTILIZATION

Teodor Petrović*,
[0009-0008-7186-2552]
Aleksa Vidaković,
[0009-0005-3527-011X]
Petar Kresoja,
[0009-0008-3343-1540]
Nikola Savanović,
[0000-0001-9670-7374]
Saša Adamović
[0000-0002-2875-685X]

Singidunum University,
Belgrade, Srebia

Abstract:

Many environments for application development face challenges due to inefficient resource utilization and a high degree of vendor lock-in when using cloud services, which limits flexibility and increases user costs. Traditional methods often rely on fixed infrastructures that can lead to host misconfigurations and suboptimal resource usage. In response to these issues, we propose a platform-agnostic application deployment tool that harnesses real-time host metrics to optimize resource allocation and enhance deployment efficiency. The solution features a robust architecture with a client interface, an orchestrator service, a metrics collection service, a dedicated management service, and a proxy service. The platform dynamically configures individual host environments based on specific application requirements by leveraging user-configurable tags and Ansible automation scripts. It further employs host performance metrics, such as CPU and RAM usage and network throughput collected via the metrics collection service, to intelligently select the least utilized hosts for application deployment. Our automated deployment strategy, based on host resource utilization, helps us avoid the traditional issues of having to configure each host manually and not having to be vendor-locked to a specific cloud provider, paving the way for a more flexible and efficient resource utilization.

Keywords:

Platform-agnostic Deployment, Ansible, Metrics-based Resource Allocation, Automated Configuration.

INTRODUCTION

Deploying applications across multiple hosts—whether virtual machines or bare-metal systems—remains a formidable challenge in modern infrastructures. Prior research indicates that inefficient resource utilization is a persistent problem [1]. In addition, containerized environments often suffer from host misconfigurations that adversely affect performance [2]. Traditional deployment methods that rely on manual setup or basic automation scripts lead to further issues such as suboptimal resource usage, insufficient monitoring, and security inconsistencies [3]. Independent studies have shown that these limitations can result in frequent deployment failures [4]. While cloud providers offer a standardized and controlled platform for deploying applications, thus removing the previously mentioned problems, the associated costs and vendor lock-in possibility provide considerable issues with this approach [5].

Correspondence:
Teodor Petrović

e-mail:
tpetrovic@singidunum.ac.rs

DOI: 10.15308/Sinteza-2025-71-77

To address these challenges, we propose a dynamic, platform-agnostic deployment tool that streamlines application management workflows using available resources. This tool enables users to register their host machines and specify resource and service requirements via an intuitive tagging mechanism. By integrating automated configuration management with continuous, real-time monitoring of host performance, the tool minimizes manual errors and resource wastage while dynamically selecting the optimal host based on current utilization. In doing so, it effectively mitigates issues such as host misconfigurations and vendor lock-in, offering a flexible, scalable, and secure deployment framework.

2. RELATED CONTENT

Previous studies in this field have highlighted significant challenges concerning deploying and managing host resources. Tools such as Kubernetes and Docker Swarm provide robust container orchestration capabilities; however, they typically require complex configuration patterns. Research has demonstrated that these platforms often lack integrated real-time host resource metrics [6] and may not adapt well to fluctuating workloads [7]. Similarly, while Ansible simplifies host configuration through its agentless design, it does not inherently support dynamic, resource-based decision-making, which limits its adaptability in rapidly changing environments. While these platforms have advanced the way containerized applications are managed, their static configuration models sometimes fail to adapt to dynamic resource fluctuations and evolving workload demands in situations where businesses rely on their infrastructure (private or hybrid cloud). This rigidity can lead to inefficient resource allocation and may result in unexpected performance bottlenecks per host machine and the organization as a whole.

Ansible, known for its simplicity and agentless operation, has become a popular choice for host configuration management. Despite its ease of use, Ansible traditionally does not incorporate dynamic, resource-based decision-making, essential for optimizing deployments in highly variable environments. As a result, many deployment scenarios still require manual reconfiguration of the playbook files, which can hinder the system's ability to adapt to evolving infrastructure conditions.

Monitoring systems like Prometheus [8] and Elastic Stack (Kibana) [9] have been instrumental in offering deep insights into application and host performance. They provide detailed graphs and metrics for visualization and analytics capabilities that allow administrators to track host machines with critical performance statuses over time. However, these systems generally operate in isolation from the deployment process, creating a disconnect between the host monitoring and its automated configuration. To bridge this gap, recent studies have explored integrating real-time performance metrics directly into deployment workflows.

In contrast to these existing solutions, our approach merges real-time monitoring with automated deployment, as demonstrated in recent studies [10]. Additionally, the work by Vankayalapati et al. (2022) [10] supports the concept of predictive scaling through integrated monitoring and orchestration.

Our approach builds upon this research by seamlessly integrating real-time host performance metrics with a custom automated deployment process. By doing so, we significantly enhance the decision-making process regarding host selection and per-host resource utilization, ensuring that deployments are not only automated but also optimized for current infrastructure conditions. This integration addresses the critical gaps left by traditional tools, offering a more adaptive, efficient, secure, and reliable deployment solution. Unlike these existing solutions, which typically rely on static configurations and operate in isolation from real-time monitoring, our approach leverages dynamic metrics to make informed, adaptive deployment decisions, thereby enhancing resource utilization and system responsiveness.

3. METHODOLOGY

Our deployment solution tool is built upon a comprehensive system that integrates several interconnected services to deliver an end-to-end application deployment process. The approach begins with an intuitive user interface that allows system administrators to register host systems (BareMetal or VM machines). During this registration process, each host is assigned descriptive tags that indicate the specific services and configurations required from it, effectively summarizing its intended role and capabilities (Docker, Kubernetes, ingress, egress, storage). This tagging mechanism lays the groundwork for an automated deployment process that tailors to specific hosts, preparing them to meet diverse deployment needs.

Upon host registration, regardless of assigned tags, a metric collection service is installed as part of the default configuration. Simultaneously, each host is automatically enrolled in our DNS system. Using DNS names instead of relying on static IP addresses provides a more robust and flexible way to manage our deployment infrastructure. Using DNS allows us to update host locations or configurations without requiring changes to specific applications or service configurations, improving resilience and reducing management overhead.

Once these foundational configurations are complete, the backend service dynamically generates configuration scripts tailored to each host's designated role or roles, as determined by their tags. These scripts automate the subsequent setup process, significantly reducing manual intervention and the risk of human error or misconfiguration. By leveraging these dynamically generated scripts alongside the standardized metric collection and DNS enrolment, we ensure that each host is optimally and securely configured according to its unique tag profile.

Simultaneously, a metrics collection mechanism continuously gathers real-time performance data from every registered host. Key performance metrics—such as CPU, RAM usage, and network activity—are monitored without interruption. This continuous data flow is critical, as it maintains an accurate, up-to-date view of all host resource utilization across the entire system infrastructure. The collected metrics serve as the foundation for the system's decision-making process.

At the heart of the system lies the management node, which is responsible for orchestrating the deployment process and making calculated decisions based on the latest performance data. When a deployment request is initiated, the management node retrieves the current metrics from the log collection service, filters, and analyzes them to determine the optimal host for a specific application deployment. It employs a decision-making process that calculates the average resource utilization across available hosts, in combination with the application's specific requirements, ultimately selecting the host with the lowest load to ensure efficient deployment and optimal application performance.

The management node not only handles the host selection process but also oversees the entire orchestration process, from initiating host configuration to finalizing application deployment. It acts as a central orchestrator, ensuring that each step of the deployment process from start to finish is executed in the correct sequence and that all system components operate in complete harmony. This coordination is accomplished by pre-defined decision criteria combined with real-time analytics, allowing the system to adapt dynamically to fluctuating resource demands and varying host conditions.

In summary, our methodology integrates automated configuration, continuous real-time monitoring, and intelligent host selection to create a robust deployment process. This integrated approach (see Table 1) not only enhances system reliability and scalability but also directly addresses challenges such as resource inefficiencies and configuration inconsistencies.

Table 1. Service components

Service Name	Function	Technology Stack
Client Interface	Provides an intuitive UI for host registration and configuration	React, JavaScript, HTML/CSS
Orchestrator Service	Manages application deployment orchestration and host selection	Nest.js.
Metrics Collection Service	Gathers real-time performance metrics from hosts	Elasticsearch, Kibana, Logstash, Metricbeat.
Management Service	Automates host configuration and deployment using generated Ansible scripts	Python Flask App.
DNS service	Manages domain name resolution and dynamic DNS records for application access	PowerDNS
Proxy Service	Configures reverse proxy routes for secure application access	Apache HTTP Server

4. IMPLEMENTATION

The system's implementation is designed to automate application deployment by integrating several interconnected components that work in synergy. At its core, the deployment process begins with a default build configuration that defines the default parameters of an application deployment. This build configuration contains the application's git repository URL, branch name, container's port mappings, volume mounts, network assignments, resource limits, restart policies, deployment order, and most importantly, its tags (CPU intensive, RAM intensive, ingress intensive, egress intensive, storage cold, storage hot, etc.). It serves as a deployment blueprint, ensuring that each application is deployed in a consistent and isolated environment while meeting its specific operational requirements.

4.1. DEFAULT CONFIGURATION EXAMPLE

Listing 1 shows an example of a JSON configuration file that defines the deployment parameters for an application. For instance, consider the following build configuration:

As seen in Listing 1, the configuration parameters are explicitly defined to support automated deployment. Based on this configuration, the management node connects to the specified host system using the SSH key and username, first building the required image for the application. Container images are named using the Git project name, with the tag being the branch name and last commit hash. However, we must note that the Docker command can create previously non-existent volumes if specified with the "-v" option, but there is no command to automatically create new networks. Therefore, if the requested network does not exist, it must be first explicitly created before running the container. Listing 2 represents the Docker run commands automatically generated by the management node based on the configuration provided in Listing 1. Outbound ports are dynamically chosen on the host system, starting from port 10000, based on the host's available free ports. The generated Docker command examples look like this:

In this example, container ports 8080 and 8081 are mapped to host ports 10000 and 10001, respectively. The command sets CPU and memory limits, mounts the specified volume, attaches the container to the primary network (if networks with the requested name do not exist, they will be created), and applies the restart policy.

```
{
  "repository_url": "https://github.com/deploy/web-app",
  "branch": "master"
  "ports": [8080, 8081],
  "tags": ["ingress", "docker"]
  "host": {
    "username": "teodor",
    "domain_name": "nebula-mng.masofinonebula.internal",
    "port": 22,
    "ssh_key_path": "/path/to/key/host_192.168.0.203_Nebula_mng",
    "ip": "192.168.0.203"
  },
  "volumes": [
    { "volumen_name": "SharedData", "container_path": "/home/shared" }
  ],
  "networks": [
    { "name": "sdn-network", "ip": "dhcp" },
    { "name": "internet", "ip": "172.66.0.10" }
  ],
  "resources": { "cpu_limit": "0.5", "memory_limit": "512m" },
  "restart_policy": "always",
  "deployment_order": 1
}
```

Listing 1. Application build config in JSON format

```
# Create the required Docker network
docker network create --internal sdn-network

# Deploy the container based on the configuration in Listing 1
docker run -d
  --name web-app-6 \
  --restart always \
  -p 10000:8080 \
  -p 10001:8081 \
  --cpus="0.5" \
  --memory= "512m" \
  -v SharedData:/home/shared \
  --network sdn-network \
  --network internet \
  --ip 172.66.0.10 \
  web-app-docker:main-b37a3cb
```

Listing 2. Generated Docker run command based on application build config

4.2. DECISION-MAKING ALGORITHM SPECIFICATION

The next step in the deployment process is the selection of the most suitable host for the proposed application. To achieve this, the management node queries and analyses real-time performance metric data from currently active hosts, which fit the application needs, to determine the optimal deployment target. The decision-making process consists of the following steps:

- **Data Collection** - Continuously gathering host performance metrics (e.g., CPU usage, memory consumption, network throughput) from all registered hosts. To account for differences in hardware capacity and application requirements, the raw metrics r_{cpu}, r_{ram}, r_{send} and $r_{receive}$ are normalized using a min–max normalization approach (see Equation 1). Specifically, for each metric, the normalized value is computed as:

$$Metric_{norm} = \frac{r_{metric}}{M_{metric}}$$

Equation 1. Normalization Equation

where M_{metric} is the maximum observed value for that metric.

- **Metric Aggregation** - For each host instance, we calculate a utilization score by combining the normalized metrics. Let CPU_{norm}, RA_{norm}, $Send_{norm}$, $Receive_{norm}$ denote the normalized values for CPU, RAM, send bytes, and received bytes, respectively. The aggregated score S is then calculated as a weighted average:

$$S = \frac{w_{cpu}.CPU_{norm} + w_{ram}.RAM_{norm} + w_{send}.Send_{norm} + w_{receive}.Receive_{norm}}{w_{cpu} + w_{ram} + w_{send} + w_{receive}}$$

Equation 2. Aggregated Score Equation

where w_{cpu}, w_{ram}, w_{send} and $w_{receive}$ are weights assigned to each metric. In our default configuration, equal weighting is assumed for the base algorithm state. Consequently, when w_{cpu}=w_{ram}=w_{send}=$w_{receive}$=1, the aggregated score simplifies to:

$$S = \frac{CPU_{norm} + RAM_{norm} + Send_{norm} + Receive_{norm}}{4}$$

Equation 3. Default Aggregated Score Equation

This calculation can be adjusted based on application-specific resource demands (e.g., increasing the weights for CPU and memory if the application is resource-intensive in these areas).

- **Host Ranking** - The aggregated scores of specific hosts that fit the application needs are compared. The host with the lowest overall score is identified, indicating the most readily available host that can handle the new deployment.
- **Decision Execution** - If the build config does not specify a target host, automatically select the host with the lowest utilization score. In the case of a tie or other scenarios (such as deployment order or historical performance data) are defined, we can apply custom decision rules to finalize host selection.

This algorithm ensures that applications are deployed on the most efficient host available at any given moment, thereby optimizing resource utilization and maintaining application stability.

5. EXPERIMENTAL SETUP

The evaluation of the proposed deployment tool was carried out in a controlled environment. The proxy server was hosted on an Ubuntu Linux VPS equipped with four cores and 8 GB of RAM. Each host machine used for deployment had four cores and 8 GB of RAM. The management and log collection machines were provisioned with four cores and 16 GB of RAM, while the backend service ran on a machine with four cores and 8 GB of RAM; all these machines operated on Ubuntu Linux. In addition, all virtual machines—except for the proxy, which is hosted on a VPS—were managed by a hypervisor running VMware Workstation. The hypervisor featured dual Intel® Xeon® E5-2630 v4 processors (2.20 GHz, 10 cores, 10 logical processors each) and 226 GB of RAM. The proxy is connected to the private infrastructure via a WireGuard VPN, ensuring secure and reliable communication between the machines.

6. RESULTS

The deployment tool was evaluated in a controlled environment involving applications with varying resource requirements. We tested the decision-making algorithm on four host instances to assess the efficiency of our metrics-driven host selection process. As the previously defined metrics, we generated aggregated scores based on normalized CPU usage, RAM usage, send bytes, and received bytes. Moreover, our decision-making process is designed in such a way that it can adjust the weights applied to these metrics based on application configuration tags. For instance, if an application specifies high CPU and RAM requirements, the algorithm increases the weighting factors for these metrics, ensuring that hosts with lower resource utilization in these areas are preferred. Similarly, significant ingress and egress network requirements can lead to higher send and received byte weights. Table 2 summarizes the normalized performance metrics and aggregated scores obtained from our experiments on four host instances.

Based on the results in Table 2, the decision-making algorithm selected abyss-mng as the best deployment target because it scored the lowest score (0.25). This outcome demonstrates that our metrics-driven approach effectively reduces resource wastage and enhances resource utilization, all while aligning with application-specific resource demands.

7. CONCLUSION

The developed platform-agnostic deployment tool successfully integrates automated host configuration, metrics-driven resource allocation, and user-driven deployment customization. Based on the experimental results presented in Table 2, our evaluation shows that the system effectively selects the best host for deployment by considering host performance metrics—including CPU, RAM memory, send bytes, and received bytes—and dynamically adjusting to application-specific requirements. In conclusion, we have successfully achieved our objectives by addressing traditional deployment challenges such as inefficient resource utilization and misconfiguration. The system not only minimizes resource wastage but also significantly enhances deployment stability and scalability, proving its effectiveness in real-world scenarios. Future work will focus on integrating machine learning techniques to further refine the adaptive weighting mechanism and on scaling the system to handle a larger number of hosts under varying workload conditions.

Table 2. Normalized Performance Metrics and Aggregated Scores for Host Selection

Host	CPU Usage	Memory Usage	Send Bytes	Received Bytes	Score
madman-mng	0.50	0.52	0.55	0.56	0.53
abyss-mng	0.28	0.31	0.22	0.18	0.25
nexus-mng	0.40	0.38	0.35	0.33	0.37
nebula -mng	0.42	0.37	0.36	0.35	0.38

REFERENCES

[1] M. Narasimhulu, D. V. Mounika, P. Varshini, A. K. and T. R. K. Rao, "Investigating the Impact of Containerization on the Deployment Process in DevOps," *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*, pp. 679-685, 2023, https://doi.org/10.1109/ICECAA58104.2023.10212240

[2] M. Abhishek, D. Rao and K. Subrahmanyam, "Framework to Deploy Containers using Kubernetes and CI/CD Pipeline," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 239, 2022, http://dx.doi.org/10.14569/IJACSA.2022.0130460

[3] W. M. C. J. T. Kithulwatta, W. U. Wickramaarachchi, K. P. N. Jayasena, B. T. G. S. Kumara and R. M. K. T. Rathnayaka, "Adoption of Docker Containers as an Infrastructure for Deploying Software Applications: A Review," *Advances on Smart and Soft Computing*, p. 247–259, 29 June 2021, https://doi.org/10.1007/978-981-16-5559-3_21

[4] M. Dînga, L. Giamattei, A. Guerriero, R. Pietrantuono, S. Russo, I. Malavolta, T. Islam, M. Dînga, A. Koziolek, S. Singh, M. Armbruster, J.-M. Gutierrez-Martinez, S. Caro-Alvaro, D. Rodriguez, S. Weber, E. F. Vogelin and F. S. Panojo, "Monitoring tools for DevOps and microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 208, no. 0164-1212, p. 111906, 16 December 2024, https://doi.org/10.1016/j.jss.2023.111906

[5] D. Mo, R. Cordingly, D. Chinn and W. Lloyd, "Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction," *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 193-199, 2023, https://doi.org/10.1109/CloudCom59040.2023.00040

[6] "Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks," *Journal of Cloud Computing*, vol. 9, no. 49, 2020, https://doi.org/10.1186/s13677-020-00194-7

[7] S. Son and Y. Kwon, "Performance of ELK stack and commercial system in security log analysis," *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*, pp. 1-6, 2017, http://dx.doi.org/10.1109/MICC.2017.8311756

[8] Y. Liu, Z. Yu, Q. Wang, H. Mei, G. Song and H. Li, "Research on cloud-native monitoring system based on Prometheus," *Fourth International Conference on Sensors and Information Technology (ICSI 2024)*, vol. 13107, p. 131071B, 2024, https://doi.org/10.1117/12.3029320.

[9] A. S. Shaji and M. M. George, "Elastic Stack: A Comprehensive Overview," *2024 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pp. 1-5, 27 July 2024, https://doi.org/10.1109/RAICS61201.2024.10690099

[10] R. K. Vankayalapati, A. Edward and Z. Yasmeen, "Composable Infrastructure: Towards Dynamic Resource Allocation in Multi-Cloud Environments," *Universal Journal of Computer Sciences and Communications*, vol. 1, 2022, https://doi.org/10.31586/ujcsc.2022.1222