Does Generative AI Generate Smells Related to Container Orchestration?: An Exploratory Study with Kubernetes Manifests

Yue Zhang
Auburn University
Auburn, Alabama, USA
yzz0229@auburn.edu

Rachel Meredith
Auburn University
Auburn, Alabama, USA
rpm0033@auburn.edu

Wilson Reeves
Auburn University
Auburn, Alabama, USA
wgr0009@auburn.edu

Julia Coriolano
Federal University of Pernambuco
Recife, Brazil
julia.acoriolano@gmail.com

Ali Babar
University of Adelaide
Adelaide, Australia
ali.babar@adelaide.edu.au

Akond Rahman
Auburn University
Auburn, Alabama, USA
akond@auburn.edu

ABSTRACT

Generative artificial intelligence (AI) technologies, such as ChatGPT have shown promise in solving software engineering problems. However, these technologies have also shown to be susceptible to generating software artifacts that contain quality issues. A systematic characterization of quality issues, such as smells in ChatGPT-generated artifacts can help in providing recommendations for practitioners who use generative AI for container orchestration.

We conduct an empirical study with 98 Kubernetes manifests to quantify smells in manifests generated by ChatGPT. Our empirical study shows: (i) 35.8% of the 98 Kubernetes manifests generated include at least one instance of smell; (ii) two types of objects Kubernetes namely, `Deployment` and `Service` are impacted by identified smells; and (iii) the most frequently occurring smell is unset CPU and memory requirements. Based on our findings, we recommend practitioners to apply quality assurance activities for ChatGPT-generated Kubernetes manifests prior to using these manifests for container orchestration.

CCS CONCEPTS

• **Software and its engineering** → *Software verification and validation*; **Software defect analysis**.

KEYWORDS

container orchestration, empirical study, kubernetes, quality, smell

ACM Reference Format:

Yue Zhang, Rachel Meredith, Wilson Reeves, Julia Coriolano, Ali Babar, and Akond Rahman. 2024. *Does Generative AI Generate Smells Related to Container Orchestration?*: An Exploratory Study with Kubernetes Manifests. In *Proceedings of 21st International Conference on Mining Software Repositories 2024 (MSR '24).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In order to rapidly deploy software applications to end-users, organizations use containers, such as Docker containers. With the practice of container orchestration, i.e., the practice of pragmatically managing the lifecycle of containers with tools, such as Kubernetes [11], organizations have yielded benefits. For example, in the case of Capital One, with Kubernetes the software deployment rate "increased by several orders of magnitude" [9]. Documented evidence of such benefits has helped Kubernetes to become the most popular tool to implement the practice of container orchestration, with an expected market size of 7.8 billion USD by 2030 [19].

While Kubernetes is growing in popularity, practitioners face challenges when using Kubernetes for tasks related to container orchestration [6, 10]. To overcome these challenges, practitioners rely on generative artificial intelligence (AI) technologies [22], such as ChatGPT [13] and GitHub Co-pilot [12]. The above-mentioned generative AI technologies show promise for Kubernetes, as practitioners perceive generative AI to be beneficial for Kubernetes-related tasks [5, 14, 18].

Despite being perceived as beneficial, Kubernetes manifests, i.e., configurations files used for Kubernetes, which are generated by ChatGPT are susceptible to quality issues. Let us consider Listing 1 in this regard. The Kubernetes manifest is generated by ChatGPT [1], and sets no requirement for CPU and memory usage using the `limits` keyword [11]. Unset CPU and memory requirement is detrimental to the security of Kubernetes-based container orchestration, and can facilitate denial-of-service attacks [17, 23]. Hence, systematic investigation is required that will empirically determine if quality issues, such as smells in ChatGPT-generated manifests

MSR '24, April 15-16, 2024, Lisbon, Portugal
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

is prevalent. Such investigation will be helpful in (i) empirically quantifying the quality issues in ChatGPT-generated Kubernetes manifests; and (ii) providing practitioners recommendations on how to use ChatGPT-generated Kubernetes manifests.

```
kind: Pod
metadata:
  name: my-container-pod
spec:
  containers:
  - name: my-container
    image: my-container-image:latest
    ports:
    - containerPort: 80
```

Listing 1: An example of a ChatGPT-generated Kubernetes manifest with one instance of unset CPU and one instance of unset memory requirements.

Accordingly, we answer the following research questions:

- **RQ1**: *How frequently do quality issues appear in Kubernetes manifests that are generated by ChatGPT?*
- **RQ2**: *What Kubernetes objects map to quality issues that appear for ChatGPT-generated Kubernetes manifests?*

We conduct an empirical study where we use container orchestration smells, i.e., coding patterns that violate recommended security and network best practices, as a surrogate measure for quality issues in Kubernetes manifests. We use 98 Kubernetes manifests that are generated by ChatGPT using the DevGPT dataset [24]. By applying static analysis, we quantify the frequency of smells that occur in the set of 98 manifests. We also identify what Kubernetes objects map to the identified smell categories. Code and dataset used in our paper is available online [15].

Contributions: Our contributions of the paper is an empirical evaluation of how frequently smells appear in Kubernetes manifests generated by ChatGPT.

2 BACKGROUND AND RELATED WORK

We discuss background and related work in this section.

Background: Kubernetes provides support for practitioners to manage containerized applications at scale [8] [4]. Practitioners can install Kubernetes on-premise, on cloud platforms, or a combination of both. A Kubernetes installation is also colloquially referred to as a Kubernetes cluster [8]. A pod is the most fundamental unit of a Kubernetes cluster [8]. A pod groups one or more containers with shared network and storage resources according to configurations provided in manifests. Manifests are configuration files developed in YAML, which are executed using the 'kubectl' utility. Listing 2 shows an example of a Kubernetes manifest where configurations of a pod are specified.

Related Work: Our paper is related with prior research that have investigated quality aspects of Kubernetes. Kubernetes security is one topic that have gained a lot of interest. Researchers have quantified vulnerability-related commits for Kubernetes manifests [3], and derived best practices [23]. Blaise et al. [2] evaluated security model of a Kubernetes package manager by identifying the riskiest attack paths. Rahman et al. [17] identified security misconfigurations in Kubernetes manifests. While the above-mentioned publications provide valuable contribution in growing the science of Kubernetes-related quality assurance, these publications have not characterized quality issues in Kubernetes manifests generated by ChatGPT. We address this research gap in our paper.

```
apiVersion: v1
kind: Pod
metadata:
    name: sample-site
    labels:
        app: web
    spec:
       containers:
           - name: front-end
              image: nginx
              ports:
                 - containerPort: 80
           - name: sample-reader
              image: php-nginx
              ports:
                 - containerPort: 88
```

Listing 2: An example of a Kubernetes manifest that specifies a pod with two container instances.

3 METHODOLOGY

We provide the methodology to our research questions as follows:

3.1 Methodology for RQ1

We answer RQ1 using the following steps:

3.1.1 Manifest Collection. For our empirical study, we use all snapshots included in the DevGPT dataset: 'snapshot_20230727', 'snapshot_20230803', 'snapshot_20230810', 'snapshot_20230817', 'snapshot_20230824', and 'snapshot_20230831'. We use all snapshots in order to investigate temporal trends. For each of these snapshots, *first*, we identify entries for which the keyword 'kubernetes' appears in the following fields: 'Title', and 'Body'. *Second*, from the identified entries we separate linked code snippets that are developed in YAML format. We use this step because Kubernetes configuration files are developed in YAML. *Third*, we apply a syntax check to determine which of the YAML files are in fact used for Kubernetes by inspecting for the 'Kind' configuration.

3.1.2 Prompt Collection. From the collected manifests we identify what prompts are used to generate the manifest. For each manifest, we obtain a mapping with the generated prompt.

3.1.3 Qualitative Analysis with Open Coding. As part of RQ1, we conduct a qualitative analysis technique called open coding, which is used to derive high-level categories from unstructured text [21]. With open coding, we derive categories from the prompt text.

3.1.4 Static Analysis. For RQ2, we focus on identifying quality issues in Kubernetes manifests generated by ChatGPT. We use a static analysis tool called 'KubeLinter', which is a static analysis tool that detects smells in Kubernetes manifests [7]. KubeLinter detects two types of smells namely, security and network, which

Table 1: Attributes of Kubernetes Manifests

Attribute	Value
Snapshots	‘snapshot_20230727’, ‘snapshot_20230803’, ‘snapshot_20230810’, ‘snapshot_20230817’, ‘snapshot_20230824’, and ‘snapshot_20230831
Manifest Count	95
Total Lines of Code	1,493
Duration	07/2023 - 08/2023

respectively detects violations of security and network-related best practices for Kubernetes manifests. We apply static analysis on all collected manifests. From the detected smells, we compute metrics as described in Section 3.1.5.

As static analysis tools are susceptible to generating false positives [16], we perform rater verification with randomly-selected 50 smells generated from KubeLinter. Of the 50 smells, 27 and 23 smells are respectively, security and network-related. The first author performs manual inspection to identify if there are any false positives for the generated smells. The first author do not find any false positive instances for the collected 50 smells.

3.1.5 Metric Computation. From the collected smells we report RQ2 using the following metrics: (i) Density: This metric computes how many smells appear for every 1,000 lines of code. We compute this metric for all smell categories using Equation 1; (ii) Proportion: This metric computes the proportion of manifests for which at least one instance of smell appears. We compute this metric for all smell categories using Equation 2; and (iii) Temporal Proportion: This metric computes the proportion of manifests for which at least one instance of smell appears for every month. Unlike, the proportion metric, this metric measures the frequency of smells based on month-wise occurrence. We compute this metric for all smell categories using Equation 3.

$$\text{Density}\ (x) = \frac{\text{Total occurrences of smell } x}{\text{Total line count for all manifests}/1000} \tag{1}$$

$$\text{Proportion}\ (x) = \frac{\text{Total manifests with} >= 1 \text{ instance of } x}{\text{Total manifests}} * 100\% \tag{2}$$

$$\text{Temporal Prop}(x, y)\% = \frac{\#\text{ of smells in month } y \text{ and marked as category } x}{\text{total smells in month } y} * 100\% \tag{3}$$

3.2 Methodology for RQ2

We answer RQ2 by *first* determining which smells appear for what Kubernetes manifests. *Second*, we identify if the detected smell is used to provision an object using `kind`.

4 RESULTS

In all, we mine 95 Kubernetes manifests collected from 6 snapshots. Attributes of the mined Kubernetes manifests is available in Table 1.

4.1 Answer to RQ1: Frequency of Quality Issues

We answer RQ1 using Table 2. We observe proportion and density to be respectively, 146.0 per KLOC and 46.3% for all categories of smells. The most frequently occurring smells are: unset CPU requirements and unset memory requirements, both of which are related to security. The least frequently occurring category is absent anti-affinity, which is network-related.

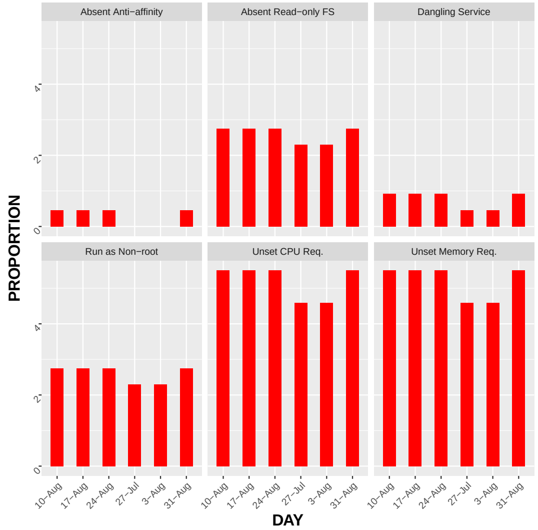

Figure 1: Temporal Proportion for Identified Smells.

We also provide the temporal proportion values in Figure 1. We observe that the manifests collected from latter snapshots, such as ‘snapshot_20230831’ to have higher proportion of smells compared to that of the earlier snapshots. The security-related smells are higher in proportion compared to that of network-related smells.

4.2 Answer to RQ2: Object Categories

We identify two categories of Kubernetes objects. A mapping of smell categories and identified categories is provided in Table 3.

Deployment: This object is used to provide declarative updates for pods. With `Deployment`, a desired state is specified, which is taken as input by the Kubernetes Controller to change the actual state to the desired state.

Example: Listing 4 shows as an example of a `Deployment` object for which an instance of unset CPU and unset memory requirements appear in a Kubernetes manifest generated by ChatGPT.

Service: This object is used to expose a network application that is running as one or multiple pods in a Kubernetes cluster.

Example: Listing 4 shows as a `Service` object called for which dangling service appears in a manifest generated by ChatGPT.

5 DISCUSSION AND CONCLUSION

We discuss the implications of our findings from our empirical study in Section 5.1, and then we conclude our paper in Section 5.2.

5.1 Implications

5.1.1 Implications for Practitioners. The implications of our empirical study is listed as follows:

Table 2: Answer to RQ2: Frequency of Smells in ChatGPT-generated Manifests

Smell	Type	Definition	Count	Density	Proportion
Dangling Service	Network	The issue of not assigning a service to a deployment.	10	6.7	10.5
Absent Anti-affinity	Network	The issue of violating anti-affinity. Anti-affinity ensures that Kubernetes is scheduling replicas on different node.	4	4.2	2.7
Absent Read-only Filesystem	Security	The issue of allowing containers to run without a read-only filesystem.	34	22.7	35.8
Run as Non-root	Security	The issue of not provisioning containers with a non root account.	34	22.8	35.7
Unset CPU Requirements	Security	The issue of not specifying containers with CPU requests and limits.	68	45.5	35.8
Unset Memory Requirements	Security	The issue of not specifying containers with memory requests and limits.	68	45.5	35.8
All Categories	—	—	218	146.0	46.3

```yaml
kind: Deployment
metadata:
  name: influxdb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: influxdb
  template:
    metadata:
      labels:
        app: influxdb
    spec:
      containers:
      - name: influxdb
        image: influxdb:1.8
        ports:
        - containerPort: 8086
```

Listing 3: An example of unset CPU and unset memory requirement appearing for a `Deployment` object.

```yaml
kind: Service
metadata:
  name: influxdb
spec:
  selector:
    app: influxdb
```

Listing 4: An example of dangling service appearing for a `Service` object.

Table 3: Answer to RQ2: Mapping of Smells and Objects

Smell	Object
Dangling Service	Service
Absent Anti-affinity	Deployment
Absent Read-only Filesystem	Deployment
Run as Non-root	Deployment
Unset CPU Requirements	Deployment
Unset Memory Requirements	Deployment

(1) Kubernetes manifests generated by ChatGPT are susceptible to smells that violate security and network-related best practices. Practitioners should apply quality assurance activities, such as application of static analysis tools.

(2) Certain security smells seem to be more prevalent than that of network-related concerns. Practitioners can leverage this finding to prioritize their validation efforts, such as inspecting for smells.

(3) While provisioning service with the Kubernetes object `Service`, network-related smells, such as dangling service occur, and hence while provisioning services practitioners should identify instances of dangling services.

5.1.2 Implications for Researchers. Our empirical study is a preliminary investigation of how frequently smells occur for Kubernetes manifests that is generated by a generative AI technique namely, ChatGPT. We advocate for researchers to conduct further investigation that will build on our research from the following perspectives: (i) a comprehensive understanding of what other categories of smells occur in Kubernetes manifests; (ii) a comparative evaluation of generative API techniques with respect to generating smells; and (iii) an extended mapping of Kubernetes objects to that with smells unique to container orchestration.

5.1.3 Threats to Validity. We discuss the limitations as follows:

Conclusion Validity: Our empirical study is susceptible to conclusion validity as we use manifests that are available from the DevGPT dataset. The generated conclusions are therefore limited, and may not hold for other datasets. Also, the generated smells are limited to to one tool, namely KubeLinter. Furthermore, the analyzed Kubernetes manifests may not be used in production.

External Validity: Our findings may not generalize to other datasets, e.g., manifests that are generated using other generative AI technologies, such as Llama [20].

5.2 Conclusion

Generative AI technologies, such as ChatGPT have shown promise in automating computational tasks, including tasks related to software engineering. The domain of container orchestration is no different. However, as described in our empirical study, we observe Kubernetes manifests generated by ChatGPT to include smells that violate network and security-related best practices. In all, we observe 35.8% of the 98 Kubernetes manifests in the DevGPT dataset to include at least one instance of smell. Based on our findings we conclude that ChatGPT-generated Kubernetes manifests include smells, and developers using ChatGPT should apply quality assurance activities, such as application of static analysis tools to pro-actively detect and mitigate smells in manifests use for container orchestration.

ACKNOWLEDGMENTS

We thank the PASER group at Auburn University for their valuable feedback. This research was partially funded by the U.S. National Science Foundation (NSF) Award # 2247141 and Award # 2312321.

REFERENCES

[1] Anonymous. 2024. K8s-Pod-Example-1. https://chat.openai.com/share/0cee78c4-4290-4bc0-9f2a-a492859d6281. [Online; accessed 20-January-2024].

[2] Agathe Blaise and Filippo Rebecchi. 2022. Stay at the Helm: secure Kubernetes deployments via graph generation and attack reconstruction. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. 59–69. https://doi.org/10.1109/CLOUD55607.2022.00022

[3] Dibyendu Brinto Bose, Akond Rahman, and Shazibul Islam Shamim. 2021. 'Under-reported' Security Defects in Kubernetes Manifests. In *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. IEEE, 9–12.

[4] Canonical. 2021. Kubernetes and cloud native operations report 2021. https://juju.is/cloud-native-kubernetes-usage-report-2021

[5] Dev Nag. 2023. Overcoming the Kubernetes Skills Gap with ChatGPT Assistance. https://thenewstack.io/overcoming-the-kubernetes-skills-gap-with-chatgpt-assistance/. [Online; accessed 18-November-2023].

[6] Jayne Groll. 2022. IT talent: 4 ways to address a Kubernetes skills shortage. https://enterprisersproject.com/article/2022/3/address-kubernetes-skills-shortage. [Online; accessed 19-June-2023].

[7] KubeLinter. 2024. Introduction - KubeLinter. https://docs.kubelinter.io/. [Online; accessed 19-January-2024].

[8] Kubernetes. 2021. Production-Grade Container Orchestration. https://kubernetes.io/

[9] Kubernetes. 2023. Case Study: Capital One. https://kubernetes.io/case-studies/capital-one/. [Online; accessed 10-June-2023].

[10] Janae Lee. 2022. How to beat the Kubernetes skills shortage. https://www.infoworld.com/article/3679749/how-to-beat-the-kubernetes-skills-shortage.html. [Online; accessed 18-June-2023].

[11] S. Miles. 2020. *Kubernetes: A Step-By-Step Guide For Beginners To Build, Manage, Develop, and Intelligently Deploy Applications By Using Kubernetes (2020 Edition)*. Independently Published. https://books.google.com/books?id=M4VmzQEACAAJ

[12] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 1–5.

[13] OpenAI. 2022. ChatGPT: Optimizing Language Models for Dialogue. https://openai.com/blog/chatgpt/. [Online; accessed 12-November-2023].

[14] OpsCruise. 2023. Exploiting ChatGPT for TroubleShooting Kubernetes Problems. https://www.opscruise.com/newsroom-post/exploiting-chatgpt-for-troubleshooting-kubernetes-problems#. [Online; accessed 17-November-2023].

[15] Akond Rahman. 2023. Verifiability Package for Paper. https://doi.org/10.6084/m9.figshare.24786711.v1. [Online; accessed 10-December-2023].

[16] Akond Rahman and Chris Parnin. 2023. Detecting and Characterizing Propagation of Security Weaknesses in Puppet-based infrastructure Management. *IEEE Transactions on Software Engineering* (2023), 1–18. https://doi.org/10.1109/TSE.2023.3265962

[17] Akond Rahman, Shazibul Islam Shamim, Dibyendu Brinto Bose, and Rahul Pandita. 2023. Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. *ACM Trans. Softw. Eng. Methodol.* 32, 4, Article 99 (may 2023), 36 pages. https://doi.org/10.1145/3579639

[18] Renjith Ravindranathan. 2023. ChatGPT for your Kubernetes Cluster — k8sgpt. https://medium.com/techbeatly/chatgpt-for-your-kubernetes-cluster-k8sgpt-649f2cad1bd5. [Online; accessed 19-November-2023].

[19] Markets N Research. 2023. Case Study: OpenAI. https://www.globenewswire.com/news-release/2023/03/06/2621358/0/en/Latest-Global-Kubernetes-Market-Size-Share-Worth-USD-7-8-Billion-by-2030-at-an-23-40-CAGR-Markets-N-Research-Share-Trends-Cap-Adoption-Forecast-Segmentation-Growth-Value.html. [Online; accessed 12-June-2023].

[20] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. (2023).

[21] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.

[22] Daniel Schlagwein and Leslie Willcocks. 2023. 'ChatGPT et al.': The ethics of using (generative) artificial intelligence in research and science. , 232–238 pages.

[23] M. Islam Shamim, F. Ahamed Bhuiyan, and A. Rahman. 2020. XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. In *2020 IEEE Secure Development (SecDev)*. IEEE Computer Society, Los Alamitos, CA, USA, 58–64. https://doi.org/10.1109/SecDev45635.2020.00025

[24] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*.