Immutable Operating Systems: A Survey

Sebastian Böhm[1,*], Guido Wirtz[1]

[1]University of Bamberg, An der Weberei 5, Bamberg, 96047, Germany

Abstract

Immutable Operating System is a relatively new term in the area of operating systems. This type of operating system claims to be immutable, reliable, resilient, and even more secure compared to traditional operating systems. The number of scientific publications regarding Immutable Operating Systems is still limited. There is no widely accepted conceptualization yet. Therefore, this work aims to close this research gap. We utilize a literature review on a subset of solutions to derive the concepts. We use the gained insights to provide a general conceptualization, an overview of use cases and limitations, and finally, a definition of the term. An Immutable Operating System is a special type of operating system, primarily a minimal Linux distribution that introduces read-only file systems, automatic atomic updates, rollbacks, declarative configuration, and workload isolation to achieve better reliability, scalability, and security, especially to serve as a container host. We conclude that this particular type of operating system provides noteworthy enhancements. It can reduce the maintenance efforts in distributed and heterogeneous areas, like edge and fog computing.

Keywords

Immutable Operating System, Container Operating System, Operating System, Linux

1. Motivation

Immutable Operating Systems (IOSs), also called Container Operating Systems (COSs), are an emerging trend in the area of Operating Systems (OSs) research [1–5]. Over the years, several Linux distributions have been published, introducing new concepts to fulfill the claim of being described as an IOS or COS. Specifically, openSUSE MicroOS (oMOS) [1], Fedora CoreOS (FCOS) [6], Flatcar Container Linux (FCL) [3], AWS Bottlerocket OS (ABOS) [7], and Talos Linux (TL) [5] are used for server-centric applications. Whereas in traditional OSs, all files are potentially modifiable by the system or processes, IOSs have a read-only root file system. This leads to many open questions about how an OS can be maintained without the chance to apply changes, like performing updates and writing log files during operation. So far, there is a lack of peer-reviewed scientific literature to provide a common understanding, characterization, and application scenarios of IOSs. Therefore, this paper aims to provide insights into IOSs. This addresses a definition, the concepts, and the technologies used by those systems. In addition,

Hannover'23: 14th Central European Workshop on Services and their Composition (ZEUS 2023), February 16–17, 2023, Hannover, Germany

*Corresponding author.

sebastian.boehm@uni-bamberg.de (S. Böhm); guido.wirtz@uni-bamberg.de (G. Wirtz)

https://www.uni-bamberg.de/en/pi/team/boehm-sebastian/ (S. Böhm);
https://www.uni-bamberg.de/en/pi/team/wirtz/ (G. Wirtz)

0000-0003-3719-1923 (S. Böhm); 0000-0002-0438-8482 (G. Wirtz)

there is no comprehensive overview of typical use cases and limitations, where IOSs play their strengths. Therefore, this contribution addresses the following research questions:

RQ1: What concepts do Immutable Operating Systems follow?

RQ2: What are typical use cases for Immutable Operating Systems?

RQ3: How can an Immutable Operating System be defined?

We perform qualitative literature research to obtain the essential aspects. This procedure allows us to identify already available solutions and alternative identifiers for the area of IOSs. To answer **RQ1**, we analyze the documentation of the identified solutions to assemble a comprehensive list of used concepts and technologies. In addition, we can get insights about the supposed application scenarios of those systems (**RQ2**). Finally, referring to the previous results, we can provide a first definition of IOSs (**RQ3**).

This paper is structured as follows: Section 2 clarifies the term immutability. In Section 3, we take a look at related works. We present our review on IOSs and their features in Section 4. This allows deriving a conceptualization in Section 5. Finally, we conclude our contribution with the limitations of our study (Section 6) and the conclusion (Section 7).

2. Immutability

According to the Cambridge Dictionary [8], immutability means "the state of not changing, or being unable to be changed". In computer science, the term refers mainly to the area of Object-Oriented Programming (OOP). Programs that follow the OOP paradigm consist of objects with state and behavior. A set of attributes represents the state. Also, objects have a particular behavior by providing methods, which may allow the modification of the objects' state [9]. Immutable objects do not allow any modification after the object has been initialized. To derive a modified version of the object, a new (immutable) copy of the object must be created. Conveyed to an OS, the state must not change during runtime. This means that modifications may result in destroying and redeploying the entire system with the updated configuration [10].

3. Related Work

Since the term IOS is relatively new, we could find only one peer-reviewed contribution by Rothmund [11]. It refers to the LinuxKit project [12] that discusses the fundamental potentials of Immutable Servers. The main motivation is the avoidance of configuration drifts, where multiple servers with the same image, configuration, and modifications vary over time nonetheless [13]. The work of Rothmund [11] discussed fundamental requirements for building an immutable server that were also used to design LinuxKit. Although it mentioned a subset of characteristics of immutable servers that are helpful in getting a first basic understanding and characterization of IOSs, it did not perform a review and evaluation of alternative IOSs to derive these characteristics. Furthermore, the concepts of alternative IOSs were not regarded, which limits the possibility and validity of deriving a general conceptualization. The work mentioned only a few alternative IOSs. Therefore, this work aims to provide a more detailed investigation of IOSs by performing a more comprehensive literature review to close this gap.

4. Review: Immutable Linux Distributions

We started our literature review by using Google Scholar, IEEEXplore, and ACM Digital Library with the search terms `Immutable (Linux [Server] | Operating System | Infrastructure)`. This revealed a GitHub repository [14] that contains a collection of immutable Linux distributions. Based on this collection, we checked and added related Linux distributions, which fulfill the claim of IOSs. We excluded all Linux distributions focusing only on desktop environments because our work targets distributed environments (Section 1). We considered five Linux distributions that are explicitly claiming to be immutable for a server-centric application (Table 1).

Immutable (Root) File System. Most of the IOS provide a fully immutable root file system [1, 4, 5, 15]. An exemption is FCL, which provides a writeable root file system for all types of data, for example, container images. However, all system-related files are mapped to the immutable directory mounted at `/usr`. Most IOSs keep some directories, like `/etc`, `/var`, `/home`, and `/root` accessible for read-write operations [4, 15–17]. These directories are used to store configurations, log files, and user data [18]. Nevertheless, OS-related files are still immutable.

Atomic Updates and Rollbacks. Since the root file system is fully immutable, updates must be installed in a particular way. According to the definition of immutability (Section 2), the state cannot be changed. Changes must be applied on a copy of the OS. The distributions achieve this in different ways: oMOS uses `transactional-update` [19]. Technically, a new snapshot of the OS is created by using the underlying copy on write file system btrfs [20]. Then the desired changes are applied, and it can be booted from the newly created snapshot. In case of issues during a reboot, the file system can be reverted to the last known working snapshot [19]. FCOS uses `rpm-ostree` [21] that is a hybrid image and package system. It manages the OS files in a git-like manner with a content-addressed object store and checksums. This allows managing the OS as a repository that enables atomic updates by committing changes and rollbacks by reverting changes [21]. FCL, ABOS, and TL perform atomic updates via an A-B schema. These distributions provide two partitions for the OS, one active partition for the currently running system, and one passive one for upgrades. Once an update process is started, an updated OS base image is downloaded to the passive partition that can then be taken on the next reboot. In case of a failed boot process, the previous working partition is used for the rollback [4, 19, 22, 23]. All IOSs follow the standard release model, where at least a stable and testing branch can be selected [4, 23–25]. In addition, oMOS focuses on a rolling release model [26].

Table 1
Features of Immutable Operating Systems

	openSUSE MicroOS	Fedora CoreOS	Flatcar Container Linux	AWS Bottlerocket OS	Sidero Labs Talos Linux
Initial Release	2018	2018	2018	2020	2018
Release Model	Rolling / Standard	Standard	Standard	Standard	Standard
Immutable (Root) File System	✓	✓	✗, only `/usr`	✓	✓
Atomic Updates / Rollbacks	`transactional-update`	`rpm-ostree`	A-B	A-B	A-B
Automated Updates / Rollbacks	✓ / ✓	✓ / ✗	✓ / ✓	✓ / ✓	✗ / ✓
Declarative Configuration	Ignition, cloud-init	Ignition	Ignition	API, Custom	API, Custom
Workload Isolation	Podman	Podman, Docker	Docker	Docker	Docker, Kubernetes

Automated Updates / Rollbacks. IOSs target an unattended operation [1]. Regular updates should be done automatically. Nearly all of the investigated solutions support automated updates, also based on a schedule [4, 25–27]. Only TL requires a manual update [23].

Since updates may be disruptive, rollbacks to the previous working state should be automatically performed for a seamlessly unattended operation. Mostly all Linux distributions check if the system is booting after an update with the help of integrated low-level tools [4, 22, 23, 26]. FCOS requires manual intervention if an update fails [27].

Declarative Configuration. IOSs should be usable in different environments, like bare-metal, virtualized, or cloud environments. This requires a flexible way of configuration to meet the requirements of the target environment (Section 1). For example, usernames, passwords, SSH public keys, and network information must be set during the first boot process. A declarative configuration is also necessary to easily realize unattended installations for mass deployments. All of the solutions require and provide different tools for declarative configuration. The most popular solution is the tool Ignition [28] that is supported by oMOS, FCOS, and FCL [16, 29, 30]. Configuration tools, in general, allow the modification of disks during early boot. This comprises modifying the disk partitioning, adding files, and basic system configurations as mentioned above [28]. oMOS allows further cloud-init [31] that is also quite popular for declarative configuration [16]. ABOS and TL provide vendor-specific solutions, which are applied during the boot process. Besides that, both can be configured via an API [4, 32].

Workload Isolation. All presented Linux distributions strongly encourage the users to use container technology. This way of virtualization offers workload isolation between different applications that contributes to managing security concerns [33]. Due to the high popularity of OS-based virtualization, a large number of container runtimes, engines, and orchestrators emerged over the recent years [34]. The distributions that are part of this work offer different high-level container engines or orchestrators. Common is Podman [35], for example used by oMOS and FCOS [26, 36]. Docker [37] is also frequently used, for example, by FCL and ABOS [4, 38]. TL itself is designed to run Kubernetes [39], which is integrated into the OS [5]. ABOS does also provide a version that is equipped with Kubernetes by default [4].

5. Immutable Operating Systems

After investigating the core features of IOSs, we can summarize them in general concepts (Section 5.1). Afterward, we highlight use cases and limitations (Section 5.2). Finally, we conclude this chapter by providing a detailed definition, based on the obtained insights (Section 5.3).

5.1. Concepts

Reliability. According to the Institute of Electrical and Electronics Engineers (IEEE), software reliability is defined as “the probability that software will not cause the failure of a system for a specified time under specified conditions” [40, p. 17]. The following two criteria successfully contribute to the reliability of IOSs in a broader sense:

Immutability. Immutability means that all OS-related files cannot be modified during runtime (Section 2). That implies that a working version of the OS runs consistently on every boot [1, 4, 15, 17]. Furthermore, configuration drifts, as described in Section 3, can be effectively avoided [5]. This contributes to reliability because fewer fundamental changes during runtime may avoid crashing system services and user workloads.

Resiliency. The term resiliency can be defined as "the capacity of a system [...] to remain reliable, [...] in case of any malicious or accidental malfunctions or failures that result in a temporal or permanent service disruption" [41, p. 1]. IOSs offer resilience by atomic updates that reduce the likelihood of making the system inoperable. Some Linux distributions automatically verify updates and perform a rollback to the previous working version if an update was not successful. This rollback usually occurs if the OS is no longer able to boot.

Scalability. Scalability is described as follows: "The concept connotes the ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement" [42, p. 17]. In the context of IOSs, it is possible to use them as candidates for scaling out and provisioning a large-scale infrastructure. The following two characteristics contribute to scalability:

Minimality. IOSs provide minimal installation images for bare-metal, virtualized, or cloud environments. They only contain a minimal set of software binaries and libraries to allow for running containerized workloads [3, 5, 26, 43]. This also contributes to security-related aspects because the attack surface is notably reduced.

Configurability. An IOS should be run at scale as a container host. This requires a comfortable way of configuration at scale because mass deployments should be possible. A file-based declarative configuration is an essential aspect of IOSs. Furthermore, using an API to register configuration changes during runtime leads to better configuration flexibility.

Security. All workloads are executed by container virtualization. As already pointed out in the former section, containerization offers security enhancements by isolating workloads from each other. However, there is still the chance for privilege escalation due to vulnerabilities in kernel, container runtimes, and container engines, which allow access to the whole system. Due to the immutable design, it is at least not possible to manipulate the OS, for example, by replacing system libraries. This is a considerable security enhancement. Furthermore, automated updates keep the systems always up-to-date, to avoid security vulnerabilities.

5.2. Use Cases and Limitations

IOSs are intended to operate as container hosts. All of the platforms explicitly state that they provide a minimal OS with preinstalled and preconfigured container runtimes and container engines. Hence, IOSs can be used as standalone container hosts or in clusters, as implemented by TL. IOSs may foster the complexity reduction in edge and fog computing, where many heterogeneous nodes need to be equipped with container virtualization capabilities [44]. Many of the cloud providers adopted the benefits of these systems by providing their solutions, like Google [45], AWS [4], and Microsoft [46].

IOSs limit the modifiability in an extensive way. Mostly, there is no package manager available, or the distributor recommends not installing packages. The usage of applications that cannot be run containerized might be limited. The modification of the root file system is not possible during runtime. Only selected locations are allowed to be written [4, 15, 16], depending on the Linux distribution (Section 4). Consequently, the proposed solutions are not fully immutable in a technical sense. A fully IOS cannot be used because process and application data changes over time [11]. The update of system-related services requires a reboot. For distributions that use an A-B schema for updates, more storage might be required, because the image of the new version must be downloaded and installed.

5.3. Definition

Based on the former qualitative evaluation that has been derived from the official documentation of the different OS vendors, a definition of the term IOS can be derived:

An Immutable Operating System is a special type of operating system, primarily a minimal Linux distribution that introduces read-only file systems, automatic atomic updates, rollbacks, declarative configuration, and workload isolation to achieve better reliability, scalability, and security, especially to serve as a container host.

They are usually built on top of an immutable root file system, perform atomic updates and rollbacks mostly automatically, and are configured in a declarative way to allow for an unattended operation in distributed areas. They leverage advanced file systems, tools, or strategies to realize these mechanisms. Instead of installing workloads with packages, they require the users to provide containers. Whereas not fully immutable in a technical sense, because there are still areas that are allowed to be written, they offer considerable enhancements.

6. Threats to Validity

We conducted literature research with no comprehensive qualitative comparison between the different Linux distributions and no quantitative evaluation, like a performance comparison. To gain our insights, we considered only a subset of available solutions and second only a subset of features to conceptualize and define IOSs. Besides a limited set of peer-reviewed literature, we relied only on the vendors' documentation. Some of the Linux distributions do not have an official release status yet. Fundamental design decisions and used tools of these systems might change rapidly, and the results and findings of this work might expire.

7. Conclusion

This contribution provides a first conceptualization of IOSs (**RQ1**). They have a minimal and immutable root file system, can perform atomic updates and rollbacks, can be configured and modified in a declarative way, and use container virtualization for workload isolation. Typically, they operate as container hosts, expecting to have the primary workload in a containerized form available (**RQ2**). Finally, we can define an IOS as a special type of OS, primarily a minimal Linux distribution that introduces read-only file systems, automatic atomic updates, rollbacks, declarative configuration, and workload isolation to achieve better reliability, scalability, and security, especially to serve as container host (**RQ3**).

We want to expand our work by evaluating further aspects that have been excluded from this survey. This survey only covered a literature-based analysis of the solutions to get a first conceptualization. Practical evaluations, like usability analyses, feature comparisons, and performance analyses between the different distributions, are helpful in gaining further insights.

References

[1] openSUSE Contributors, opensuse microos, 2023. URL: https://microos.opensuse.org/.

[2] Fedora, Fedora coreos documentation :: Fedora docs, 2023. URL: https://docs.fedoraproject.org/en-US/fedora-coreos/.

[3] Flatcar Project Contributors, Flatcar container linux | flatcar container linux, 2023. URL: https://flatcar-linux.org/.

[4] Bottlerocket OS, bottlerocket-os/bottlerocket: An operating system designed for hosting containers, 2023. URL: https://github.com/bottlerocket-os/bottlerocket.

[5] Sidero Labs, Inc., Talos linux, 2023. URL: https://www.talos.dev/.

[6] Fedora, Fedoracoreos, 2023. URL: https://getfedora.org/en/coreos?stream=stable.

[7] Amazon Web Services, Inc., Container host - bottlerocket - amazon web services, 2023. URL: https://aws.amazon.com/bottlerocket/.

[8] Cambridge University Press, Immutability | english meaning - cambridge dictionary, 2023. URL: https://dictionary.cambridge.org/dictionary/english/immutability.

[9] T. Rentsch, Object oriented programming, ACM SIGPLAN Notices 17 (1982) 51–57. doi:10.1145/947955.947961.

[10] B. Fitzgerald, N. Forsgren, K.-J. Stol, J. Humble, B. Doody, Infrastructure is software too!, SSRN Electronic Journal (2015). doi:10.2139/ssrn.2681904.

[11] K. Rothmund, Immutable linux distributionen mit linuxkit, Proceedings of the 2020 OMI Seminars Research Trends in Data Centre Operations, Selected Topics in Data Centre Operations, and Research Trends in the Internet of Things (PROMIS 2020) (2021) 1–10.

[12] Linuxkit, Linuxkit, 2023. URL: https://github.com/linuxkit/linuxkit.

[13] B. Tak, C. Isci, S. Duri, N. Bila, S. Nadgowda, J. Doran, Understanding security implications of using containers in the cloud, in: 2017 USENIX Annual Technical Conference (USENIX ATC 17), 2017, pp. 313–319.

[14] J. Castro, castrojo/awesome-immutable: A list of resources for people who want to investigate image-based linux desktops, 2022. URL: https://github.com/castrojo/awesome-immutable.

[15] Fedora, Configuring storage :: Fedora docs, 2023. URL: https://docs.fedoraproject.org/en-US/fedora-coreos/storage/.

[16] openSUSE Contributors, Portal:microos/design, 2022. URL: https://en.opensuse.org/Portal:MicroOS/Design.

[17] Flatcar Project Contributors, Flatcar container linux disk layout | flatcar container linux, 2023. URL: https://flatcar-linux.org/docs/latest/reference/developer-guides/sdk-disk-partitions/.

[18] LSB Workgroup, The Linux Foundation, Filesystem hierarchy standard, 2023. URL: https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html.

[19] openSUSE Contributors, transactional-update, 2023. URL: https://kubic.opensuse.org/documentation/man-pages/transactional-update.8.html.

[20] Btrfs, btrfs wiki, 2023. URL: https://btrfs.wiki.kernel.org/index.php/Main_Page.

[21] Red Hat, Inc., rpm-ostree, 2023. URL: https://rpm-ostree.readthedocs.io/en/stable/.

[22] Flatcar Project Contributors, Performing manual flatcar container linux rollbacks | flatcar container linux, 2023. URL: https://flatcar-linux.org/docs/latest/setup/debug/

manual-rollbacks/.
[23] Sidero Labs, Inc., Upgrading talos linux | talos linux, 2023. URL: https://www.talos.dev/v1.3/talos-guides/upgrading-talos/.
[24] Fedora, Download fedora coreos, 2023. URL: https://getfedora.org/coreos/download?tab=metal_virtualized&stream=stable&arch=x86_64.
[25] Flatcar Project Contributors, Switching release channels | flatcar container linux, 2023. URL: https://flatcar-linux.org/docs/latest/setup/releases/switching-channels/.
[26] openSUSE Contributors, Portal:microos, 2022. URL: https://en.opensuse.org/Portal:MicroOS.
[27] Fedora, Auto-updates and manual rollbacks :: Fedora docs, 2023. URL: https://docs.fedoraproject.org/en-US/fedora-coreos/auto-updates/.
[28] Red Hat, Inc., Ignition - coreos/ignition, 2023. URL: https://coreos.github.io/ignition/.
[29] Flatcar Project Contributors, Ignition | flatcar container linux, 2023. URL: https://flatcar-linux.org/docs/latest/provisioning/ignition/.
[30] Fedora, Running fedora coreos directly from ram :: Fedora docs, 2023. URL: https://docs.fedoraproject.org/en-US/fedora-coreos/live-booting/.
[31] Canonical Ltd., cloud-init 22.4.2 documentation, 2023. URL: https://cloudinit.readthedocs.io/en/latest/.
[32] Sidero Labs, Inc., Editing machine configuration | talos linux, 2023. URL: https://www.talos.dev/v1.3/talos-guides/configuration/editing-machine-configuration/.
[33] C. Pahl, A. Brogi, J. Soldani, P. Jamshidi, Cloud container technologies: A state-of-the-art review, IEEE Transactions on Cloud Computing 7 (2019) 677–692. doi:10.1109/tcc.2017.2702586.
[34] L. Espe, A. Jindal, V. Podolskiy, M. Gerndt, Performance evaluation of container runtimes, in: Proceedings of the 10th International Conference on Cloud Computing and Services Science, SCITEPRESS - Science and Technology Publications, 2020. doi:10.5220/0009340402730281.
[35] Podman, Podman, 2023. URL: https://podman.io/.
[36] Fedora, Running containers :: Fedora docs, 2023. URL: https://docs.fedoraproject.org/en-US/fedora-coreos/running-containers/.
[37] Docker Inc., Docker: Accelerated, containerized application development, 2023. URL: https://www.docker.com/.
[38] Flatcar Project Contributors, Getting started with docker | flatcar container linux, 2023. URL: https://flatcar-linux.org/docs/latest/container-runtimes/getting-started-with-docker/.
[39] The Kubernetes Authors, Kubernetes, 2023. URL: https://kubernetes.io/.
[40] IEEE, IEEE recommended practice on software reliability, 2017. doi:10.1109/ieeestd.2017.7827907.
[41] C. Colman-Meixner, C. Develder, M. Tornatore, B. Mukherjee, A survey on resiliency techniques in cloud computing infrastructures and applications, IEEE Communications Surveys &amp; Tutorials 18 (2016) 2244–2281. doi:10.1109/comst.2016.2531104.
[42] A. B. Bondi, Characteristics of scalability and their impact on performance, in: Proceedings of the 2nd international workshop on Software and performance, ACM, 2000. doi:10.1145/350391.350432.
[43] Fedora, Fedora coreos frequently asked questions :: Fedora docs, 2022. URL: https://docs.

fedoraproject.org/en-US/fedora-coreos/faq/.

[44] S. Böhm, G. Wirtz, Towards orchestration of cloud-edge architectures with kubernetes, in: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer International Publishing, 2022, pp. 207–230. doi:10.1007/978-3-031-06371-8_14.

[45] Google Cloud, Container-optimized os overview, 2022. URL: https://cloud.google.com/container-optimized-os/docs/concepts/features-and-benefits.

[46] Microsoft, Cbl-mariner documentation - cbl-mariner, 2022. URL: https://microsoft.github.io/CBL-Mariner/.