Design of a microkernel-based services manager for IoT with isolated processing

Diseño de un gestor de servicios IoT basado en *microkernel* con procesamiento aislado

Jose Antonio Ortega-González[1], Luis G. León-Vega[2]

Ortega-González, J.A.; León-Vega, L.G. Design of a *microkernel*-based services manager for iot with isolated processing. *Tecnología en Marcha*. Vol. 35, special issue. IEEE International Conference on Bioinspired Processing. December, 2022. Pág. 46-52.

https://doi.org/10.18845/tm.v35i9.6492

1 Computer Engineering Student, Instituto Tecnológico de Costa Rica. Costa Rica. E-mail: j.ortega98@estudiantec.cr
https://orcid.org/0000-0002-2924-7037
2 Master's in Electronics Student, Instituto Tecnológico de Costa Rica. Costa Rica. E-mail: lleon95@estudiantec.cr
https://orcid.org/0000-0002-3263-7853

Keywords

Internet of things; scheduling algorithms; protocols; high-performance computing; web services; distributed computing.

Abstract

The development of projects related to the Internet of Things has generated a significant impact on the growth of the software industry. Although there are several alternatives to their development, users are limited by inherent factors, such as vendor lock-in, software development flexibility, and security. This article shows a platform design that provides a deployment and management system for IoT projects avoiding some of the limitations of the current tools, such as vendor lock-in and limitations in the development technologies.

Palabras clave

Internet de las cosas; algoritmos de calendarización; protocolos; computación de alto rendimiento; servicios web; computación distribuida.

Resumen

El desarrollo de proyectos relacionados con el Internet de las Cosas (IoT, por sus siglas en inglés) han generado un impacto significativo en el crecimiento de la industria del software. Si bien existen alternativas para su desarrollo, los usuarios se ven limitados por factores inherentes, tales como la dependencia de un proveedor, flexibilidad de desarrollo de software y seguridad. Este artículo muestra el diseño de una plataforma que facilita el despliegue y manejo de proyectos de IoT, solventando algunos de los retos actuales del mercado como los bloqueos de proveedor y limitaciones en las tecnologías de desarrollo.

Introduction

Web services development has been rapidly evolving, moving from monolithic and microservices architectures until arriving at the serverless approach, where a provider offers the option to only upload programmed functions. In particular, the serverless architecture came as a solution to reduce operational costs and has proven to be a good fit for IoT applications [1], with quick set-up times and focusing the development on the application functionality.

Serverless is a paradigm that provides processing resources based on uploading and setting individual functions executed on demand and includes subtasks to other services without having them in the same monolithic environment, such as running a database query or sending a message through the network. The serverless provider fully manages server maintenance tasks, software installation, and security.

Amazon, Google, and Microsoft offer several alternatives for serverless applications. However, most of these products have vendor lock-in or technology limitations that restrict users to migrate from one service to other, as mentioned in [2], tying customers to a single cloud provider and making it difficult and expensive to move to a different vendor in the future. Moreover, there are restrictions to the functionality and the software that can run on the serverless instance. Therefore, there are concerns about vendor lock-in when adopting a cloud service in new projects [3].

There are open-source approaches such as Apache OpenWisk, OpenFaaS, Knative, and Kubeless, that offer an alternative to avoid vendor lock-in [4]. While these solutions provide an advance, there are also remaining challenges to attend to, in particular, the security risk due to the usage of containers [5]. Also, this project integrates services such as storage, communication, and databases, with the computational power of the serverless approach.

This work aims to create an open-source platform that handles automated infrastructure creation and deployment to speed up IoT projects development avoiding vendor lock-in and improving security. Our target is a *high-performant*, *robust, flexible*, *scalable*, *reliable*, and *easy to use* system.

General System Design

The system follows a microkernel-like architecture, which provides communication and scheduling functionalities and delegates other functionalities as services connected to it [6]. In our work, the services comply with a serverless infrastructure for delivering basic web service functionality.

Figure 1 shows a top-level architecture for the system where services are connected to the *Kernel* using Inter-process communication (IPC). The Kernel controls the operation providing an agnostic communication mechanism between the services. The *Doer* is an isolated environment that manipulates *Workers* for process execution. The *DBriver* is a database administrator that handles SQL and NoSQL database entities. The *CommServer* handles the communication services based on HTTP and MQTT network protocols. The *Resadmin* is responsible for the management of the resources and the users. The Control Gate dashboard allows users to create and control their IoT Projects entities.

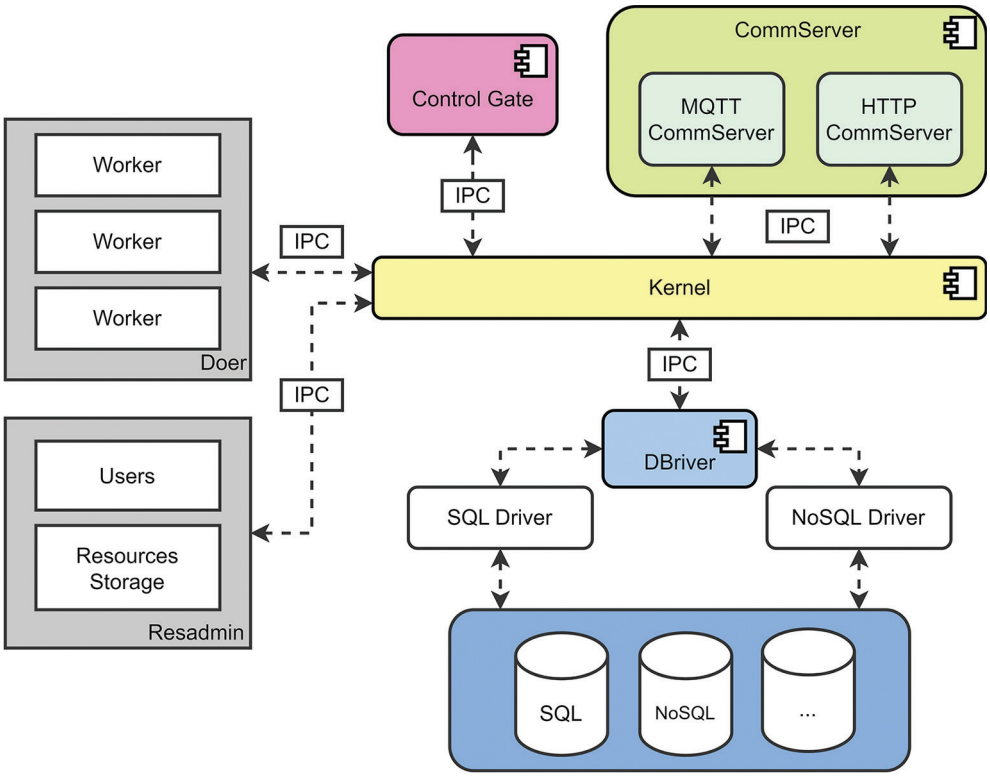

Figure 1. Top-level platform architecture.

Kernel

Figure 2 shows the Kernel design. The Kernel consists of four main modules and one module for interface purposes. We start with the *IPC Module*, which handles all the communication amongst the services. This communication is implemented over an IPC protocol provided by ZeroMQ [7], a high-performance message passing library that provides Operating System (OS)-agnostic support for multiple protocols and communication patterns to create an infrastructure for several services. The IPC Module connects to the *IPC Connections* that provide the mechanism for each service to send and receive a message from the Kernel and other services. It can be seen as an abstracted software interface.

We propose the *Process Controller* for the processes and their management, which handles all the requests that arrive at the system and manages them as processes. Besides, we propose a priority-based queue as *Scheduler*. It handles the message (process) dispatching for each service. Each queue is a Red-Black Binary Search Tree (BST) where the message with the highest priority will be at the left, similar to the Completely Fair Scheduler (CFS) from Linux. Last, we propose a *Services Controller* to handle the information related to the services connected to the Kernel.

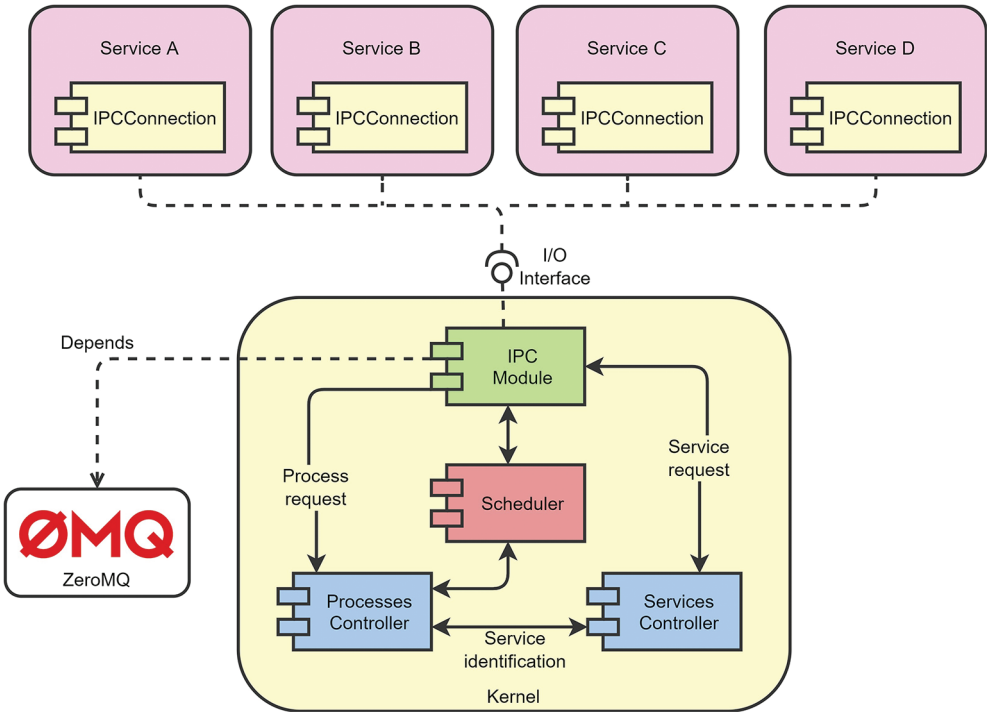

Figure 2. Kernel design overview.

Doer

Figure 3 shows a design overview of the Doer. The Doer is a serverless implementation based on Singularity [8] as a virtualization tool also used in FuncX [9] for Python serverless execution, which could be used in local and cloud environments, provides better security than other tools such as Docker, and focuses on native integration of high-performance computing tools.

The logic for each project is executed using the Doer, which includes three main modules: 1) The *Doer Admin* handles the requests that arrive at the Doer and holds them as *tasks*. It manages three queues depending on the state of each task. 2) The *Worker Handler* invokes and controls the Workers, isolated environments based on containers for executing the programmed functions (called in this project as Doees). 3) The *Worker*: is the environment where the functions are executed. We are using Singularity containers to provide a safe and isolated environment. Furthermore, 4) The *Doee Executor* handles the execution of the *Doees* within the containers. It resides in each worked as a server daemon and is triggered when the Worker receives an execution request.

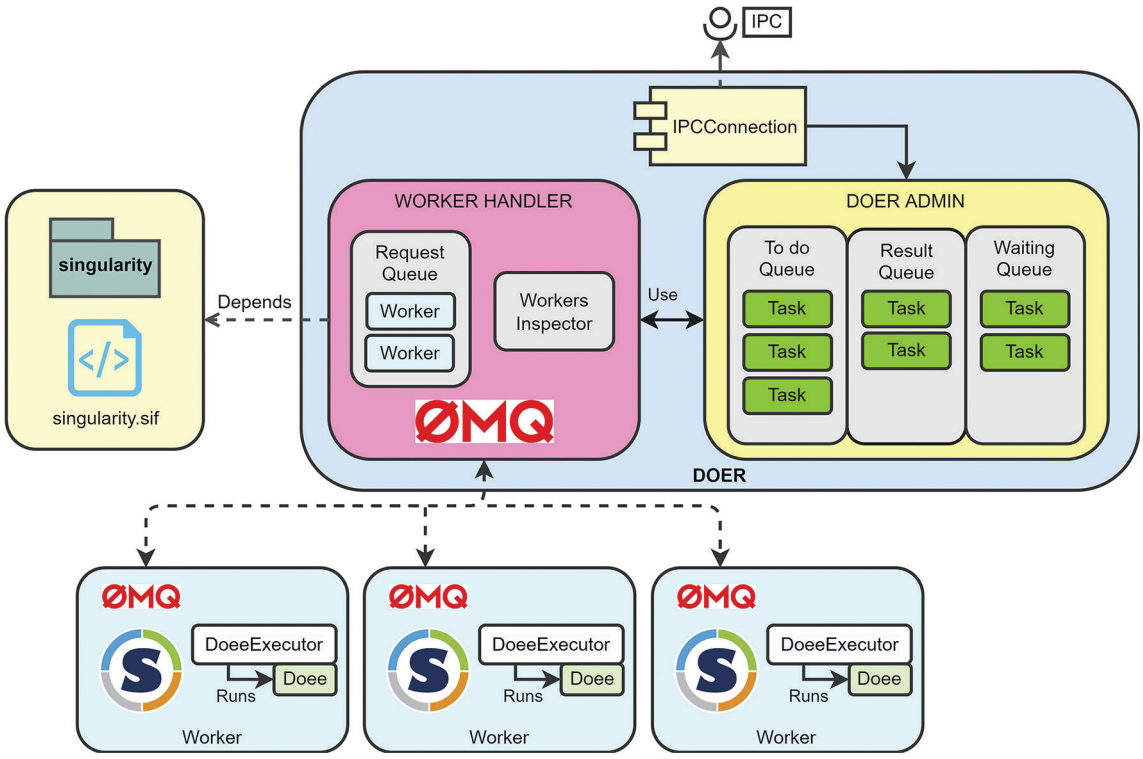

Figure 3. Doer design overview

Since the Doer involves the communication of a host server with isolated Singularity containers, we propose using of ZeroMQ as the communication bridge.

Doee

The functions executed by the Workers are called *Doees*. They are custom pre-compiled shared objects, compiled just after the source code of the Doee is added to the platform. A Worker executes the Doee after a trigger, which can be an action specified like an HTTP GET request. It adds support for other programming languages by using object bindings and interfaces.

The Doee Executor requests the Resadmin to provide the pre-compiled binaries of a single Doee, so the Worker will only contain one Doee at a time. Figure 4 shows the structure of the Doees and how they are created.

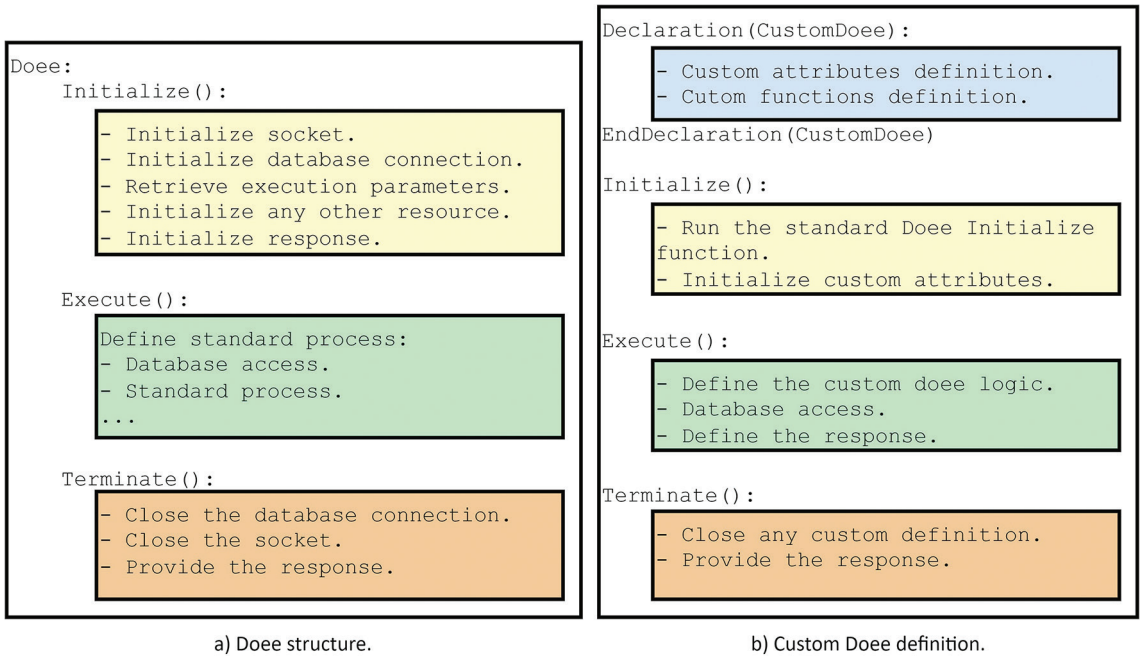

Figure 4. Example of the Doee usage.

Figure 4.a is the structure of the Doee interface class; it requires three main functions: 1) *Initialize* sets the Doee resources required to execute the process, initializes the socket for external communication, the database connection for queries, and retrieves the parameters to be used in the process. 2) *Execute* runs the Doee standard process and a specific logic implemented by the actual user implementation (concrete Execute method). Then, we have 3) *Terminate*, which closes the connections and resources initialized by the Initialize function.

The Doee class is a pure virtual class, and it cannot be executed like any other function. It shall be implemented by a Custom Doee defined by the developer through inheritance, where the specific logic of the function is implemented. Figure 4.b shows how the custom Doee is defined:

1. *Declaration*: The Custom Doee is declared, and the inheritance is automatically added. Also, between the declaration, the developer can define attributes and functions to be used for this Custom Doee.
2. *Initialize*, *Execute* and *Terminate* works as same as for the Doee class.
3. Custom Doee is the function that is executed in the Worker.

The current development of the project can be found in [10]. Updates and new modules will be added there.

Conclusions

Our proposal addresses the vendor lock-in by offering an open-source solution capable of being installed in any computing premise proposed by Apache Open Wisk, Open FaaS, Knative, and Kubeless. Moreover, our contribution to safety is based on Singularity, as applied by FuncX, which differs from many current cloud-based methods in the market.

The platform has more services under development that will be addressed in future work, such as the Resadmin, DBriver, CommServers, and the Control Gate. These modules provide a more integrated platform that offers storage and management to the overall infrastructure. Besides, we plan to compare FuncX and our platform to validate our implementation, executing Python functions.

References

[1] G. McGrath and P. R. Brenner, "Serverless Computing: Design, Implementation, and Performance," *IEEE,* vol. 37th, pp. 405-410, 2017.

[2] J. Opara-Martins, R. Sahandi and F. Tian, "Critical review of vendor lock-in and its impact on adoption of cloud computing," *nternational Conference on Information Society,* pp. 92-97, 2014.

[3] J. Opara-Martins, R. Sahadi and F. Tian, "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective," *Journal of Cloud Computing,* pp. 1-18, 2016.

[4] A. Palade, A. Kazmi and S. Clarke, "An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge," *2019 IEEE World Congress on Services (SERVICES),* pp. 206-211, 2019.

[5] S. Sultan, I. Ahmad and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access,* vol. 7, pp. 52976-52996, 2019.

[6] A. S. Tanenbaum, Modern operating systems, Boston: Pearson, 2015.

[7] ZeroMQ, "ZeroMQ," 2021. [Online]. Available: https://zeromq.org/. [Accessed 2021].

[8] "SingularityCE," Sylabs, 2022. [Online]. Available: https://sylabs.io/singularity. [Accessed 2022].

[9] R. Chard, T. J. Skluzacek, Z. Li, Y. Babuji, A. Woodard, B. Blaiszik, S. Tuecke, I. Foster and K. Chard, "Serverless Supercomputing: High Performance Function as a Service for Science," 2019.

[10] J. A. Ortega and L. León, "IoT Services Management System [Source Code]," 2022. [Online]. Available: https://gitlab.com/klooid/isms/isms-kernel. [Accessed 2022].