A novel approach to software development in the microservice environment of vf-OS

Luís Manteigas da Cunha 1, Ludo Stellingwerff 2 and Andries Stam 3
Almende BV,
Stationsplein 45 unit D1. 116, 3013AK Rotterdam
1 luis@almende.org, 2 ludo@almende.org, 3 andries@almende.org.

Abstract: Within the vf-OS project, a common approach is taken for the deployment of computing resources and data models. As such vf-OS represents a modern approach to the software middleware layer, specifically aimed at the manufacturing domain. The approach consists of "Docker"-based microservices, orchestrated through common REST services. Because of the domain, special attention is given to the challenge of running resources OnPremise versus purely InCloud, and the challenges of letting such heterogeneous environments work together. As this environment poses challenges for the development process of applications, a novel approach to the development of such applications is needed. In this paper, this approach is described. During the vf-OS project, experience will be obtained, validating this approach.

Keywords: Microservices, InCloud vs OnPremise resources, Dynamic resource scheduling, Software development,vf-OS, IoT.

1 Introduction

The World is facing the fourth industrial revolution based on ICT, specifically architectures and services, as key innovation drivers for manufacturing companies. Factories are complex systems of systems and there is a need to develop a platform on which future manufacturing applications can be built.

vf-OS, virtual factory Open Operating System, is an Open Operating System for Virtual Factories composed of a kernel, application programming interface, and middleware specifically designed for the factory of the future. The purpose is to attract talent from solution developers and to provide accessible manufacturing smart applications to European SMEs.

vf-OS offers a manufacturing oriented cloud platform, supporting a multi-sided market ecosystem that provides a range of services for the connected factory of the future, allowing manufacturing companies to develop and integrate better manufacturing and logistics processes. [1]
Due to the highly distributed microservices architecture used in vf-OS, such an environment poses some challenges for the software development process which will produce the assets. This paper describes the approach taken to handle these challenges and describes how this can be evaluated over the lifetime of the project.

2 vf-OS platform

The design of the vf-OS platform environment is formed based on the requirements set out in the public requirements document D1.5. [2] It provides the environment in which vf-OS assets are installed, run and accessed. These assets encompass services, tools and applications. In the case of end-user visible applications, these are called vApps. Such assets interact with each other through web-technologies, such as REST-services, web-based GUIs and modern message busses. See Figure 1.

![Fig 1 vf-OS asset interaction](image)

The platform allows the assets to run both InCloud or OnPremise. This heterogeneous model is the result of the specific requirements for the manufacturing domain. In most use-cases there is data produced within the factories and through the legacy systems of the end-users. This data can be used within cloud applications. This introduces quite stringent security challenges, to allow the users to control their data dissemination. Similarly it requires security measures to prevent undesired access to machinery and other local resources. The manufacturing domain poses several unique privacy and control issues, especially due to the inter-corporate interactions. The relationship between a factory, its subcontractors and its customers is complex, with many contracts on liability, guarantees, services and time constraints.

Within vf-OS this is solved by introducing a model where there are multiple vf-OS platform instances, with a strict inter-platform communication model. One of the
implementation options for this model is shown in Figure 2. It shows a factory, which will run a platform instance locally, which will communicate with a cloud platform instance, through a controlled proxy asset. The customer application will then run in the cloud platform and only get very controlled access to the data from the factory’s premises. The SCC component in the platform is the Security Contact Center, part of the security model of vf-OS.

![Fig 2 vf-OS inter platform communication](image)

The vf-OS assets will have a standardized structure and packaging format. The basic executable entity will be based on Docker images. [3] This Docker image will be enveloped in a wrapper structure, containing metadata, like access rules, dependency information and security signatures. The assets will be storable in the vf-OS Marketplace and can be bought and deployed into the Execution environment, provided by the Platform.

This distributed environment makes it harder to create consistent, coherent applications, especially with regard to debugging, versioning and other software life-cycle aspects. An important part of the goals of the vf-OS project is aimed at providing an answer to these challenges.

### 3 Development approach

The vApp development is the most visible asset development process, as facilitating this process is the stated goal of the project. But the development of the other assets within the vf-OS environment is very similar. It is therefore a good approach to use a single development model for both application development and for the development of other assets. This means that different types of developers will interact with the development model, requiring some flexibility in supporting tools and building blocks.
To tackle this challenge of building consistent, coherent applications within the highly distributed environment of vf-OS, a integrated development environment has been chosen. This environment is called the vf-OS Studio. As all parts of the vf-OS project, it will be available as a set of assets, forming an IDE-like application. An overview of the building blocks of the studio is given in figure 3.

![vf-OS Studio components](image.png)

Fig 3 vf-OS Studio components

As can be seen, the main assets are Javascript-based services. Each service will run in its own docker container, providing some generic APIs for system interacting. For example, each asset will provide data for the vf-OS Dashboard. Similarly each asset exposes an API for interacting with the Execution environment. Assets may choose to also carry a web-based GUI, in the form of containing a web server. (e.g. an NGINX host [4]) Several SDKs and JavaScript libraries are provided by the project for interacting with other assets. Examples of such libraries are the publish/subscribe mechanism, the messaging framework and access to storage facilities.

To support the distribution and marketing of the developed vApps, the project offers a MarketPlace. This MarketPlace has several tasks in the software development process: a) a marketing and sales channel; b) a services registry; and c) hosting of the assets for deployment. Because of the single development model, incorporating this marketplace, these tasks all contribute to the ease of developing vApps.

External services can be represented in the single development model as well, by encapsulating them into vf-OS Assets. For this purpose the project provides a support module, called the External Services Provisioning framework.
4 Conclusions

We have explained a solution to a particularly interesting problem, arising from the manufacturing domain and the micro-services architecture which vf-OS employs to handle the domain. The described approach of standardized packaging, marketing, deployment and interaction between micro-services, is quite novel. Future steps are evaluating this approach, using the projects intermediate releases, use-cases and experimentation. The results of this evaluation will be reported in similar publications.

Acknowledgment

This paper presents work developed in the scope of the project vf-OS. This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement no. 723710. The content of this paper does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in this paper lies entirely with the authors.

References