WHITEPAPER

CLOUDICAL

The Death of the Developer

Cloud, especially when understood as PaaS (Platform-as-a-Service)-approach changes a lot for developers, technologically and in regard to mindset. In fact, it changes so much, that we as developers need to reinvent ourselves – and actually need to bury our old incarnations. This Whitepaper gives a high-level overview on the growing complexities and challenges, developers are confronted with.

Kontakt

Karsten Samaschke
karsten.samaschke@cloudical.io
www.cloudical.io

CONTENTS

When moving into a cloud environment, we could try to execute as we did in the past. This is embraced by lift & shift-approaches, that basically recreate environments existing in traditional datacenters in clouds.

Although this is a viable first step, it should actually be considered as the first step only, since opportunities and challenges in cloud-environments demand completely new approaches in regard to software, operations and integrations. If adjusted to these approaches, the total-cost-of-ownership (TCO) of a software will dramatically reduce while quality of service increases, and time-to-market will lower significantly.

To understand these correlations, we need to see the bigger picture first.

New requirements to software

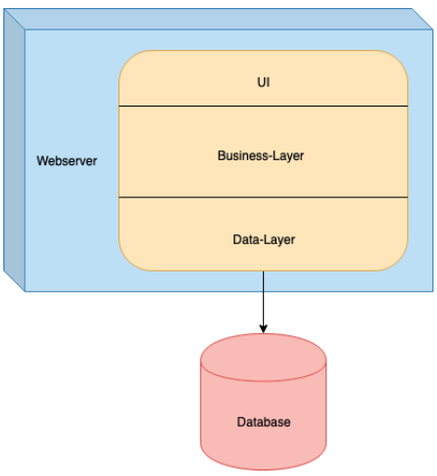
Software gets more and more into the center of attention of organizations, since it is understood as a crucial aspect of execution and strategy. Therefore, developers need to deliver software faster and in a continuous way, ensuring a constant stream of updates and ever-evolving quality. Software needs to run stable, it needs to allow for better testing, and it needs to integrate deeply with infrastructure, allowing for more automated operations and lower operational expenses. And, software needs to be future-proven by design, preventing from technology- and vendor-lock-ins.

These requirements cannot be fulfilled by using old, traditional approaches. Software running in cloud environments needs to be written differently, needs to be set up differently and needs to be orchestrated differently, ultimately forcing the developer to think and to act differently and change our perspectives and mindsets.

Also, the architecture of software needs to be different. This is where Microservices come into play, since they promise to deliver on these requirements.

Microservices

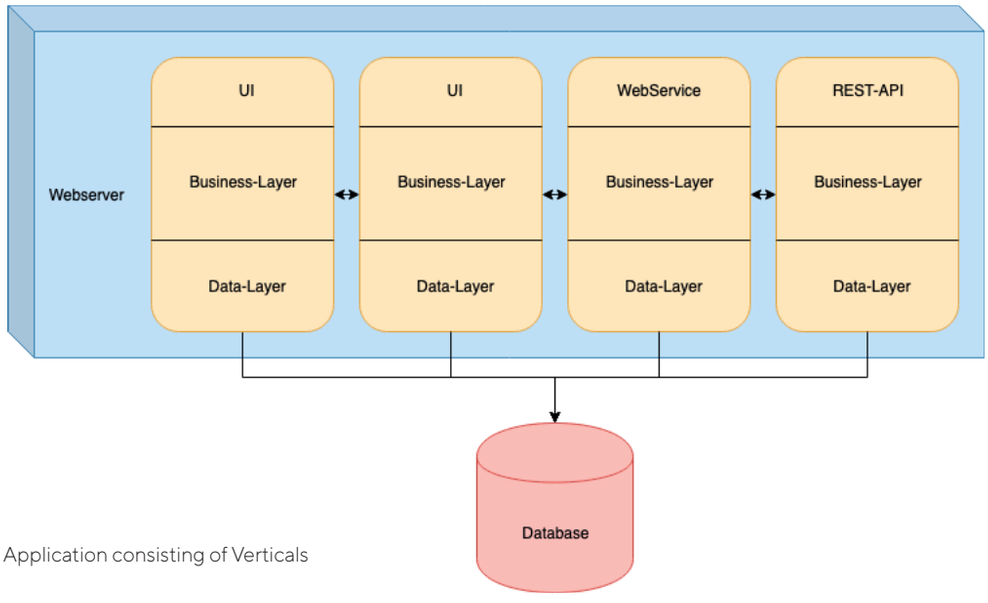
Traditional, monolithic software was easy to develop, but it did not allow for scalability, maintainability and manageability. All parts of a software were strongly coupled with each other, ensuring high performance by utilizing in-memory-communication.



Monolithic Application

This was an appropriate approach to desktop-applications, but it lacked a lot of today's requirements, as testability, simple deployment, fault-tolerance and scalability-requirements are not met at all. Furthermore, all parts of the software were written in one programming language, leading to historically grown, even harder to maintain, software.

This was tackled by the introduction of verticals, spreading an application over some not-so-big-services. While this was a step into the right direction, it is not enough for our modern kind of applications and environments.



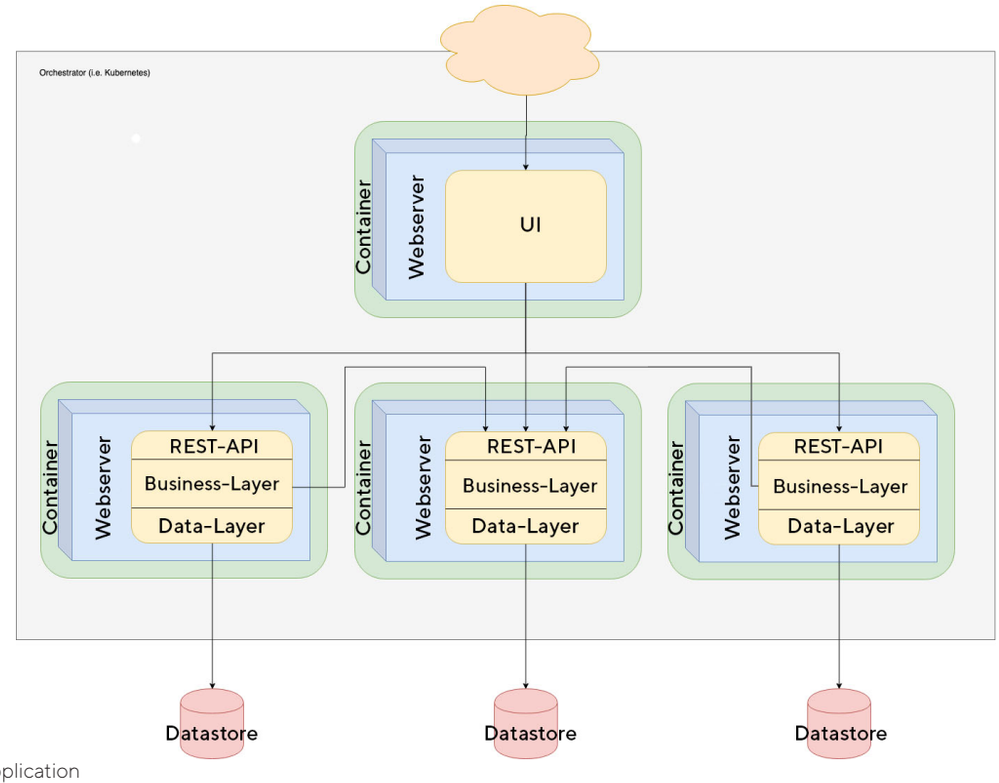
Application consisting of Verticals

This is, where Microservices, depicting a radically different way of building software, appear to be a solution. They emphasize on loose coupling of components. Communication between application components is handled via HTTP/HTTPS-protocols.

Each Microservice is completely self-contained, exposing a REST-api to the outside world, having its own datastore and is executed on a dedicated webserver. This allows for greater independence of each server from any other service, for better fail-over-behavior, for easier scaling and for using the best technology and programming language for the service's purpose.

Each Microservice is usually packed as container-image, allowing for simple installation and independence of the surrounding environment.

The usage of Microservices changes the whole game not only for developers, but also for application architects, since it implies a switch to a completely decoupled application architecture. This kind of architecture raises a lot of questions, in different areas of interest: Orchestration, deployment, fail-over-approaches, transaction management, CI/CD, automation, etc.



Microservice-Application

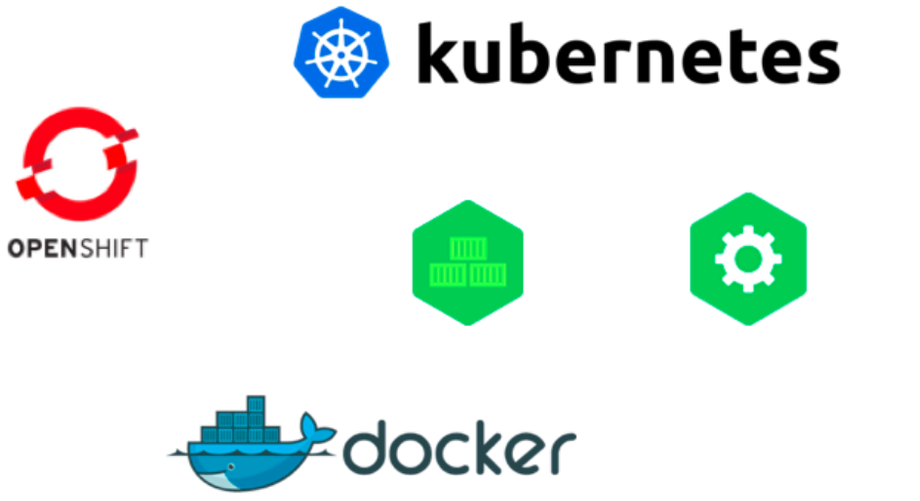
The usage of Microservices changes the whole game not only for developers, but also for application architects, since it implies a switch to a completely decoupled application architecture. This kind of architecture raises a lot of questions, in different areas of interest: Orchestration, deployment, fail-over-approaches, transaction management, CI/CD, automation, etc.

Kubernetes as Container Orchestrator

Since containerized microservices can be tough to operate (as there could be literally hundreds of instances running at the same time), it is important to have some sort of orchestration tool in place.

Currently, the orchestrator of choice is Kubernetes for most enterprises. As a result, developers need to familiarize themselves with it and need to understand where, when and how to use it.

As there are different flavors of Kubernetes available, developers, operators and other stakeholders need to ensure to understand their specific pros and cons. Especially enterprise-grade variants, such as RedHat OpenShift or SUSE CaaSP / CAP, are opinionated in their approaches – allowing for deeper integration with build- and deployment-processes and enforcing a tool-level knowledge beyond a default Kubernetes variant.



Cloud-native Orchestrators and tools

Adding different storage-technologies (such as Rook) and network-overlays to the game, the decision for a specific solution cannot be done by a development team alone, since this kind of technology tends to get strategic on the longer term, is complex by itself and is by no means easy to set up, to integrate and to operate in production grade environments.

(Secure) DevOps

As the complexity of applications and environments increases, the complexity of operations increases as well. Actually, scaling and automatically handling applications implies a lot of additional work to operation teams, if application and environment are not integrated deeply with each other.

Such an integration is not set up easily, since a lot of requirements, approaches and ideas from both sides of the fence need to be coordinated and discussed upon. Therefore, it is essential to set up a DevOps-process by constantly involving developers and operators with each other. Each application-related aspect and requirement of operations needs to be known to developers as early as possible, and each operations-related aspect in regard to infrastructure and processes needs to be known to and understood by the operations team.

And that still is not enough.

Security of applications and environments needs to be targeted as early as possible as well. If not approached early, security requirements will get really expensive, in terms of costs and efforts as well. Application developers and -operators need to be aware of security requirements and constraints, applications need to be written with security in mind, infrastructures need to be set up accordingly. Since setting up a robust DevOps-process involving security aspects is already often ignored in traditional environments – and ignoring this in cloud environments, with services constantly scaling up and down and new releases being deployed every few minutes, implies even bigger problems.

(Secure) CI/CD

Speaking of deployments: Since cloud-native applications mainly consist of dozens of Microservices (or serverless functions) and are supposed to be deployed by an orchestration environment such as Kubernetes automatically, the process of creating those deployments and providing the images needs to be automated as well. And – perhaps even more importantly – this process needs to run in a secure environment. This basically implies: Base container images are downloaded from a trustworthy and controlled source, external libraries need to be downloaded from a trustworthy and controlled source as well (perhaps they even need to be acknowledged by a security team), everything is build inside a controlled environment and is to be deployed from that controlled environment as well.

Or in different words: No binaries as libraries, no base images from Docker hub and no manual installation or deployment of artifacts or images. This implies not only to think of automation, but also to set up a controlled and secure build- and deployment chain, which is understood to be a vital and essential part of any development- and operations process.

Automation and transparency

In cloud-environments, automation is a crucial aspect of lowering costs and ensuring steady and ongoing deployments of new versions. For operators, there is no such thing as SSH anymore, at least not in their tool chain. For developers, everything needs to be automated – including creating their test infrastructure and pushing out each deployment. Surely, there can be manual steps involved, but these should only be understood as quality gates. Every single aspect of building and running an application, needs to be automated.

Grafana Dashboard (Picture taken from https://github.com/grafana/grafana)

Another very important aspect is transparency: In the past, developers simply dropped some output in a log file, and when something happened, an operator (or the developer) would look into the log file and tried to understand the error message or the traces being visible in there. Basically, this approach would be valid for cloud-native applications and environments as well – with one caveat: With cloud-native environments, not only one instance of a service is available at a given time, but perhaps dozens. And when an instance crashes, the orchestration software will simply remove that instance and start a new one. This leads to major problems when trying to solve issues, if not tackled by utilizing centralized logging and centralized tracing approaches.

Integration

The ultimate goal when writing cloud-native applications, is to bring all of the things previously discussed together – and to add even more aspects on top, allowing for deep integration with cloud-native infrastructure and orchestrators. This integration ensures the environment to be aware of the state of any application component and to be able to act accordingly on its own without manual interaction. This implies scaling services up and down, replacing faulty instances, allowing for A/B-testing and Blue/Green-deployment-approaches without any downtimes.

The death of the developer (as we understood it)

All of these aspects are achievable, when we as developers (and operators) bury our old approaches and our traditional mindsets. We need to understand ourselves as a team, being responsible for an application end-to-end, involving security-experts and other stakeholders from the very beginning.

In this sense, the old understanding of developers working on their own, of monolithic applications and unchanged environments is a thing of the past. This new type and new generation of developers succeeds their predecessors in many aspects, having a wider picture and living a broader collaboration than ever before. The mindset is fundamentally different to that of past generations, ultimately implying the death of these old-fashioned developers.