UDC 004.42

4. INFLUENCE OF SOFTWARE AND HARDWARE INFRASTRUCTURE ON ARCHITECTURAL STYLES IN SOFTWARE DEVELOPMENT

Rabinkin H.M. Master's degree student, group 256241

Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus

Liakh Y.V. – Senior Lecturer

Annotation. Software architecture is a dynamic and evolving field that requires constant questioning of assumptions and axioms. Recent innovations in software development related to hardware and software infrastructure have dramatically changed the environment in which programmers work. This article analyses the capabilities and trade-offs of different architectural styles and patterns in the modern environment, considering all the recent innovations and trends.

Keywords. Architecture, cloud computing, continuous delivery, continuous integration, deployment, DevOps, distributed architecture, hardware infrastructure, microservice, monolithic application, server, software, software architect.

Software architecture is a crucial aspect of software development that defines the fundamental structures, properties, and relations of a software system. In this paper, we explore how software and hardware infrastructure affect architectural styles and design in software development. Recent innovations in software development related to hardware and software infrastructure have dramatically changed the environment in which programmers work. Limitations that influenced the origin of most common and fundamental software architecture styles and designs do not exist anymore or have been mitigated by new solutions. This paper analyses the capabilities and trade-offs of different architectural styles and patterns in the modern environment. It also reviews well-known patterns from a fresh perspective, using new tools,

metrics, and examples to illustrate their strengths and weaknesses. The article could be considered as an attempt to understand software architecture considering all the recent innovations and trends that have shaped the software ecosystem.

Software architecture refers to “the fundamental structures of a software system and the discipline of creating such structures and systems”. It describes how multiple components or processes interact with each other to complete a system that meets functional requirements defined by stakeholders. Software architecture also involves making decisions based on various factors such as cost, feasibility, modifiability, reliability, reusability, performance, security, manageability, etc.

Software architecture is a complex concept that encompasses different factors that affect how a system is designed and implemented. It is not enough to describe an architecture by its structure alone, such as the type of architecture style used. An architecture also needs to specify the architecture characteristics, which are the non-functional requirements that define the success criteria of a system. Additionally, an architecture needs to include the architecture decisions, which are the rules and constraints that guide the development teams on how to build the system. Finally, an architecture needs to consider the design principles, which are the best practices and recommendations that help improve the quality of the system.

One of the challenges of defining software architecture is understanding what each factor entails and how they relate to each other. For example, what are some common architecture characteristics and how do they affect the structure and behavior of a system? How do architects choose and apply design principles that suit their context and goals?

Software architecture is a term that has different meanings for different people. Some think of software architecture as the plan of the system, while others think of it as the process of creating a system. But these definitions are too vague and do not explain what the plan or the process actually includes.

According to its definition, software architecture is made up of four elements: the structure of the system, the architecture characteristics that the system needs to have, the architecture decisions that set the rules for how to build the system, and finally the design principles that give advice on how to improve the system [1].

The structure of the system means what kind of architecture style (or styles) is used to design and implement the system (such as microservices, layered, or microkernel). But just knowing the structure is not enough to understand an architecture. For example, system could be described as a “microservices architecture”, it only tells you about the structure of the system, not about its architecture. It is also necessary to know about its architecture characteristics, architecture decisions and design principles to get a complete picture of its architecture [1, 2].

But at the same time, architecture style is one of the main characteristics of the system. One way to classify software architecture is by using architectural styles or patterns. An architectural style is a named collection of architectural design decisions that are applicable in a given development context, constrain architectural design decisions that are specific to a particular system within that context, elicit beneficial qualities in each resulting system. Some examples of architectural styles are:

Layered: a style where components are organized into layers that provide services to higher layers while hiding details from lower layers.

Client-server: a style where components are distributed into clients that request services from servers that provide them.

Pipe-and-filter: a style where components are connected by data streams (pipes) and process data (filters).

Event-driven: a style where components communicate by sending or receiving events.

Microservices: a style where components are small, independent services that communicate through lightweight protocols.

These are just some examples of architectural styles; there are many more depending on different criteria or perspectives. Moreover, complex systems allow to combine several styles at the different abstraction layers. For example, a client-server system could consist of microservices that could be divided into layers (layered architecture) and even some microservices could be organized as pipe-and-filter modules.

Microservices are an architectural style that has gained popularity for designing and developing complex software systems in recent years. Unlike traditional monolithic applications that consist of a single large unit, microservices decompose an application into a collection of small, independent services that can be deployed and scaled separately. Each service is focused on a specific domain or functionality and communicates with other services through well-defined interfaces. This architecture style offers several benefits, such as resilience, scalability, flexibility, and faster delivery. However, it also introduces some challenges, such as increased complexity, network latency, testing difficulties, and configuration management. Therefore, adopting microservices requires careful analysis of the trade-offs and implications for each system context.

According to a report by O'Reilly Media, 77 % of respondents have adopted microservices in 2020, with 92 % experiencing success with them. Another report by JetBrains shows that 35 % of all respondents develop microservices in 2021, with Java (41 %), JavaScript (37 %), and Python (25 %) being the most popular languages for microservices development [3, 4]. These statistics indicate that microservices are becoming a mainstream choice for building modern software applications.

Each of the styles has its own requirements and limitations. For example, microservice architecture approach requires highly developed infrastructure and orchestration tools in order to organize interaction between system internal components via APIs and gateways. Therefore, it is no longer a problem for developers to access these tools, increased system complexity can introduce a lot of problems in further development and system maintenance.

Infrastructure is another important factor that influences software architecture. In software development, infrastructure can refer to both software and hardware resources that support the creation, deployment, and operation of a software system. Some examples of software infrastructure are:

Operating systems: the software platforms that provide basic services and interfaces for applications running on them.

Middleware: the software layers that provide common functionality and communication mechanisms for distributed applications.

Frameworks: the reusable software libraries or tools that provide specific functionality or structure for applications built on them.

Cloud computing: the delivery model of computing resources as services over the Internet [5].

Some examples of hardware infrastructure are:

Networks: the physical devices, cables, routers, switches, etc., that enable data transmission between computers.

Servers: the computers that host applications or services for clients.

Storage devices: the devices that store data persistently, such as hard disks, solid state drives, tapes, etc.

Sensors: The devices that collect data from physical phenomena, such as temperature, humidity, motion, etc.

Infrastructure affects software architecture in various ways. For example, infrastructure can impose constraints or requirements on software architecture, such as compatibility, scalability, availability, security, etc. Infrastructure can enable or facilitate certain architectural styles or design choices, such as distributed computing, parallel processing, service-oriented architecture, etc. Infrastructure can also evolve over time due to technological advances or changes in demand or environment. This may require adaptation or migration of existing software architectures [6].

The most noticeable impact on software development is related to the recent changes in cloud infrastructure, so called “cloud revolution”. Cloud infrastructure refers to the hardware and software components that enable cloud computing, such as servers, networks, storage, virtualization, and management tools. Cloud infrastructure can be deployed in different models, such as public cloud, private cloud or hybrid cloud. Cloud infrastructure offers various benefits to organizations, such as scalability, flexibility, cost-efficiency, security, and innovation.

Cloud infrastructure has evolved rapidly in recent years due to several factors, such as technological advancements, changing business needs, increased competition, and COVID-19 pandemic [3, 4]. Some of the key trends that have shaped the development of cloud infrastructure are:

The adoption of multicloud solutions that leverage multiple cloud providers and platforms to optimize performance, reliability, security, and cost.

The emergence of edge computing that brings data processing closer to the source of data generation to reduce latency and bandwidth consumption.

The integration of artificial intelligence and machine learning capabilities into cloud infrastructure to enable automation, optimization, prediction, and personalization.

The shift to distributed DevOps that enables faster and more agile delivery of software applications across multiple cloud environments.

The increase in cloud spending driven by COVID-19 that has accelerated the digital transformation of many organizations.

These trends indicate that cloud infrastructure is becoming a vital component of modern business architecture and a key driver of innovation and growth.

Cloud providers offer various services and platforms that enable software developers to build, test, deploy, and manage applications on the cloud. However, there are also some challenges, such as data privacy, compliance, vendor lock-in, and performance issues.

According to a report by Statista, the global market size of cloud computing was estimated at 371.4 billion U.S. dollars in 2020 and is expected to grow to 832.1 billion U.S. dollars by 2025. The leading cloud IT infrastructure vendors worldwide in 2019 were Dell Technologies (15.2 %), HPE (14.7 %), Cisco (13.6 %), Inspur (8 %), and Lenovo (5.9 %). Another report by statista.com shows that 94 percent of

enterprises use at least one cloud service in 2020 and that the average enterprise uses nearly five different clouds. The most used cloud platforms by developers worldwide in 2020 were AWS (51 %), Microsoft Azure (45 %), Google Cloud Platform (31 %), IBM Cloud (12 %), and Alibaba Cloud (9 %) [3, 4].

These statistics indicate that cloud software development is a dominant trend in the software industry and that cloud providers play a crucial role in enabling digital transformation and innovation for organizations of all sizes.

In recent years, there have been several trends in software development that have implications for architectural styles and design. There also have been several innovations in software and hardware infrastructure that have mitigated or even removed some of the limitations that existed when most common architectural styles were founded.

For example, cloud computing has enabled developers to access virtually unlimited computational resources on demand, without worrying about installation, maintenance, or scalability issues. Similarly, IoT devices have enabled developers to collect and process data from various sources and locations, without relying on centralized servers or networks. These innovations have also created new opportunities and challenges for software development, requiring new or adapted architectural styles and design choices.

One of the architectural styles that has gained popularity in recent years is microservices. Microservices is a style where components are small, independent services that communicate through lightweight protocols. Microservices aim to achieve high cohesion, low coupling, modularity, scalability, availability, and agility. Microservices also leverage some of the benefits of modern infrastructure, such as cloud computing, containerization, orchestration, etc [7, 8].

However, microservices is not a silver bullet and has a lot of trade-offs, vulnerabilities, and weaknesses. For example, microservices can increase complexity, overhead, latency, and inconsistency due to distributed communication and coordination. Microservices can introduce security risks due to exposure of interfaces, dependencies, or data [1, 9]. They can require more effort and expertise for testing, monitoring, debugging, or deploying. Therefore, microservices should not be considered as a one-size-fits-all solution and should be applied with caution and consideration.

Most common fallacies of distributed architectures styles like microservices could be defined as:

Network Reliability: the network is often assumed to be reliable, but it is not. This affects all distributed architectures that depend on the network for communication. For example, Service A may fail to reach Service B or get a response from it due to a network problem. This is why timeouts and circuit breakers are needed between services. The more a system uses the network (like microservices architecture), the more unreliable it may become [1, 10].

Communication latency: local calls are much faster than remote calls in any distributed architecture. Remote calls have latency that is not zero and varies widely. Architects need to know the average and the worst-case latency of their remote calls, especially for microservices that communicate a lot. Latency can affect performance and feasibility of a distributed architecture.

Bandwidth limitations: bandwidth matters in distributed architectures like microservices, because services need to communicate a lot. Too much data transfer can slow down the network and affect latency and reliability. For example, a wish list service may request too much data from a customer profile service that it doesn't need. This wastes bandwidth and causes coupling. To avoid this, services should only pass the minimal amount of data needed by using techniques like private RESTful API endpoints, field selectors, GraphQL, value-driven contracts or internal messaging endpoints [11].

Network Security: the network is always not secure in a distributed architecture. VPNs, trusted networks and firewalls are not enough. Every endpoint to every service must be secured from “bad” requests. The risk of threats and attacks grows when moving from a monolith to a distributed architecture. Securing every endpoint also slows down performance in synchronous distributed architectures like microservices or service-based architecture.

Topology changes: the network topology is not fixed. It changes often due to upgrades or maintenance. This can affect latency and cause timeouts and circuit breakers. Architects need to communicate with operations and network administrators to know what is changing and when. This can help them avoid surprises and adjust their assumptions [1, 11].

Multiple administration points: there are many network administrators in a large company. Architects need to communicate and collaborate with them about latency and topology changes. This is complex and challenging for distributed architecture. Monolithic applications do not need this much communication and collaboration.

Transport cost: it means the money needed to make a remote call or use a distributed architecture. Architects often assume that the existing infrastructure is enough and cheap. It is not. Distributed architectures cost more than monolithic architectures because they need more hardware, servers, firewalls and other things. Architects should analyze the current infrastructure before using a distributed architecture to avoid surprises.

Network is not homogeneous: software developers could assume that a network is made up of only one type of hardware, but this is not true. Most networks have different hardware from different vendors,

and they may not work well together. This can cause problems with network reliability, latency, and bandwidth, which affect distributed architectures. This fallacy is related to all the other network fallacies.

Distributed architecture has more challenges than monolithic architecture besides the eight fallacies mentioned above. They are distributed transactions, distributed logging, cost maintenance and versioning and others.

In conclusion, it should be mentioned that software architecture and infrastructure affect each other in modern software development. Software architecture is defined as a combination of four elements: structure, characteristics, decisions, and principles. Different architectural styles or patterns affect and are affected by these elements.

The availability of computing resources and affordability of cloud infrastructure brought distributed architectural styles to the foreground of software development. The focus on microservices is defined by the popularity of this style for building complex software systems. The impact of cloud infrastructure enables cloud computing as a delivery model of computing resources over the internet. But besides the obvious advantages of such architectural style, microservices introduce to the system common fallacies of distributed architectures that affect network reliability, communication latency, bandwidth limitations, etc.

Software architecture is not a static or isolated concept but a dynamic and interrelated one that evolves with changing requirements, environments, tools, metrics, and examples. Understanding the capabilities, trade-offs, vulnerabilities, and weaknesses of different styles and patterns in the current context is necessary. Software architects should also be able to adapt and migrate existing architectures to leverage new opportunities and overcome new challenges created by modern infrastructure innovations such as cloud computing, containerization, microservices, DevOps, and continuous delivery.

References:

1.Richards, M. Fundamentals of Software Architecture: An Engineering Approach / M. Richards, N. Ford. – Sebastopol : O'Reilly Media, Inc., 2020. – 401 p.

2.Richards, M. Software Architecture Patterns / M. Richards. – Sebastopol : O'Reilly Media, Inc., 2017. – 47 p.

3.Statista [Electronic resource] : Cloud computing – Statistics & Facts. – Mode of access: https://www.statista.com/topics/1695/cloud-computing/. – Date of access: 23.02.2023.

4.Cloudwards [Electronic resource] : Cloud Computing Statistics, Facts & Trends for 2023. – Mode of access: https://www.cloudwards.net/cloud-computing-statistics/. – Date of access: 19.02.2023.

5.Patel, H. Cloud Computing Deployment Models: A Comparative Study / H. Patel, N. Kansara // International Journal of Innovative Research in Computer Science & Technology. – 2021. Vol. 9, iss. 2.

6.Lv, Y. Infrastructure Smart Service System Based on Microservice Architecture from the Perspective of Informatization / Y. Lv, W. Tan // Mobile Information Systems. – 2022. Vol. 2022, article 1344720.

7.Waseem, M. A Systematic Mapping Study on Microservices Architecture in DevOps / M. Waseem, P. Liang, M. Shahin // Journal of Systems and Software. – 2020. Vol. 170, article 110798.

8.Francesco, P.D. Architecting with microservices: A systematic mapping study / P.D. Francesco, P. Lago, I. Malavolta // Journal of Systems and Software. – 2019. Vol. 150, 2019.

9. Microservice Architecture: Aligning Principles, Practices, and Culture / I. Nadareishvili [et al.]. – Sebastopol : O'Reilly Media, Inc., 2016. – 199 p.

10. Haris, M. A Systematic Review on Cloud Computing / M. Haris, R. Kahn // International Journal of Computer Sciences and Engineering. – 2018. Vol. 6, iss. 11.

11. Jaiswal, M. Cloud computing and Infrastructure / M. Jaiswal // International Journal of Computer Sciences and Engineering. – 2017. Vol. 4, iss. 2.