Aspecte arhitecturale in cloud - SOA and Microservices -

Assoc. Prof. Lenuța Alboaie
adria@info.uaic.ro
Content

• SOA and Microservices in the same picture?
• SOA Dichotomy
• Microservices
• SOA, Microservices in the APIs world
Service Oriented Architecture

- *Service Oriented Architecture* is a design methodology and architecture aimed at maximizing the reuse of multiple services (possibly implemented on different platforms and using multiple programming languages)
  - Services share a formal contract – contains details (and additional metadata) such as their capabilities, interfaces, policies, supported protocols
  - Services must be *loosely coupled*
  - *Abstracting* functionality and technology
  - *Reuse* principle promotes the idea of service as a “repeatable value”. When a business logic is built, reusing it decreases the time of creation
  - Service *Discoverability*
  - Services can be composed to provide other services – *composability*
  - Services are individually useful – they are *autonomous*.
    - => predictable performance, independence of the service, flexibility, scalability, ability to be easily replaced, fault tolerance and, of course, reuse
  - Service *stateless*
Service Oriented Architecture

• Open Group SOA Integration Maturity Model (OSIMM) proposed seven levels that should be respected by a SOA application in order to reach “maturity level”
  – E.G. a point-to-point integration is considered an integration service with 2 value (Service Integration Maturity Model - SIMM = 2) or a hub-and-spoke architecture corresponds to SIMM =3
• Obs. For small-scale systems, redundancy or even refactoring and consolidation may not be so expensive, for large companies these can spell total failure
Microservices

• Microservices architecture
  – an approach to structuring applications
  – an application is broken into smaller, completely independent components => enabling greater agility, scalability, and availability
  – focus on the structure and components within an application => application scope
Confusion

• SOA versus Microservices
  – componentization, decoupling, and standardized communication protocols, describe most software initiatives in the last few decades

  – ? microservices architecture is a "fine-grained SOA" or that it is "SOA done right“ ? -> NO
SOA

- SOA exposes services through standardized interfaces between applications => *enterprise scope*
SOA

• SOA address two different needs:
  – Integration-led technical element
  – Business-led functional element
SOA

- SOA address two different needs:
  - Integration-led technical element
    - the need to integrate existing systems over their complex and often proprietary data formats, protocols, and transports
    - expose them by using standardized mechanisms (such as SOAP/HTTP or more recently JSON/HTTP) to make them easier to re-use in new applications
    => Enterprise Service Bus (ESB) pattern

  **Deep integration:** (integration hub and adapters) and to expose these integrations as services or APIs in a standardized way

  ? Little relationship with microservices 😊
SOA

• SOA address two different needs:
  – **Business-led functional element**
    • the current systems are largely meaningless ⇔ they don't make sense to the business ⇔ data that is required might be spread across multiple systems
    • => functional refactoring to expose something that the business can tangibly build into future solutions
    • This refactoring requires the creation of new applications to bind together requests across the existing systems of record. In the SOA reference architecture, these applications were referred to as service components

? a relationship to application design (and, therefore, the microservices architecture) and the functional decomposition of capabilities into separate components✿
SOA

- Challenge to mix the views => painfully results
  - For some organizations, their greatest challenge is the diversity and complexity of the integration
  - For others, refactoring and relandscaping to achieve the right business functionality is the primary challenge

- Integration tools are not the right place to perform business logic
- Conversely, you do not want your business applications to be cluttered with technical integration concerns
SOA

• SOA reality:
  – The goal of most SOA programs is to achieve the functional aspects; They want easily accessible business-relevant services that can be used to build new applications more effectively
  – In large enterprises, SOAs are often perceived to have failed; This thought might be true in that they failed to deliver the final business value, although huge strides were made to improve the accessibility of systems of record
  – In smaller companies (or more contained environments within larger companies), SOAs often claim true business successes because they can quickly overcome integration issues and move on to functional benefits
SOA

• SOA exposed services versus/and APIs
  – APIs - it equates to REST interfaces, which provide data by using the JSON data format (sometimes XML), and the HTTP verbs PUT, GET, POST, and DELETE to depict create, read, update, and delete actions.

    These protocols and data formats are simpler to use than the web services standards based on SOAP that were more prevalent in early SOA.
  – The difference lies in the intent behind APIs and SOA services. One key difference is in their economics.
SOA

• SOA exposed services versus/and APIs
  • In SOA programs, service exposure was about exposing each business function so that it could be reused as much as possible => each new project didn't have to go through the pain of performing integration to the back-end system again.
    The typical consumers were internal applications that attempted to put fresh user interfaces onto older systems of record. At the time, integration was difficult and took a significant portion of an IT project's budget. If you could make all of the core functions of the company available over reusable interfaces, you could significantly cut project costs.
  • SOA was about cost saving, not generating new revenue.
SOA

- SOA exposed services versus/and **APIs**
  - APIs had a different starting point, with the assumption that integration was already simplified. This simplification occurred either through an earlier SOA initiative or by upgrading back-end systems to provide more ready-to-use modern interfaces.
  - API popularity skyrocketed with the rise in smartphone use. Smartphones run rich client-side applications, creating a disruptive new business channel. As a result, application developers needed simple access to back-end functions and data; they needed APIs. **APIs became a saleable product.**
  - The lines between **SOA web services** and **API** are now blurred and almost irrelevant. They have differences in their origin, to whom they are exposed to, and the data models they use, but many SOA "services" could also be potentially described as internal APIs.
SOA

- SOA exposed services versus/and APIs
Microservices

• offer a better way to decouple components *within* an application boundary

• despite being broken into separate microservice components on the inside, the application might still look the same from the outside.

• The number and granularity of APIs that a microservice-based application exposes should not be any different than if the API was built as a siloed application.

• The prefix "micro" in *microservice* refers to the granularity of the internal components, not the granularity of the exposed interfaces.
Microservices

Monolithic application

- Exposed services/APIs
- Silo component

Microservices application

- Exposed services/APIs
- Microservice component
- Microservice component
- Microservice component
Microservices

- From monolithic applications to microservices
Microservices | Benefits

• **Agility and productivity**
  The team that is developing the microservice can completely understand the codebase. They can build, deploy, and test it independently of other components in much faster iteration cycles.

• **Scalability**
  The microservices development team can scale the component at run time independently of other microservice components, enabling efficient use of resources and rapid reaction to changes in workload.

• **Resilience**
  With a carefully decoupled design, such as avoiding synchronous dependencies and using circuit breaker patterns, each microservice component can be written to satisfy its own availability requirements without imposing those requirements across the application domain.
Microservices | Key factors to consider when choosing microservices

• **New patterns of technology**
  – Because they are on the network, they require a whole new set of components on the network alongside them. The enabling technologies exist, including service discovery, workload orchestration, container management, and logging frameworks.

• **Application suitability:**
  – *e.g.* One recommendation is not to use microservices until the evolution of a traditionally written application starts to reach an inflexion point of complexity.
  – Accept eventual consistency models, rather than the transactional interactions that you are used to.
  – Understand how to work with event-sourced applications with no central operational data store.
Microservices | **Key factors to consider when choosing microservices**

- Ensure that your application logic is state free if it needs to take advantage of the significant rapid scalability benefits.
- Become familiar with the subtle potential side effects of asynchronous communication if you decouple yourself from downstream components.
- Understand the logic implications of implementing circuit breaker patterns.
- Recognize the error handling limitations of HTTP/JSON communication compared to in-process communication.
- Consider network latency in chained interactions.
SOA, Microservices in the APIs world

- From point-to-point integration architectures (where sometimes the integration code was written in both sides of every integration) we evolved towards EAI systems (Enterprise Application Integration).

- Adapters can have multiple roles: connection and data handling, transport management, protocol conversion, while Integration Hub handles aspects such as: routing/distribution, data translation, composition.

Figure: Hub and spoke architecture
SOA, Microservices in the APIs world

- All these functionalities could gain shape in the early 2000’s, at the same time when the standards for web services appeared. Services using SOAP/REST [AB06] could be invoked and therefore exposed business function becomes reusable.

- When a service is **exposed**, more than a protocol or data format are necessary:
  - Virtualization - a client sees just a virtual service endpoint; behind the scene complex operations are performed: conversions of various data formats, routing, conversion to standard protocols; any changes at these levels should be transparent for the consumer
  - Security - mechanisms that ensure good security but at the same time are reducing the number of security models for the client, are desirable
  - Monitoring - there is a wide set of contexts in which monitoring allows decision-making: if system business functions are exposed you need to monitor them, if you want to control the traffic in order to prioritize the service access for priority clients

=> the Integration Hub could be an ESB in a SOA architecture, being used on the Integration level 😊
SOA, Microservices in the APIs world

• The service exposure is a difficult problem, because it must approach issues such as: service discovering, the used data format, security approaches if there are internal users (inside a company) and also external users of the provided services.

• In the context of Web growth (mobile, IoT) SOA should face not just requirements from an enterprise but unknown ones, numerous consumers and a competitive marketplace

• Furthermore, the consumers must be able to seamlessly access data and applications (e.g. by exploiting API’s through browser-based applications).

![Service Exposure Diagram](image-url)
SOA, Microservices in the APIs world

- If the SOA services (and old APIs) were mainly exposed inside the enterprise, and the SOA played a key role for efficiency in an enterprise boundaries (Figure from Previous Slide), currently the services are available to external customers, and thus the APIs became a capitalisable product.

**Figure: Exposing Services inside/outside of the enterprise**
SOA, Microservices in the APIs world

- APIs exposure required a new modified architecture and therefore the software architects have created a new management level, namely the API Management which may be conducted on enterprise level or may employ a partner.

- API Management provides mechanisms to expose and manage securely a web API.

Figure: Exposing Services enhanced with an API Management service
SOA, Microservices in the APIs world

• API Management is able to provide:
  – security (e.g. support against DOS attacks, XML threats or monitor partner applications to control their services)
  – an API Gateway (that exposes the API offered by a provider, exposes a new integration in case of enterprises with no SOA based architectures or simple hub-and-spoke architectures)
  – track interactions with partner applications (e.g. if the supplied API is part of an external business a strict monitoring of services levels should be provided, to control that established agreements of API use are respected)

• The API management is made through API gateway by dissociating capabilities and externalising API exposure to viable partners such as cloud providers

• Issues?
  – latency (instead of one step, the client's request is going through two steps; as a solution a caching mechanism can be employed)
  – privacy and security (are you willing to give someone access/”control” of your internal structure?)
SOA, Microservices in the APIs world

- In real life, there are two possible situations:
  - a company designing from scratch a microservice oriented architecture
    - In this case, the architectural challenges reside in the way the microservices are designed, and the line separating the SOA (meant to integrate and to provide more meaningful “services”) and microservices is blurred.
  - older system aiming at capitalising on novel technologies and are system architectures that have a hybrid SOA and API Management approaches
    - the internal consumer requests are directly fulfilled using an API Gateway (proprietary or given as a service) that benefits from an existing SOA architecture or in worst case creates a services level from a siloed application
    - There is also an API Management level, that is layering in cloud or on-premise, providing all features we have previously discussed about.
SOA, Microservices in the APIs world

• In this environment, the SOA plays a crucial role by modeling back-end systems data and functions into “something” useful for business requirements. It is sufficient to think of the mobile world to understand how traditional applications’ architectures have been changed.

• The microservice application and API gateway depend on the manner the SOA conducts the re-modelling, in order to comply to requirements such as: availability, scalability, responsiveness or agile change.

Figure: Enterprise and APIs and microservices
API Management | Solutions

- **Amazon AWS’s API Gateway**
  - AWS’s API Management solution is called API Gateway [AAG18].
  - It serves as a single point of interest for creating, maintaining and monitoring APIs.
  - It offers the possibility to set up throttling policies, access control management and even API versioning.
  - The APIs created with API Gateway can have many back-ends, such as code running directly on EC2 instances, web applications deployed in Elastic Beanstalk, but what makes API Gateway shine is the way it works with AWS’s serverless solution, AWS Lambda

- **IBM’s API Connect**
  - IBM’s solution to API Management [IAC18]
  - offers the same core functionalities as the previous solution: automatic API creation, analytics, security enforcement etc.
  - apart from the other solutions is the fact that besides being available for on-premise use, API Connect is also available to be used cross-cloud
API Management | Solutions

Apigee’s Edge

- Edge [AE18] is a solution that allows developers to create proxy APIs that will serve as an abstraction layer between the clients and the backend service.
- Edge can add throttle rates, improve security and monitor inbound traffic.
- Apigee’s Edge can also be purchased to be used in a on-premise Cloud, besides the cloud managed version.
- The main advantages of Apigee’s solution are the analytics it provides, making it easier to monetize the traffic.

Figure: Example - Edge handling request for various backend services [AE18]
For **Infrastructure automation** we list below several already existent solutions:

- **Amazon AWS’s Cloud Formation**
  - Cloud Formation allows the automatic creation of every possible AWS resource. Whether we are talking about provisioning 15 EC2 instances, a Virtual Private Cloud (VPC) for them to reside on and a Direct Connect to extend your on-premise cloud, or simply templating the machines the team mates will use for development.
  - Using Cloud Formation, the infrastructure becomes code, that can be checked in source control and can be peer reviewed, not to mention being reproducible for future uses.

- **RedHat’s Ansible Tower**
  - helps you scale IT automation, manage complex deployments and speed productivity
  - Centralize and control your IT infrastructure with a visual dashboard, role-based access control, job scheduling, integrated notifications and graphical inventory management
  - Ansible Tower's REST API and CLI make it easy to embed Ansible Tower into existing tools and processes.

- **Chef’s Chef**
  - Chef provides an easy to learn DSL that lets you define your infrastructure as code. This makes your system configurations more testable, portable, and auditable then with traditional run books or shell scripts, giving teams the freedom to focus on building new innovations, rather than re-solving old problems [ClInf8].
  - Chef requires an agent that runs on each provisioned host. The agent contacts the Chef Master Server and retrieves a Cookbook, which it executes to get in the desired state.
API Management | Solutions

For Deployments and Continuous Integration we list below several already existent solutions:

- **Jenkins**
  - Jenkins [Jen18] has the advantage of being an open source project as well as the multitude of plugins available. It is written in Java and can be used to automate part of the development process, specifically operations that happen to achieve continuous deployment.
  - Jenkin’s builds can be triggered multiple ways, but the most used are when a commit is pushed in a source control repository (e.g. Git, SVN etc.) or can be scheduled to run on certain periods.

- **Travis CI**
  - Travis CI [TCI18] is another well known solution to the continuous integration problem. It is one of the oldest hosted solutions, but it is also available for on-premise use. It is free for any open source projects on GitHub.

- **CodeFresh**
  - CodeFresh [Pec18] is a peculiar solution, being conceived with only one thing in mind: working with Docker. It offers sever templates, allowing you to start using it straight away.
  - An interesting aspect of CodeFresh is that after the build phase, it allows you to preview the image that has just been built, offering a staging like feature, without the provisioning of a new instance.
API Management | Solutions

For **Monitoring** we list below several already existent solutions:

- **Amazon AWS CloudWatch**
  - A solution from AWS’s offerings, CloudWatch [ACW18] has the advantage of being deeply integrated with the other AWS services. CloudWatch can be used to keep track of system specific metrics, such as CPU utilization, memory usage etc., but also keep track of log files written by the application. Additionally, it supports creating alarms on top of the metrics and defining custom actions that will be taken if the alarm threshold is breached.

- **Azure Monitoring**
  - Azure Monitor is a platform that has similar functionalities such as CloudWatch. Azure Monitor has the ability to monitor Azure resources (collect performance and utilization data, activity and diagnostics logs, and notifications from your Azure resources) [AzM18].

- **Google StackDriver**
  - Stackdriver enables, beside already standard functions, combining logos, metrics and metadata of multiple systems, be it in cloud or on-premise [GooS18]
Summary

• SOA and Microservices in the same picture?
• SOA Dichotomy
• Microservices
• SOA, Microservices in the APIs world
Bibliography

Kim Clark, Microservices, SOA, and APIs: Friends or enemies?

Martin Fowler, Microservices - a definition of this new architectural term


“Alexandru Ioan Cuza” University of Iasi
Faculty of Computer Science

Questions?