**Abstract**

Microservice is a relatively new paradigm for building applications based on an architectural design style whereby applications are made of loosely coupled components that are independently designed and maintained from each other yet working with each other to provide a function. The benefits that microservices introduce have made them very popular and as a result the number of microservices that are been built and made available as open source as well as proprietary is increasing by the day. The downside of that as it tends to be the case with any new thing and technology in particular always is that it does introduce some complexity. For instance, microservices are still very much loosely defined not just in the way they are architected but also functionally making it difficult if not impossible to group them into common patterns be it by functions, design approach, technology or a combination of identical dimensions which could render their discovery and composition easier, faster and more efficient. This paper attempts to define a taxonomy for microservices by looking into public repositories of microservices and identify common patterns in their design, use cases and interoperability. We then look at how microservices could be classified and discovery using this taxonomy.

**Introduction**

Garriga (2018) define Microservices as a novel architectural style that tries to overcome the shortcomings of centralized, monolithic architectures, in which application logic is encapsulated in big deployable chunks. In contrast, microservices are small components, built around business capabilities, that are easy to understand, deploy, and scale independently, even using different technology stacks. Each running in a dedicated process and communicating through lightweight mechanisms, often a RESTful API.

Namiot & Sneps-Sneppe (2014) introduce the microservices approach as a relatively new term in software architecture patterns which consists in developing an application as a set of small independent services with each service running in its own independent process. While it can be argued that microservices are still a relatively new paradigm in the software industry, it has to be noted that five years have passed since the researchers published their work and five years in the IT industry is a long time. Today, the fact of the matter is that the adoption of microservices is growing at a rapid pace as many businesses are looking to move away from monolithic approach of designing applications which is known to significantly slowdown application development cycle and lead to very risky and costly maintenance and upgrades. With microservices instead businesses are looking to take advantages of new computing paradigm such as cloud, DevOps, Continuous integration, continuous delivery (CI/CD) to name but a few.

There are obviously many definitions of microservices by many researchers and they all seem to convey a common view of the attributes of microservices which include “loosely coupled”, “independently developed, deployed and maintained”, “uses lightweight communication”, “small in size”. That being said having that many definitions for the same thing albeit their commonalities, can still create some confusion and render their understanding complex. To that end, one way that I like to think of microservices that would simplify their understanding is by using the analogy of a multi-manufacturer machinery. All the parts of a particular machinery designed to provide a given function are made by different manufacturers and assembled to work together to produce a particular type of end product such as cars, bicycle, fridge, microwave, TV, etc. If we take the example of cars, then we would be talking about a car fitted with an engine that is made by say Toyota, while the doors are the work of BMW, the bumpers built by an unknown brand and so on. While this provides the advantage that each manufacturer can work independently from one another and make changes and improvements as they wish without necessarily having to wait for other teams that are designing and building other parts. In the context of microservices that is what Chen (2018) is alluding to in his paper when he talks about the benefits of continuous delivery (CD) that microservice introduce which basically allows team to produce software reliably in short release cycle and at any time. The downsides of that however is that it could lead to duplicate efforts as different manufacturers may end up building the same part leading to wasted time, and resources. Furthermore, in the assembly process of the car some preassembly work could done already by assembling two or more compatible parts into one.

While such approach would offer a plethora or options to design machinery from and as such provide more options in terms of functions that could be achieved. It also makes it very complex and time-consuming task to assemble a particular end product capable of fulfilling a given function and for microservices that’s precisely why Chen (2018) while presenting the benefits of microservices was equally quick to remind us that they are not silver bullet as their adoption does introduce complexities and challenges which if not managed properly could lead to another problematic situation. Going back to our analogy, one would for instance have to search through all types of parts to assemble a car including non-related ones such as parts designed for microwaves or perhaps invest resources and money into building new ones which would end up duplicating what already exists leading to waste of financial resources. One way to address such issue would be to identify common attributes that the parts share so as to be able to group them into family of parts. That way we would have parts for microwaves grouped together and clearly distinct from say parts for freezer, cars and so one. It is obvious that the process would be faster if say one only has a way of searching a repository with the intent to retrieve only for car parts if they are looking to assemble a car. Similarly, there would be no need to search for fridge and freezer parts when looking to assemble a microwave.

While the analogy used here helps simplify the understanding of microservices paradigm, it does also present a number of challenges associated with microservices notably two of them which are the challenge related with searching for a particular type of microservice and where exactly to search for it. For the former this could be addressed by way of a rigorously designed taxonomy to help identify common patterns or attributes that microservices share which could be used for their classification and this is precisely what we attempt to do in this paper which is organized as follows: in the first section we will look into existing taxonomy of microservices, in section 2 we dive into their advantages and shortcomings and in section 3 we provide details of the proposed taxonomy before concluding and looking at opportunities for further research in section 4.

**Previous work on microservices taxonomy**

A quick search on google scholar on microservices taxonomy return a number of results for which the most significant are the work done by Garriga (2018) who already had done some similar work classifying web services in the context of SOA. For their work on microservices taxonomy, the researcher basically based his study on 64 relevant articles on microservices which he found from searching popular online libraries including Scopus, Science Direct, IEEE Xplore, ACM Digital Library, SpringerLink, Google Scholar and Wiley Online. He then conducted a detailed qualitative analysis on those 64 articles which reduced the list down to 46 articles which he basically classified into two main groups: primary studies, that is, literature investigating specifically their research question (using microservices, proposing microservice-oriented frameworks, tools or architectures) and 28 out the 46 articles were marked as being part of that group; and secondary studies for the remaining 18, that is, literature reviewing primary studies (surveys, reviews and comparative studies assessing microservices or microservice-based approaches). He classified microservices based on 6 dimensions including design, implementation, deployment, runtime, crosscutting concerns and organizational aspects which they further classify into 18 sub-dimensions

The classification done by Garriga (2018) is mostly based on non-functional requirements making it difficult to find microservices based on the functionalities they are intended to fulfil. Newman (2015) explains that microservices are primarily modelled around business domains which helps avoid problems related to traditional tiered architectures. For instance, say microservices providing logging or monitoring functionalities. This paper aims to produce a taxonomy for microservices to help with their classification in a fine-grainer way that would account for both functional and non-functional attributes of microservices. For testing the taxonomy, we use an examples of microservice taken randomly from amazon Serverless architecture repository <https://serverlessrepo.aws.amazon.com/applications> or by running a search on GitHub using the keyword microservice.

**Taxonomy of microservices. A new framework**

This research paper proposes a new framework for a more fine-grained taxonomy of microservices. Previous work done by Garriga (2018) have shown some limitations in classifying web services. For instance, if we consider the following webservice description which could also be an application composed of multiple microservices *“An application that provides a real time report of all the meeting rooms that have been booked but which are not utilized”* how would one go about retrieving discrete microservice components that make of the overall microservice to provide the sought functionality?

In this framework we propose using the keywords and map them to specifics function. In the case of the example above we have “real time report” will be mapped to reporting functionality while “all the meeting rooms” will indicate some sort of “querying” functionality. We also have “booked but not utilized” which corresponds to a state would most likely map to a “monitoring” functionality. last but not least the keywords “meeting room” that in itself could constitute a function such as a microservice for meeting rooms management. See table below for details of full mapping

|  |  |
| --- | --- |
| **Keyword** | **Function** |
| Looking for | Search Monitoring Ingestion |
| Real time | Monitoring/Reporting |
| Report | Display/Reporting |
| Meeting rooms | Meeting rooms management |
| Booked/Not utilized | Monitoring/Search/meeting rooms management |

Table 1. Keywords-to-function mapping table

A quick search on GitHub, Google or Amazon Serverless Repository returns over 10,000 results for microservices and out of those results I looked at hundred or so microservices and was able to identify non-exhaustive list of functions listed below as functions fulfilled by microservices:

* Search
* Monitoring
* Logging
* Authentication
* Encryption
* Compression
* Reporting
* Data ingestion
* Upload

This approach of classifying microservices by function aligns very much with Newman (2015) when he talks about focusing service boundaries on business boundaries to help identify code that provide a given piece of functionality.

Dragoni et al. (2017) define microservices from technical point of view microservices as independent components conceptually deployed in isolation and equipped with dedicated memory persistence tools (e.g. databases). They argue that since all the components of a microservice architecture are microservices, its distinguishing behaviour derives from the composition and coordination of its components via messages. This indicates that microservices could be classified by the technical attributes of their components. By following that approach, this framework uses technology-related attributes as second type of dimensions for identifying microservices. Microservices can therefore be classified say based on the programming languages used for their development e.g. scripting, object-oriented, etc. Their communication protocols such as Restful/http(s), MQ, xml, etc. The data store including relational databases, NoSQL databases. The other technical dimensions are data transfer methods (push/pull, sync/async, streaming); the platforms and the deployment method.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Programming Languages** | **Communication** | **Data store** | **Data transfer** | **Platform** | **OS** | **Deployment** |
| Scripting  Object-oriented  Procedural  Other | Restful/http  MQ  RPC  XML | Relational DB  NoSQL DB  Graph  Object  Flat files | Push  Pull  Sync  Async  Streaming | Virtual machines  Physical servers  Containers  Serverless | Windows  Linux | Public cloud  Private Cloud  On-premise  Hybrid |

Table 2. Technical dimensions table

Programming languages and data types can be further classified by programming language including Python, JavaScript, PHP, Go, Ruby, html, C#, Java and the type of data being manipulated such as text, video or voice.

Table 3. Programming languages & Data types

|  |  |
| --- | --- |
| **Languages** | **Data Type** |
| Python  Javascript  Java  PHP  Go  Html  C# | Text  Numbers  Video  images  Voice  Mixed |

The other dimensions the framework looks at is the dependencies and or compatibilities in other words what are the microservices that would require one or more other microservices in order to fulfil a given function (dependency) or is there some microservices that cannot work together for one reason or another (compatibility)

Figure 1. Microservices taxonomy

As shown in the above figure, the taxonomy is made of functional and non-functional dimensions. The boxes on the left side are the function types that basically serves for the classification of microservices. On the right side we have the non-functional dimensions which will be weighted to help determine which function(s) type(s) a microservice fits into. For instance, for logging type of microservices it is expected that the key component would be a data store and possibly flat files or some object store for storing log files and as such those dimensions should carry the highest weight. Let’s take a look at both type of dimensions in more details in the following section.

**Non-functional dimensions**

* **Deployment**

This dimension indicates whether the microservices can be deployed and run on a server located in a data centre on-premise refer to as “local” in the diagram. Or whether, the microservices needs to be run in the cloud which is further subdivided into private, public and hybridtype. Also, for the deployment dimension, microservices could be further classified based on the type of compute service used to run them including physical, virtual machine or inside a container such as docker or kubernetes, or in a dynamically powered server by an event also known as serverless.

* **Technology**

From a technology standpoint, microservices can be classified based on the type of programming language that they are developed on. For instance, Java-based or C++ microservices for object-oriented types, Javascript for scripting language and so on.

The second sub-dimension under technology is the communicating protocols used for when communicating with other microservices. A lightweight API gateway based on HTTP/RESTful could be used or a message transfer protocol such as MQ could be used as its name implies for transferring messages between microservices. The other option might be a remote procedure call whereby a microservice would trigger the execution of a function call on a remote server.

The next sub-dimension under technology is whether the execution done synchronously or asynchronously. This subdimension is somewhat tightly related to the communication protocols and are sometime used interchangeably. In reality they designate two distinct concepts albeit the close relation between those concepts. In general, messaging queuing tend to be more suitable for asynchronous execution type whereas synchronous execution mode tends to work with lightweight protocols such as REST but that is not always the case.

The other type of dimension type of technology subdimension is the data store. Microservices could be designed to use relational, non-relational data store or even flat files for storing data. Relational data stores are more suitable for microservices that are expected to manipulate structured data and such data could be accessed using standard SQL queries. Some microservices are designed to work with non-relational data store in order to address specific use cases that require handling unstructured data, and for that various options are available including key-value, document or graph databases which typically can store larger amount of data than relational databases as noted by Vicknair (2010). Another option for non-relational database are object storage type such as amazon s3 typically used for data lake given the amount of data it can store. Data could also be stored in files such as csv or text files although this method is less and less utilized perhaps due to the variety of databases for structured and unstructured data available today as open source and also the continued drop in price of the COTS (Commercial-Off-The-Shelf) ones.

Another key technology dimension for microservices is the type of data that is processed by the microservices and this include input and output data which could be in the form of texts, numbers, voice (such as amazon Alexa or google assistant), videos or images or in some cases a combination of two or more data types which we classify in the taxonomy as mixed data type.

The other technology dimension that the framework proposes using for classifying microservices is security and for that we’ve identified four categories including microservices using client side certificates and the ones using server side certificates on one hand, and on the other hand microservices using some kind of local directory such as Microsoft Active directory or LDAP to authenticate users or applications and microservices using identify federation for authenticating users which could be based on technologies such as SAML, OAuth or OpenID.

**Description of the taxonomy**

In our framework microservices are classified based on two main dimensions that is functional and non-functional dimensions. For a fine-grained classification we use a weighting mechanism of non-functional dimensions to be able to classify microservices by type of functionality tor functionalities they are designed to provide e.g. Logging, Authentication. The weighting ranges from between 0 and 5 with zero being the lowest weight meaning that any dimension with a weight of zero has very little or no influence in the classification by functionality of the microservice. The deployment dimension carries the lowest weight while a weighting of 5 means the dimension carries the highest weight which indicates that the dimension is the most important one in helping classifying microservices with that particular dimension. For example, an authentication microservice will have a weight of 1 for the deployment dimension for most microservices tend to be platform-agnostic given that they can run both on premise and in the cloud either by installing them directly on a server instance or by deploying inside a container. The table below shows how weighting is applied to non-functional dimensions.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Non-functional  Functional | Deployment | | | | | | | | | | |
| Cloud | | | | | | | | | Local/on-premise | |
| Public | | | Private | | | Hybrid | | | Physical | Virtual |
|  | Serverless | containerized | Virtual/Physical | Serverless | containerized | Virtual/Physical | Cloud | On-premise | Serverless |
| Upload | 3 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| Search | 3 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| Monitoring | 3 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| Logging | 3 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| Reporting | 3 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| Authentication | 3 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| Encryption | 3 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |

Table 4. Weighting of non-functional dimensions: Deployment

From the above table we can see that the most influential dimension for classifying microservices from a deployment perspective is the serverless characteristic which can be implemented in the cloud (public, private). Serverless is a relatively new concept that allows running an application piece of code without the need to operate or manage the server that it is deployed on. It is an approach which Castro (2017) qualifies as function as a service that makes life easier for the developer who only has to write the code and not worry about the underlying infrastructure. The dynamic nature of cloud makes it a good platform for serverless-based microservices.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Non-functional  Functional | Technology | | | | | | | | | | | | | |
| Programming languages | | Communication protocols | | | Execution mode | | Data store | | | Security | | | |
|  | Scripting | Object-oriented | Restful | RPC | Messaging | Sync | Async | Relational | Non-relational | Flat files | SSL/TLS Certificate | API gateway | Local directory | Identity Federation |
| Upload | 2 | 2 | 3 | 3 | 1 | 2 | 2 |  |  |  | 1 | 1 | 1 | 1 |
| Search | 2 | 2 | 3 | 3 | 1 | 3 | 1 |  |  |  | 1 | 1 | 1 | 1 |
| Monitoring | 2 | 2 | 3 |  | 1 | 1 | 3 |  |  |  | 1 | 1 | 1 | 1 |
| Logging | 2 | 2 | 3 | 3 | 1 | 1 | 3 |  |  |  | 1 | 1 | 1 | 1 |
| Reporting | 2 | 2 | 3 | 3 | 1 | 3 | 1 |  |  |  | 1 | 1 | 1 | 1 |
| Authentication | 1 | 1 | 2 | 2 | 1 | 3 | 1 |  |  |  | 5 | 3 | 5 | 5 |
| Encryption | 1 | 1 | 1 | 1 | 1 | 2 | 1 |  |  |  | 5 | 3 | 1 | 1 |

Table 4. Weighting of non-functional dimensions: Technology

Now let’s take a look at the technology dimensions we can see from the table above that Upload type of microservices would mostly use RESTful or RPC communication protocol type while they could execute synchronously or asynchronously. Zimmermann (2017) cites RESTful HTTP as one of the most widely used communication protocol for microservices designed to help overcome the limitations of Service Oriented Architecture (SOA). The rest of the dimensions have very little influence in terms of where to classify the microservice hence their weighting of 1 except for the programming language that has a weight of 2 but still does not differentiate the microservice as both language types (scripting and object-oriented) carry an equal weight of 2. It is pretty obvious that microservices providing security functionality such as authentication and encryption have the highest weight on SSL/TLS certificate, API gateway, local directory and identity federation dimensions

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Non-functional  Functional | Technology | | | | | | | | | | | |
| Data store | | | | | | | | | | | |
| Relational | Non-relational | | | | | Flat files | Data types | | | | |
| Key-value | Document | Column | Graph | Object | Numbers | Text | Voice | Videos/images | Mixed |
| Upload | 2 | 3 | 4 | 3 | 2 | 4 | 1 | 1 | 1 | 1 | 3 | 2 |
| Search | 4 | 4 | 2 | 3 | 4 | 3 | 2 | 2 | 2 | 1 | 3 | 1 |
| Monitoring | 3 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 1 | 1 | 1 |
| Logging | 2 | 3 | 1 | 2 | 2 | 4 | 4 | 1 | 1 | 1 | 1 | 1 |
| Reporting | 4 | 3 | 1 | 3 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| Authentication | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Encryption | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 5. Weighting of non-functional dimensions: Technology/Data Store

In the table above, we zoom into the data store dimension and we can see that microservices that use relational database in general would fall into the group of microservices providing search or reporting function while the logging functionality is mostly provided by microservices that are designed to work with flat files or object storage as data store.

**Classification of microservices using the taxonomy**

Let’s take as an example the microservice: microservice-http-endpoint published on Amazon Web Services (AWS) serverless Application Repository (SAR) with the code available in awslabs GitHub repository <https://github.com/awslabs/serverless-application-model/tree/master/examples/apps/microservice-http-endpoint>. It is a basic microservice that performs a read and write to and from a non-relational database via a Restful API using amazon API gateway. This microservice is serverless by design as it does not require having deploy or manage any server for running it, instead it’s been developed in such a way that all the infrastructure resources that it requires (in this case some Amazon cloud services) are referenced in the form of code within a template file written in yaml or json format. These infrastructure resources will be spun up when the code runs which effectively means when the microservice is invoked. Let’s try to understand the most important parts of the contents from template file below:

AWSTemplateFormatVersion: '2010-09-09'

Transform: 'AWS::Serverless-2016-10-31'

Description: >-

A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.

Parameters:

TableNameParameter:

Type: String

Resources:

microservicehttpendpoint:

Type: 'AWS::Serverless::Function'

Properties:

Handler: index.handler

Runtime: nodejs6.10

CodeUri: .

Description: >-

A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.

MemorySize: 512

Timeout: 10

Policies:

- DynamoDBCrudPolicy:

TableName: !Ref TableNameParameter

Events:

Api1:

Type: Api

Properties:

Path: /MyResource

Method: ANY

The line of code Transform: 'AWS::Serverless-2016-10-31' tells the interpreter that this is a serverless microservices which contains a number of resources that will be deployed as code as with the microservice. Those resources are specified under Resources section. We can see that a lambda function - which is a compute service from Amazon Web Services that let you run your code without provisioning or managing servers – is listed as resource using the following line of code:

Type: 'AWS::Serverless::Function'. The code for this particular lambda function is written in nodejs6.10 which is precisely what the following line of code does Runtime: nodejs6.10. The other resources are DynamoDB which is amazon managed non-relational database service and an API gateway, a managed API gateway service. The codes for both resources are shown below:

- DynamoDBCrudPolicy:

TableName: !Ref TableNameParameter

Events:

Api1:

Type: Api

Properties:

Path: /MyResource

Method: ANY

We can see that this microservice is heavily tied to a cloud platform and it can only be deployed without being modified to Amazon cloud. Also, the microservice is of serverless type which explains why the serverless on cloud dimension would carry more weight from a deployment perspective if we were to classify this microservice. From a communication protocol perspective, the microservice is making Restful api calls as indicated in the code. As there is no data to tell whether the microservice execute synchronously or asynchronously nor any indication on the security technology used if any we will give those dimensions a weight of 0. This leaves us with the data store dimension to finalize our classification. The microservice uses DynamoDB which is a key-value data store which at this point makes it falls under a Search or Monitoring type of microservice. However, we will look further into the type of database action this eliminates the possibility of this microservice being classify as Search type for search tend to perform read only action. This effectively means that the microservice-http-endpoint having scored high on serverless (no server to deploy/manage), cloud (Amazon public cloud), Restful api call, key-value data store (Amazon DynamoDB) with read and write database actions will highly likely performing a monitoring function hence fall into the Monitoring category of microservice.

**Summary and future work**

In this paper we look at the existing taxonomy of microservices and how they can be used to classify microservices. Next, we discussed the limitations of the existing taxonomy which introduced us to the newly proposed framework. We explained how microservices could be classified by function they provide based on the technology dimensions. Each dimension is assigned a weighting which basically is a number between 0 and 5 with 0 representing the lowest measurement which indicates that the dimension has little or no influence over the classification of the microservice while 5 is an indication that a dimension is the most important in helping determine which type of function the microservice delivers hence should be classify as. We discussed the different dimensions that can be used which are technology and deployment related dimensions. The deployment dimensions were subdivided into Cloud and local/on-premise which were further divided into 10 subdimensions including private, public, hybrid cloud, virtual/physical, container, serverless to name but a few. The technology dimensions were identified as the key ones for helping classify microservices under our taxonomy and they include programming languages, communication protocols, execution mode, security, data stores, data types, database actions and more. To test the taxonomy, we then used an example of microservice taken randomly from amazon serverless repository and walked through the process of classifying microservice using our framework. While this taxonomy could be regarded as a significant step forward for helping classify microservices, more work can still be done to improve the taxonomy. For instance, we could look at the code level to try to identify common patterns that microservices share which eventually could be accounted for in their classification. We could also look at things like frameworks, packages and libraries that can help identify microservices. Another dimension that could be looked at is the compatibility of microservices in other words look at a way of classifying microservices based on their ability to work with each other to provide one of more functions.

**References**

Andrews, B. and Kumar Ramesh, S. (2019). *awslabs/serverless-application-model*. [online] microservice-http-endpoint. Available at: <https://github.com/awslabs/serverless-application-model/tree/master/examples/apps/microservice-http-endpoint> [Accessed 15 May 2019].

Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2017). Serverless Programming (Function as a Service). *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. doi:10.1109/icdcs.2017.305

Chen, L. (2018). Microservices: Architecting for Continuous Delivery and DevOps. 2018 IEEE International Conference on Software Architecture (ICSA). doi:10.1109/icsa.2018.00013

Dmitry Namiot, Manfred Sneps-Sneppe (2014). On Micro-services Architecture. International Journal of Open Information Technologies ISSN: 2307-8162 vol. 2, no.9, 2014

Dragoni N. et al. (2017) Microservices: Yesterday, Today, and Tomorrow. In: Mazzara M., Meyer B. (eds) Present and Ulterior Software Engineering. Springer, Cham.

Garriga M. (2018) Towards a Taxonomy of Microservices Architectures. In: Cerone A., Roveri M. (eds) Software Engineering and Formal Methods. SEFM 2017. Lecture Notes in Computer Science, vol 10729. Springer, Cham

S. Newman (February 2015), Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 1st edition

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., & Wilkins, D. (2010). A comparison of a graph database and a relational database. *Proceedings of the 48th Annual Southeast Regional Conference on - ACM SE 10*. doi:10.1145/1900008.1900067

Zimmermann, O. Computer Science Research & Development (2017) 32: 301. <https://doi.org/10.1007/s00450-016-0337-0>

**Appendix**

Sample of microservices

|  |  |  |
| --- | --- | --- |
|  | **Microservice** | **Reference** |
| 1 | Alexa-smart-home-skill-adapter | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~alexa-smart-home-skill-adapter> |
| 2 | Datadog-Log-Forwarder | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:464622532012:applications~Datadog-Log-Forwarder> |
| 3 | SecretsManagerRDSMySQLRotationMultiUser | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:297356227824:applications~SecretsManagerRDSMySQLRotationMultiUser> |
| 4 | image-resizer-service | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:526515951862:applications~image-resizer-service> |
| 5 | alexa-skills-kit-color-expert-python | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~alexa-skills-kit-color-expert-python> |
| 6 | SecretsManagerRDSPostgreSQLRotationSingleUser | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:297356227824:applications~SecretsManagerRDSPostgreSQLRotationSingleUser> |
| 7 | alexa-skills-kit-nodejs-howtoskill | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:173334852312:applications~alexa-skills-kit-nodejs-howtoskill> |
| 8 | SecretsManagerRDSMySQLRotationSingleUser | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:297356227824:applications~SecretsManagerRDSMySQLRotationSingleUser> |
| 9 | alexa-skills-kit-nodejs-triviaskill | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:173334852312:applications~alexa-skills-kit-nodejs-triviaskill> |
| 10 | alexa-skills-kit-nodejs-factskill | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:173334852312:applications~alexa-skills-kit-nodejs-factskill> |
| 11 | Hello-world | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~hello-world> |
| 12 | microservice-http-endpoint | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~microservice-http-endpoint> |
| 13 | Image resizer service | <https://github.com/cagataygurturk/image-resizer-service> |
| 14 | Uploader service | <https://github.com/evanchiu/serverless-galleria> |
| 15 | Image processing service | <https://github.com/awslabs/serverless-application-model/tree/master/examples/apps/image-processing-service> |
| 16 | Simple Contact-us form | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:318300609668:applications~simple-contact-us-handler> |
| 17 | Cryptomonitor | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:713541911133:applications~crypto-monitor> |
| 18 | Step-function-send-to-sns | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~step-functions-send-to-sns> |
| 19 | SQS poller | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~sqs-poller> |
| 20 | Alexa random-restaurant | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:934629381695:applications~alexa-random-restaurant> |
| 21 | Uploader | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:233054207705:applications~uploader> |
| 22 | https-request | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~https-request> |
| 23 | Event-stream-processing | <https://github.com/kbastani/event-stream-processing-microservices> |
| 24 | Microservices-kubernetes | <https://github.com/ewolff/microservice-kubernetes> |
| 25 | Asynchronous HTTP microservices | <https://github.com/zeit/micro> |
| 26 | Customer stores | <https://github.com/spring-cloud-samples/customers-stores> |
| 27 | Front-end application for ALL microservices | <https://github.com/microservices-demo/front-end> |
| 28 | alexa-skills-kit-python36-factskill | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:173334852312:applications~alexa-skills-kit-python36-factskill> |
| 29 | Datadog-RDS-Enhanced | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:464622532012:applications~Datadog-RDS-Enhanced> |
| 30 | SecretsManagerRDSPostgreSQLRotationMultiUser | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:297356227824:applications~SecretsManagerRDSPostgreSQLRotationMultiUser> |
| 31 | standard-redirects-for-cloudfront | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:621073008195:applications~standard-redirects-for-cloudfront> |
| 32 | alexa-skills-kit-color-expert | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~alexa-skills-kit-color-expert> |
| 33 | api-lambda-save-dynamodb | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:375983427419:applications~api-lambda-save-dynamodb> |
| 34 | hello-world-python3 | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~hello-world-python3> |
| 35 | serverless-todo | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:233054207705:applications~serverless-todo> |
| 36 | dynamodb-display | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:251940851769:applications~dynamodb-display> |
| 37 | dynamodb-elasticsearch | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:831212071815:applications~dynamodb-elasticsearch> |
| 38 | api-lambda-send-email-ses | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:375983427419:applications~api-lambda-send-email-ses> |
| 39 | alexa-skills-kit-nodejs-premium-facts-skill | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:173334852312:applications~alexa-skills-kit-nodejs-premium-facts-skill> |
| 40 | cloudfront-response-generation | <https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:077246666028:applications~cloudfront-response-generation> |
| 41 | Online Table Reservation System | <https://www.codementor.io/packt/how-to-implement-an-online-table-reservation-system-with-microservices-kw2iacz17> |
| 42 | Sample logging microservice | <https://github.com/mohhmekk/sample-logging-microservice/blob/master/src/main/java/org/sample/assignment/log/MongoAppenderInitializer.java> |
| 43 | sample-spring-microservices-new | <https://github.com/piomin/sample-spring-microservices-new> |
| 44 | Distributed log system | <https://github.com/leoChaoGlut/log-sys> |
| 45 |  | <https://github.com/search?q=microservice+logging> |
| 46 | Microservices demo | <https://github.com/GoogleCloudPlatform/microservices-demo> |
| 47 | Sock shop | <https://github.com/microservices-demo/microservices-demo> |
| 48 | Asynchronous HTTP microservices | <https://github.com/zeit/micro> |
| 49 | Cinema microservice | <https://github.com/Crizstian/cinema-microservice> |
| 50 | vertx-blueprint-microservice | <https://github.com/sczyh30/vertx-blueprint-microservice> |
| 51 | Microservice kafka | <https://github.com/ewolff/microservice-kafka> |
| 52 |  | <https://github.com/bigcompany/hook.io> |
| 53 | Debugger for microservices | <https://github.com/solo-io/squash> |
| 54 |  | <https://github.com/networknt/microservices-framework-benchmark> |
| 55 |  | <https://github.com/kbastani/event-stream-processing-microservices> |
| 56 |  | <https://github.com/pywren/pywren> |
| 57 | Microservice dashboard | <https://github.com/ordina-jworks/microservices-dashboard> |
| 58 |  | <https://github.com/lipp/login-with> |
| 59 | Customers stores | <https://github.com/spring-cloud-samples/customers-stores> |
| 60 | Cinema written using Flask | <https://github.com/umermansoor/microservices> |
| 61 | Airships manufacturer example | <https://github.com/nameko/nameko-examples> |
| 62 | SpaCy | <https://github.com/explosion/spacy-services> |
| 63 | Microservices Dockerfiles | <https://github.com/izuolan/dockerfiles> |
| 64 | Todo | <https://github.com/h4xr/todo> |
| 65 | ICp Banking Microservices | <https://github.com/IBM/ICp-banking-microservices> |
| 66 | Microservice for rendering PDF/PNG/JPEG from HTML with Electron | <https://github.com/msokk/electron-render-service> |
| 67 | Stockquote | <https://github.com/IBMStockTrader/stock-quote> |
| 68 | Email service | <https://github.com/clevertech/email-service> |
| 69 | Store and display sales receipts | <https://github.com/pagarme/tldr> |
| 70 | Timestamp | <https://github.com/Rafase282/Timestamp-API> |
| 71 | Spreadsheet csv conversion | <https://github.com/SheetJS/sheetaki> |
| 72 | Pitstop – Garage Management system | <https://github.com/EdwinVW/pitstop> |
| 73 | Shopping Cart | <https://github.com/CiscoCloud/shipped-demo-web> |
| 74 | IBM Watson Assistant journey | <https://github.com/IBM/conversation-with-linuxone-using-watson-microservices> |
| 75 | eSchool microservice | <https://github.com/OpenCodeFoundation/eSchool> |
| 76 | Report microservice | <https://github.com/k8guard/k8guard-report> |
| 77 | Real Time streaming PCF microservice | <https://github.com/kgshukla/Realtime-Streaming-PCF-Microservices-Docker> |
| 78 | Homepage for all the microservices | <https://github.com/TangentMicroServices/Dashboard-WebClient> |
| 79 | School service microservices | <https://github.com/vzhuleho/microservices-school> |
| 80 | Stripe charge | <https://github.com/ColeTownsend/micro-stripe-charge-example> |
| 81 | Auditing server as a service | <https://github.com/audit4j/audit4j-microservice> |
| 82 | PDF generator | <https://github.com/ministryofjustice/pdf-generator> |
| 83 | Proxy | <https://github.com/Clamenu/eddie-proxy> |
| 84 | Microservices orchestration | <https://github.com/nittapp/main> |
| 85 | Microservices agency | <https://github.com/TheMicroservicesAgency/msa-website> |
| 86 | Simple stateless microservice | <https://github.com/SVM143/Microservices> |
| 87 | File conversion | <https://github.com/sgbj/versed> |
| 88 | Microservices infrastructures | <https://github.com/josedab/spring-cloud-examples> |
| 89 | VM creation | <https://github.com/ewolff/microservices-vm> |
| 90 | Scientific visualization as a microservice | <https://github.com/seelabutk/tapestry> |
| 91 | Tom microservice example | <https://github.com/Kikobeats/tom-example> |
| 92 | The user microservice | <https://github.com/microservices-demo/user> |
| 93 | eCommerce microservice | <https://github.com/digota/digota> |
| 94 | Microservice toolkit for NY times | <https://github.com/nytimes/gizmo> |
| 95 | Greeter microservice | <https://github.com/micro/examples/tree/master/greeter> |
| 96 | Booking example | <https://github.com/micro/examples/tree/master/booking> |
| 97 | Secure | <https://github.com/micro/examples/tree/master/secure> |
| 98 | Metadata | <https://github.com/micro/examples/tree/master/metadata> |
| 99 | Language sentiment | <https://github.com/cdipaolo/sentiment-server> |
| 100 | MariaDB for microservices | <https://github.com/bstaijen/mariadb-for-microservices> |