A Systematic Study of Micro Service Architecture Evolution and their Deployment Patterns

Chaitanya K. Rudrabhatla
Executive Director- Solutions Architect
Media and Entertainment domain
Los Angeles

ABSTRACT
With the advent of Local Area Networks (LAN), the client server architecture gained traction. The ever-growing need for the distributed systems have paved the path for client server architecture to transform in to Service Oriented Architecture (SOA). Due to the reusable and loosely coupled nature of the services, SOA became a successful representation of client server architecture. However, over time SOA fell short of expectations, as it was fully reliant on monolithic system design. Achieving horizontal scalability, faster response times, high availability, infrastructure agility, service and resource isolation was a challenge in SOA frameworks. Micro service architecture (MSA) soon came to the rescue. It offered various solutions to overcome most of the shortfalls of the traditional monolithic SOA architecture. But at the same time, MSA comes with its own set of challenges due to the complex distributed design. Among various design complexities involved in MSA, creating, managing and deploying microservices in a clustered production grade environment is a major challenge. A micro service can be deployed to run on a virtual machine (VM) or on a container which itself runs on a VM. The VM can be in the data center or in the public cloud. The containers can be self-managed or orchestrated. The orchestration can be done by the cloud provider or a third-party software. This research paper illustrates (1) the journey of architectural design patterns from SOA to MSA, by citing the related work and the reasons for evolution. (2) various deployment models available for MSA (3) comparison of the deployment models and a quantitative analysis of the use cases.

General Terms
Micro service architecture, Micro service deployment patterns, SOA, containerization, containers, orchestration.

Keywords
Service Oriented Architecture (SOA), Microservice architecture (MSA), containers, orchestration, deployment patterns, Micro service cloud deployment patterns.

1. INTRODUCTION
Web application architecture has come a long way from the initial days of client server model to the fully distributed container based micro service architecture. The ever-growing demand to build the distributed systems which are light weight, reusable, reliable and highly available has been the major reason for the advent of various architectures and design patterns, which paved the path to the current day distributed micro service architecture [1]. A decade back in time, client-server model was one of the most popular designs for building the distributed systems [2][3]. Client server model is a network computing architecture where a powerful central server hosts, manages and delivers the resources or services needed by the clients. The clients are usually lesser powerful computers which connect to the central server using remote procedure calls (RPC) over the internet. This remote method invocation has given rise to a n-tier or multi-tier architecture in java-based applications. However, it was soon realized that the client server model is not easily scalable [4]. It was expensive and time taking to scale the central server. Added to this, the other drawback was the inability to integrate autonomous services. These shortfalls have given rise to a more decoupled Service Oriented Architecture (SOA). SOA was successful for some time. But due to its monolith design it couldn't sustain for long either (discussed in Section 3). This led to the evolution of micro service architecture(MSA), which is an architectural design pattern that structures an application as a collection of loosely coupled services, which implement business capabilities. The MSA is designed on the same principles of SOA. Only difference is that, in micro service architecture, the services are broken down into granular light weight and standalone deployable units. The wide spread usage and acceptance of MSA has popularized the container technology in parallel. This container technology has been long existent in Unix and Linux world. With the onset of newer changes which contributed to the ease of usage of containers and the rapid growth of cloud platforms, has greatly contributed to the steep rise of containers in the world of web development. MSA has been able to solve many problems involved in SOA based architecture like achieving the horizontal scalability, high availability, modularity and infrastructure agility. But at the same time, it introduced its own set of challenges due to the complex distributed nature of its design. We observed that a micro service can be deployed in a variety of ways. It can be made to run on (a) single virtual machine or (b) a cluster of VMs or (c) on a container which runs on any kind of VM running in the local data center like a Hypervisor based VM or a vSphere VM, or (d) on a container deployed on a computing machine running on the cloud like AWS EC2 or Azure VM to name a few, or (e) as a serverless deployment on the cloud like AWS Lambda or Azure functions or GCP functions. It can be clearly seen that there are multiple ways to deploy and run the micro services. Even within the techniques mentioned above, there are a multitude of ways to maintain the containers. They can be orchestrated using the native services provided by the cloud platforms like ECS by AWS or ACS by Azure, or by using the third-party orchestration tools that have been introduced by various big players in the industry like kubernetes by google, Mesos by Apache, to name a few. All the deployment techniques mentioned above have their own set of advantages and disadvantages depending on the use case which is being implemented. Lot of researchers have suggested many ways and pointers to pick up the right deployment pattern suitable for the use case. In this

research paper, a quantitative analysis is performed by deploying a custom project developed in java spring boot technology and recommendations are provided for choosing the suitable deployment model.
The paper is organized as follows. In section 2, the challenges involved in SOA based architecture which led to the evolution of micro service architecture are explained by referring to the related work. In section 3, the challenges involved in the designing the deployment framework for MSA is explained. In section 4, we explain how the deployment patterns are analyzed using the custom research project. section 5, provides the conclusion to the paper.

2. TRANSITION FROM SOA TO MSA

2.1 Background on SOA and related work

Over the past decade or so, Service Oriented Architecture(SOA) became one of most successful implementations of client server model. Before SOA, all the applications were monolith in nature. With the monolith design, as the size of the applications grew, it became extremely difficult to scale the application and at the same time the productivity of the developers declined due to the confounding nature of the design. The code reusability was another major challenge with the monolith architecture. SOA attempted to solve these challenges associated with the monolith applications [5]. It allows the services to be loosely coupled and reusable. Multiple end user systems can make use of common services once they are developed. SOA is designed based on the principles of (a) service abstraction – where end user applications are agnostic of service implementation. Services act as black boxes. (b) service autonomy – Services are designed and run independently and control the functionality they encapsulate. (c) Service discovery - All the services are supplemented with additional metadata making them discoverable. (4) Service reusability - Functionality is logically divided in to services such that the code can be reused and extended. Fig 1. Given below shows the basic SOA architecture. As shown in the diagram, multiple end user client applications can trigger calls to the services. These calls are handled by Enterprise Service Bus (ESB). As name suggests it integrates various application services together over a Bus-like infrastructure. It includes a service registry which keeps track of widely located services. When the client makes a call, ESB translates it to the suitable message type understood by the contacted service. SOA at high-level acts as a wrapper for loosely coupled web services where services share a common standard for interaction [6].

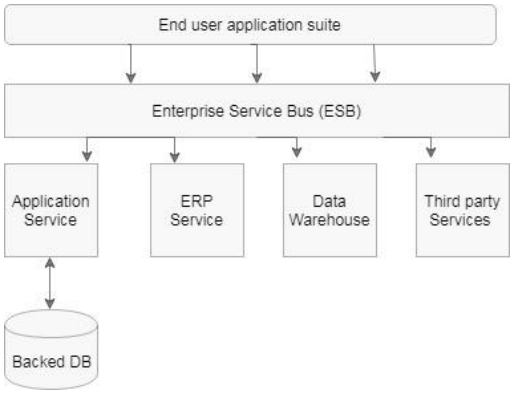


Fig 1: SOA based design with ESB columns

This architecture provides a model for various modules and organizations to reuse their services with various customers and clients. Researchers in [6] and [9] evaluated the standards and distributed capabilities of SOA based architecture. They investigated the various layers involved in SOA protocol stacks and concluded the benefits of its usage in huge enterprises.

2.2 Drawbacks of SOA

Though SOA was a great improvement from the monolith design, it could not keep up the pace with the ever-growing business demands. Though the services were delineated in SOA, they need to be deployed as a single unit in the form fat application services. This model couldn't cope up with the needs to develop scalable solutions with lesser resources. Though it was argued by some that SOA based services can be scaled by deploying multiple copies of the same application service, it becomes very hard and daunting task for developers to maintain large code bases for the single application service and patch it regularly with minor business enhancements. Also, deploying multiple copies of large monolith is not a real scalable design as the resources are not allocated as per the individual service needs [7]. It is broadly assigned at the application level and may lead to underutilization and wastage of assigned memory. On top of it, redeploying the large application for every minor enhancement is quite challenging for the developers. Also, SOA performs the service routing, orchestration and business validations at a single central hub called ESB, which becomes a cumbersome layer as services grow. To handle these draw backs Micro Service Architecture (MSA) came in to light.

2.3 Evolution of MSA

The drawbacks mentioned above with SOA architecture have prompted the researches to come up with a newer architecture which fulfills those short comings. On top of it, the ever-growing business needs and the constant demand to push numerous enhancements to the production systems multiple times a day, created the need for a lighter, highly scalable and easily deployable architecture. Due to these needs, MSA came into existence. MSA is a design pattern in which an application is broken down in to a set of smaller, light weight and independent services. MSA relies mostly on SOA principles but it is fine-tuned with following major differences:
(a) As discussed in section 2.1 SOA relies heavily on an intelligent and heavy central layer called ESB. Whereas MSA aims to identify the services based on dumb endpoints. The intelligence is all embedded within the services rather than a heavy central layer. (b) The heavy monolith-based services are broken down into several smaller and independent services. This makes the services totally delineated. (c) Each service can be built in a different programming language and platform than the others, based on the business needs. [8] (d) Each service can be independently developed and deployed without impacting other services.

2.4 MSA benefits - Related work

As discussed above, MSA is designed on the principles which are different than the multitiered monolith frameworks. MSA comprises of multiple light weight services which are logically grouped based on the functionality of the business domain. Microservices are lightweight, flexible and highly scalable. Since MSA evolved from SOA, many technologies related to routing, service discovery, circuit breakers, security, monitoring, configuration management and load balancing are developed which when combined, overcome the shortfalls of

SOA. Fig 2. shows all the layers involved in the typical Micro service architecture.

Most of the researchers have analyzed the benefits of MSA over SOA and submitted the related work. Based on the earlier research, the major advantages of MSA can be broadly classified into the following: (a) Reusability of the code – Since it is a loosely coupled architecture, the services once written can be reused across multiple applications [10] (b)loosely coupled design – Each service is independent of the other. Services are totally delineated including the databases. [11] [12] (c) Horizontal scalability – Scalability has been one of the major advantages and primary reasons for the evolution of MSA. Lot of research has been done to describe the benefits of scalability in MSA. [13] [14] (d) resilience [14] (e) cost benefits. [15]

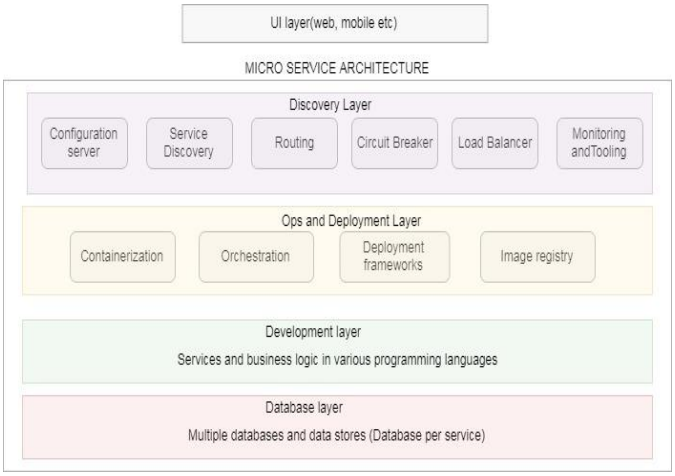


Fig 2: Layers in Micro Service Architecture

3. MSA DEPLOYMENT TECHNIQUES AND CHALLENGES

As briefly discussed in Section 1, micro services increase efficiency and are advantageous in many ways. However, MSA comes with its own set of challenges. Out of many design challenges which are present in MSA design, the deployment architecture is one of the most common ones faced by the architects. There are a multiple of deployment patterns available for MSA which can soon become overwhelming, when trying to pick the right one. The design considerations which lead to deployment challenges can be broadly classified as follows – (a) Choice of the right infrastructure platform. (b) Logical grouping the micro services. (c) Containerization of services and their orchestration. Let us discuss the design considerations in little detail.

3.1 MSA deployment design considerations

3.1.1 Choice of the right infrastructure platform – related work

Micro services can either be deployed on virtual machines running in the data center or in the cloud infrastructure. Lot of researchers have already worked to identify the bottle necks and implications in the infrastructure design involved in the local data centers [16] [17]. MSA was designed to achieve horizontal scalability and quality of service (QoS). It is difficult to achieve in the data-center based infrastructure. Whereas the cloud platforms have evolved to a great scale and provide numerous services to achieve the scalable designs. But with the multitude of public cloud platform options available, for example: AWS, Azure, GCP, IBM Bluemix to name a few, and the plethora of infrastructure services they offer, it becomes a daunting task to pick up the right choice and design an optimal deployment architecture for MSA [18].

3.1.2 Logical grouping the micro services - related work

As per the principles of MSA design, every service needs to be totally independent of the other. Even the databases cannot be shared as per the principles of database per service pattern [11] [19]. Depending on the application design there may be a need to group two or more containers running the micro services on the virtual machine to achieve efficiency in the remote procedure calls (RPC). For example, a caching service may be logically grouped with the database service on the same pod in the Kubernetes cluster using Docker containers for better performance. This might be argued as an anti-pattern. But while doing a green field application design, it might be needed for higher efficiency. However, this logical grouping adds to the deployment complexity [20].

3.1.2.1 Containerization and orchestration of services

Though the concept of containers was present in Unix from a long time, they gained real significance only after the wide spread usage of cloud platforms. It has been proved to be highly advantageous to run the micro services on containers rather than virtual machines directly. Fig 3 shows the basic layout of Hypervisor based VM vs container. As shown in the diagram, virtual machines(VM) run the heavy guest operating systems with their own set of heavy binaries and libraries managed by Hypervisor which runs on top of the physical server with a Host OS. This heaviness increases the boot up time for VMs. Whereas a container runs with a virtual OS as opposed to VMs which makes it very quick and light weight. It is evident from the diagram, that the containers are light and can be quickly be created, destroyed or restarted. The quick creation and destruction aspect helps in achieving the horizontal scalability at a lightening pace which is not possible in VM based data centers. And the quick restart feature helps the businesses to roll out new features with a minimal or no downtime. Though it is greatly beneficial to run a micro-service based application on a multitude of containers in a clustered environment for the reasons mentioned above, we observed that it comes with its own set of challenges- few related to micro-services, like the port allocations, service discovery, routing, distribution of services and few others which are related to containers like health monitoring and container management. Many container orchestration tools have been introduced by various big players in the industry like kubernetes by google, ECS by AWS, Mesos by Apache to name a few. Introducing containers and orchestration tools adds up to the complexity of the design.

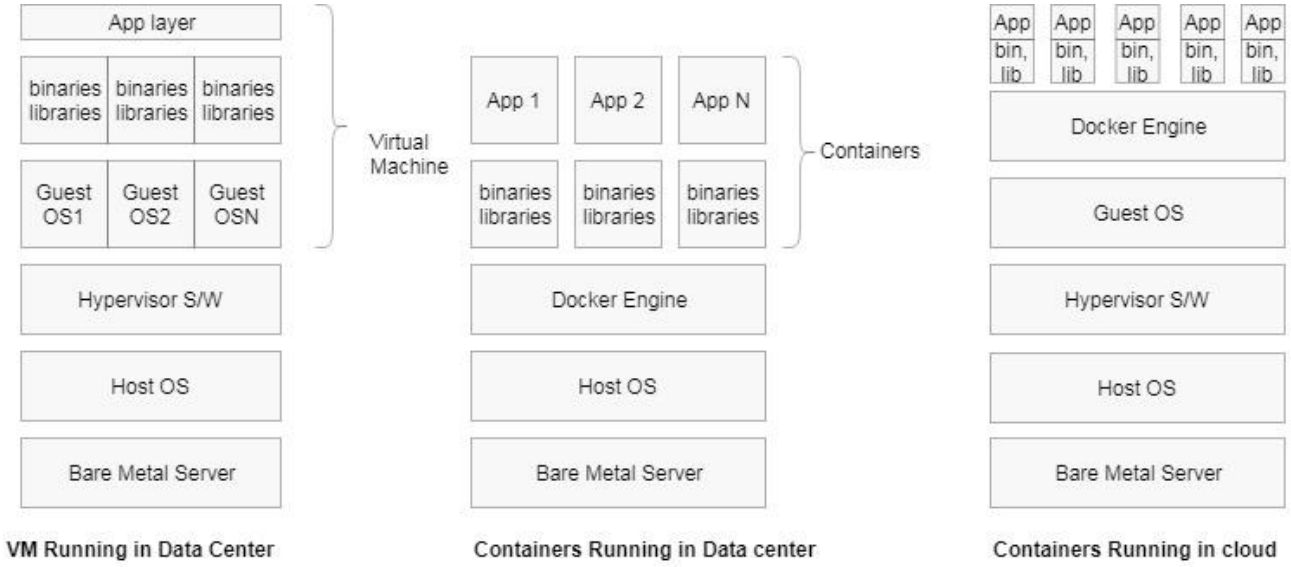


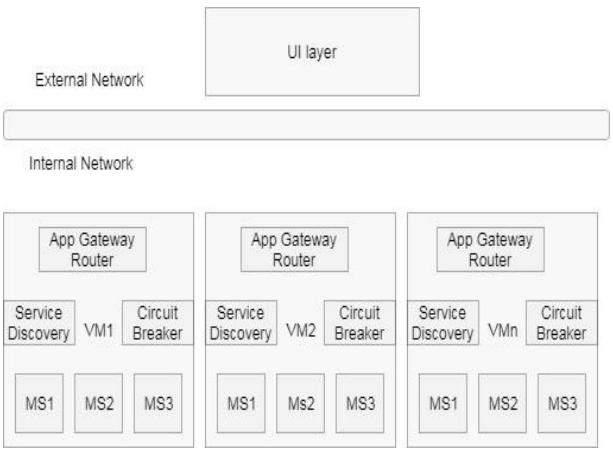
Fig 3: Virtual Machine VS Containers Layout

4. CUSTOM RESEARCH PROJECT TO EVALUATE DEPLOYEMENT PATTERNS

To determine which deployment technique is more suitable under which scenario, we have implemented a research project and simulated various circumstances. We have implemented java based micro services in spring boot technology. We deployed 3 micro services MS1, MS2 and MS3 in various deployment patterns mentioned below and summarized the comparisons in the next section –

4.1 On-Premises Data Center deployment

As a part of this, 3 micro services MS1, MS2 and MS3 are deployed by running the docker images on the virtual machines VM1, VM2 and VM3 running on Hypervisor software as shown in Fig 4. A router component is used for routing the requests to the appropriate service which is discovered using a service discovery layer. A circuit breaker was used to handle failures.



4.1.1 Observations

We simulated load tests by firing the HTTP get requests using an open source tool called JMeter. It has been observed that the advantage with the above deployment pattern is that it is simple enough and easily understandable. The complexity and number of components involved in the design are limited. However, it is observed that there are following disadvantages with this model – (a) Maintainability: Maintainability is difficult as it is custom built. (b) Auto-Scaling: Auto scaling the application is not possible. (c) High Availability and Load Balancing: Load balancing of the services and adding fault tolerance needs extra components and custom configurations which are difficult to manage. (d) Upgrades and Rollback Application roll backs and upgrades to newer versions is tedious and requires a downtime. (e) Service discovery: It is complex and needs to be custom built. (f) Health checks: Need to add custom components and adds to the complexity.

4.2 Cloud based deployment and orchestration

Deployed 3 micro services MS1, MS2 and MS3 in AWS cloud using the Elastic container service (ECS) cluster. A group of EC2-T3.Mediums are created which are managed by ECS. An Auto Scaling Group (ASG) is created for the EC2 instances with min=2 and max=4 configuration. An Elastic Load Balancer (ELB) is used for routing the requests to the appropriate micro service. This is achieved by configuring the listeners and Target groups on the ELB. The design is implemented as shown in Fig 5. Appropriate IAM roles needed for the EC2 instances to be operated by ECS and at the same time the IAM roles needed for ECS to communicate with ELB are chosen. Linux based AMIs needed to run the spring boot based micro services and docker containers is used. Then the docker images containing the spring boot based micro services are deployed and run on the ECS cluster

as Tasks. A Task is a unit of work which defines how the container should be co-located in EC2 instances.

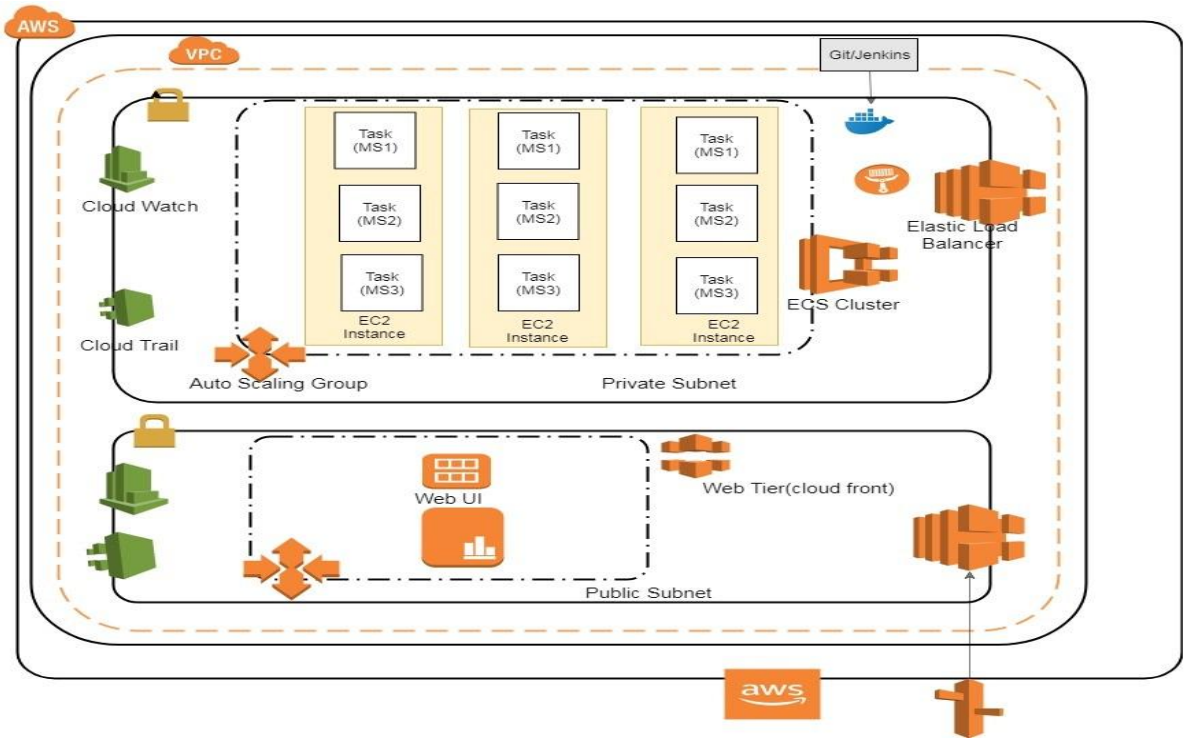


Fig 5: MSA on AWS cloud using ECS

4.2.1 Observations

We observed that deploying the micro service containers in AWS cloud is highly advantageous compared to running them in the local data centers. Here are a few advantages with this model – (a) Maintainability: It is not difficult as in the data center world. Cloud providers like AWS provide a rich set of tools and technologies to maintain the deployed services (b) Auto-Scaling: Auto scaling groups can be configured to scale up or scale down the ECS service based on CPU or memory usages. (c) High Availability and Load Balancing: ELB can provide out of the box load balancing solution. Services are run as ECS Tasks on the EC2 instances. ELB can load balance and distribute the requests to healthy containers. (d) Upgrades and Rollback: Application roll backs and upgrades to newer versions can be achieved with minimal or no downtime. There are parameters like minimum and maximum healthy percent which can be adjusted to achieve rolling deployments by spinning up a batch of parallel containers.(e) Service Discovery: Route53 and ELB can provide out of the box service discovery by routing the requests using listeners and target groups, to appropriate services.(f) Health checks: AWS comes with Cloud watch service which can monitor the health of ECS cluster. (g) Multi cloud support: It is a vendor lock if we chose this option for orchestration. We cannot have one set of services running on AWS and other on Azure with this kind of orchestration technique. (h) External storage: Restricted to EBS volumes in AWS.

4.3 Cloud based deployment with Kubernetes orchestration

Kubernetes is an open-source system for managing and automating the deployment, scaling and management of containerized applications. Kubernetes, with help of Pods, takes the software encapsulation provided by Docker further. A Pod is a collection of one or more Docker containers with single interface features such as providing networking and file system at the Pod level rather than at the container level. We deployed the spring boot based Micro services MS1, MS2 and MS3 on the Kubernetes cluster running on EC2 instances in AWS cloud using Kubeops as shown in Fig 6. Kubernetes cluster contains a master node and worker nodes. Master node places container workloads in the user prods running in the worker nodes or on itself. Master node runs an (a) API server which acts as management layer which facilitates communication with the cluster and perform tasks, such as servicing API requests and scheduling containers and a (b) controller manager which maintains the state of cluster and auto scales the workloads. A kubelet receives the pod information from the API server and updates the nodes accordingly. Services are the endpoints exposed externally using the Kubernetes DNS server which connects pods using the label selectors.

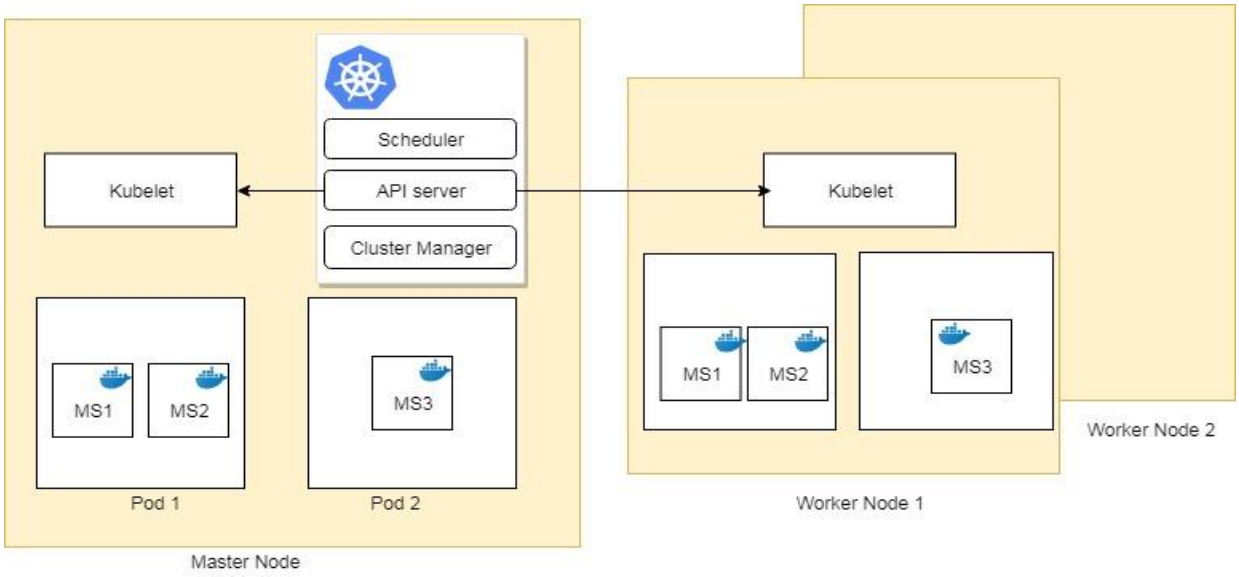


Fig 6: MSA on Kubernetes cluster

4.3.1 Observations

Here are the observations which we made by running the micro services on the Kubernetes cluster – (a) Maintainability: The deployment of the applications in to Pods is specified declaratively in the form of YAML. Deployment, scaling, co-location and administration is more convenient with YAML. (b) Auto-Scaling: It can easily be configured using the resource metrics or by adjusting the 'number of pods' parameter which can specified declaratively in deployments. c) High Availability and Load Balancing: Cluster of master and worker nodes can be load balanced for requests from Kubectl and clients. Also, pods are exposed through a service which can be used as a load balancer with in the cluster. d) Upgrades and Rollback: Application roll backs and upgrades to newer versions can be achieved with minimal or no downtime. The deployment supports both rolling update and recreate strategies. (e) Service Discovery: Each Kubelet provides environment variables to access the host and port or a DNS server as an add on. DNS server creates a bunch of DNS records for each kubernetes service. With DNS enabled, the pods can use the service names that automatically resolve. (f) Health checks: Kubernetes supports 2 kinds of health checks to find liveliness and readiness of the services to check the responsiveness and preparedness of the application. (g) Multi cloud support: Kubernetes can be used on premises, or any public clod or a combination of public clouds. (h) External storage: Using this orchestration model provides wide variety of storage options including on-premises SAN or other volume options provided in public cloud platforms.

5. CONCLUSIONS AND FUTURE WORK

From the above observations, it can be concluded that picking up the right architecture for the deployment of micro services needs too many factors to be considered. Deploying the micro services as docker containers without any orchestration tool in the data center world may be simple to start with and it can soon become very challenging to maintain if the application scalability needs grow. Cloud services like AWS ECS provide a lot of out of the box features needed for load balancing, service discovery, auto scaling, health monitoring etc. which are very difficult to custom build in the data center deployment model. However, using such native cloud orchestration services leads to a vendor lock-in. Also, the limitations like restrictions on storage options in the cloud provider may not be cost effective. On the other hand, going with open source orchestration frameworks like Kubernetes, gives the similar benefits as AWS ECS. On top of it, it has other advantages like the flexibility to pick up various storage options, and a large community support. It is also beneficial if the organization wants to use a multi-cloud deployment model to support sensitive workload management or to avoid vendor lock-ins. However, picking the open source tools like Kubernetes for container orchestration leads to steep learning curve and increased complexity, as everything is a do-it-yourself model.

Further work needs to be done to study the deployment models in other cloud providers like GCP, Azure etc. and serverless deployment models like AWS lambda, Azure and Google functions to name a few. And thus, compare the pros and cons in each approach to come up with standards which

would help organizations to pick up right deployment model for MSA.

6. REFERENCES

[1] B. A. Akinnuwesi, F.-M. E. Uzoka, and A. O. Osamiluyi, "Neuro-fuzzy expert system for evaluating the perfonnance of distributed software system architecture," Expert Systems with Applications, vol.40, no.9, pp. 33 13-3327, 2013.

[2] W. A. De Vries and R. A. Fleck, "Client/server infrastructure: a case study in planning and conversion," Industrial Management & Data Systems, vol. 97, no, 6, pp, 222-232, 1997.

[3] Salah, Tasneem & Zemerly, Jamal & Yeob Yeun, Chan & Al-Qutayri, Mahmoud & Al-Hammadi, Yousof. (2016). The evolution of distributed systems towards microservices architecture. 318-325. 10.1109/ICITST.2016.7856721.

[4] M. Van Der Vlugt and S. Sambasivam, "Redesign of stand- alone applications into thin-client/server architecture," Informing Science : International Journal of an Emerging Transdiscipline, vol. 2, pp. 723-742, 2005.

[5] L. Ismail, D. Hagimont, and J. Mossi'ere, "Evalnation of the mobile agents technology: Comparison with the client/server paradigm," Information Science and Technology (1ST),vol. 19, 2000.

[6] B. Li, "Research and application of soa standards in the integration on web services," in 2010 Second International Workshop on Education Technology and Computer Science (ETCS) , vol. 2 . IEEE, 2010, pp. 492-495.

[7] D. Namiot and M. Sneps-Sneppe, " On micro-services architecture," International Journal of Open Information Technologies, vol. 2, no. 9, 2014.

[8] S. Newman, Building Microservices. " O'Reilly Media, Inc.", 2015.

[9] P. Offermann, M. Hoffmann, and U. Bub, "Benefits of SOA: Evaluation of an implemented scenario against alternative architectures," in 2009 13th Enterprise Distributed Object Computing Conference Workshops. IEEE, 2009, pp. 352-359.

[10] M. Rahman and J. Gao, "A reusable automated acceptance testing architecture for microservices in behavior-driven development," in Service Oriented System Engineering (SOSE), 2015 IEEE Symposium on. IEEE, 2015, pp. 321-325.

[11] K. Rudrabhatla, Chaitanya. (2018). Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture. International Journal of Advanced Computer Science and Applications. 9. 10.14569/IJACSA.2018.090804.

[12] M. Vianden, H. Lichter, and A. Steffens, "Experience on a microservice based reference architecture for measurement systems," in 2014 21st Asia-Pacific Software Engineering Conference, vol. 1. IEEE, 20 14, pp. 183-190.

[13] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," in 2 0 1 6 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2016, pp. 202-211.

[14] Wan, Xili & Guan, Xinjie & Wang, Tianjing & Bai, Guangwei. (2018). Application deployment using Microservice and Docker containers: Framework and optimization. Journal of Network and Computer Applications. 119. 10.1016/j.jnca.2018.07.003.

[15] A. Levcovitz, R. Terra, and M.T.Valente, "Towards a technique for extracting Microservices from monolithic enterprise systems," arXiv preprint arXiv: 1605.03175, 2016.

[16] Gan, Yu & Delimitrou, Christina. (2018). The Architectural Implications of Cloud Microservices. IEEE Computer Architecture Letters. PP. 1-1. 10.1109/LCA.2018.2839189.

[17] Delimitrou, Christina & Kozyrakis, Christos. (2013). Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS. 41. 77-88. 10.1145/2490301.2451125.

[18] Visti, Hannu & Kiss, Tamás & Terstyanszky, Gabor & Gesmier, Gregoire & Winter, Stephen. (2016). MiCADO – Towards a microservice-based cloud application-level dynamic orchestrator. 0.7287/PEERJ.PREPRINTS.2536.

[19] Messina, Antonio & Rizzo, Riccardo & Storniolo, Pietro & Tripiciano, Mario & Urso, Alfonso. (2016). The Database-is-the-Service Pattern for Microservice Architectures. 9832. 223-233. 10.1007/978-3-319-43949-5_18.