Abstract—Microservice Application Containers are often identified in one breath with docker container technology. Docker is an open source container platform and it helps to distribute source code more rapidly, investigate earlier deploys and set up faster. Microservices consisting of multiple small, independent modules. Each module (known as a service container) is modular, reusable, and self-contained and interconnects with other service container using language-agnostic protocols often use hypertext transfer protocol (HTTP) based and representational state transfer (REST) styled protocols. In this paper, we propose an effective technique with docker container service for microservice applications.

Keywords—Container, Cloud Applications, Docker, Microservices, Orchestration Platform

I. INTRODUCTION

Microservice containers are small, independent services that work together. Microservices are used by companies like Netflix, Amazon [1][2]. This kind of design patterns is used to build big, complex and scalable applications composed of small and independent modules interact with each other using language-agnostic application programming interfaces. Microservice methodologies for modern apps and docker container-based operating system virtualization involvement a recovery in the cloud computing. Mainly the docker container-based virtualization approaches are often stated to be a high-performance marginal to hypervisors [3]. The Docker [4] is such a container service solution, and it is based on operating system virtualization using Linux container services. Recent
presentation studies show only little performance impacts to processing, network, memory, I/O [5]. That is why Docker Container Service states publicly itself a “lightweight virtualization platform” providing a constant runtime, container image format, and build system for Linux container independent service deployable to any cloud Infrastructure as a Service (IaaS) environment. These docker container services run in an isolated way on top of the operating system’s kernel.

This study investigated on modern apps for evolving next-generation cloud application with Docker Container Services. Distributed cloud-based modern apps for typical complexity often use hypertext transfer protocol (HTTP) based and representation state transfer (REST) styled protocols to enable horizontally scalable service container designs [6]. There exist many open source applications providing such a microservice approach on top of IaaS provider specific infrastructures using this methodology (e.g. Kubernetes, CoreOS and etc..). These approaches are intended to be deployable to public or private IaaS infrastructures [7].

The technological principle of Docker isolates applications by packing them with their dependencies, libraries, and configuration files into one consistent image-based environment like an atomic unit, called containers according to [8]. Those container services can be seamlessly moved between hosts or within cloud providers and we combined different approaches to proving effectiveness with equipped.

Section II presents related work on state-of-the-art container methodologies for Modern Apps with Docker Container Service. Section III clarifies the proposed technique for modern apps with Container Service. The benchmark tooling [10] and the performance data collected is provided online. Research and performance study are discussed in Section IV. Derived design recommendations for Rapid distribution, fast provisioning, and plain monitoring are presented in concluding Section V.

II. RELATED WORK

A lot of researchers have been proposed for modern apps with the cloud platform. A brief review of some of the recent researches is presented here about container service. Nane Kratzke[11] have stated about Microservice usage, Containers and their Underestimated Impact on Network Speed. Chen Yang has [12] proposed the Checkpoint and Restoration of Micro-service in Docker Container Service for high availability[13].

III. PROPOSED TECHNIQUE FOR MODERN APPS

Microservice architecture usually requires enthusiastic teams delivering specific business potential. Service container (See Fig. 1) should be designed as an independent product delivering a business capability that is well documented with the container, easy to use and accountable for one and only one business capability; this is known as the single accountability principle (SAP). A moral rule of thumb is that the microservice containers should be as small as possible but as big as necessary to represent the domain concept or business capability it represents.

A. Designs for Orchestration Layers

Considering the growing number of software device types and the fact that more company’s business strategies are dependent on providing value to customers on several devices and it is no longer optimal to have granular resource-based application programming interfaces (APIs) that closely represent the data model.

B. Embrace DevOps Practices

While microservices architecture [9] is a cleaner methodology, it is also a complicated one with more moving parts with heterogeneous languages and frameworks with service protocols. Organizations have to shift to the DevOps practice model, which identifies that software developers are in the best place to operationalize (write code, build, distribute) their own container service.

C. Docker

Docker is an open-source platform for coding, building, and running applications independently.

1) Docker images

A Docker image is a read-only template of Docker containers. For example, the Docker image could be full of an Ubuntu operating system with Apache Server and your web application can be installed within docker image. Docker Images are used to create multiple containers along with specific services.

2) Docker containers

A Docker container holds all that is required for an application to run the service container. Each service container is created from the Docker image. Docker containers will be started with container service, stopped with service container, and can be deleted.

D. Feature extraction

1) Availability

The golden rule for accessibility says container expect failures and design accordingly so that the container service system will be available for 99.999%. It means the Docker container service system can go down only for 5.5 minutes for an entire year. The cloud cluster model is used to maintain high availability of container service, where it endorse having the set of services run in the Active-Active model or Active-Standby model.

15592
2) Scalability
Microservices must be scalable both horizontally and vertically with service container. Being container with horizontal scalable, we can have various instances of the microservice to increase the performance of the container system.

3) Usability
When considering the usability of microservice, the design looks at hiding the application inner design, tech, architecture and other difficulties to the end user or another system. The APIs should be considered in a normalized way so that it is easy to achieve the necessary services with the minimal number of API calls.

4) Flexibility
Flexibility measures the adaptability to change. Each independent service is owned by different teams and established in agile methodology, modification will happen faster than any other systems. The microservices may not inter-operate if they don’t settle in or accommodate the change in othersystems.

IV. RESULTS AND DISCUSSION

This section describes the experimental results of our proposed modern apps segmentation technique using Docker images. Our proposed approach is implemented in Docker (version 1.17). Here, we have tested our proposed microservice segmentation technique using docker images taken from the publicly available sources. The performance of the proposed technique is compared with the monolithic applications and the obtained results for the microservice are evaluated through evaluation metrics namely, availability and accuracy.

A. Experimental results

The Docker experiments for modern apps(microservice) were conducted on the single host with Ubuntu 16.04 system with Intel Sandy Bridge CPU (E5-2620) clocked at 2GHz and 8GB DRAM. I have created the docker container for the fleet system, dashboard, and mongo using Table I. (See Fig. 2).

<table>
<thead>
<tr>
<th>CONTAINER_ID</th>
<th>DACE</th>
<th>PORTS</th>
<th>NAMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1e69fe2e615a</td>
<td>modernapp/fleetsystem</td>
<td>0.0.0.0:8080-&gt;8080/tcp</td>
<td>fleetsystem</td>
</tr>
<tr>
<td>2069c2e18f0b</td>
<td>modernapp/databse</td>
<td>0.0.0.0:4200-&gt;4200/tcp</td>
<td>dashboard</td>
</tr>
<tr>
<td>061f4a21203f</td>
<td>mongo</td>
<td>0.0.0.0.127.0.0.170.0.17.0.17/tcp</td>
<td>mongod</td>
</tr>
</tbody>
</table>

TABLE I COMMAND FOR DOCKER IMAGES

<table>
<thead>
<tr>
<th>Steps</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build container image</td>
<td>docker build-t modernapp/fleetsystem</td>
</tr>
<tr>
<td>Run container</td>
<td>docker run -itd p 8080:8080 modernapp/fleetsystem</td>
</tr>
<tr>
<td>Stop container</td>
<td>docker stop &lt;container-id&gt;</td>
</tr>
<tr>
<td>Remove container</td>
<td>docker rm &lt;container-id&gt;</td>
</tr>
<tr>
<td>Remove Container image</td>
<td>docker rmi &lt;image-id&gt;</td>
</tr>
</tbody>
</table>

1) Using Docker Compose to organize the running containers

Organizing a group of containers is the purpose of Docker Compose. We state the set of Docker containers in a YAML configuration file and it succeeds the runtime configuration of the containers.

[Docker-compose.yml]

    fleetsystem:
        build: fleetsystem
        ports:
            - "8080"
        links:
            - mongod
    dashboard:
        build: dashboard
        ports:
            - "4200"
        links:
            - mongod
    mongod:
        image: mongo

2) Scaling containers and load balancing

If you run “docker-compose scale [compose docker container name]=3,” then it will create three instances of your container — running “docker-compose scale fleetsystem=3”
then "docker-compose ps" and our fleet system container now
has three instances!

Let’s add a load balancer to balance the multiple fleet
system containers we now have. We’ll just add that to our
[docker-compose.yml] file

```yaml
... ha_fleetsystem
  image: tutum/haproxy
  links:
    - fleetsystem
  ports:
    - "8080:80"
```

B. Performance evaluation of the proposed technique

The performance of our proposed technique is evaluated by
means of the existing monolithic applications and the
proposed microservice applications in terms of the evaluation
metrics value. The obtained values of the evaluation metrics of
the microservice applications in cloud platform are compared
against the standard applications.

So, appropriate technology and design decisions must be
taken, to avoid re-work in the later stage. By analyzing the
results, our proposed microservice application performs better
than the existing one. The outcomes of the experimentation
proved with 99% of accuracy in microservice cloud
applications with docker images.

C. Comparative analysis

The monolithic app is very challenging to scale and is
unreliable. As a result of the legacy application, agile
development and deliveries are impossible on time. Also,
monolithic applications make it extremely difficult to adopt
new frameworks and languages.

All modern day digital architectures are supposed to be
service enabled through SOAP/HTTP web services, RESTful
APIs, JMS queues, etc. A microservice Architecture is a very
specific variation of a service-enabled architecture which is
more focused on agility. It can provide significant value in a
number of areas such as:

- For a rapidly growing business where time-to-market
  is key
- Test the market situations where functionality needs
to be rolled out in a low-cost manner focused on
validating the demand
- For improving the functionality of OOB(out-of-the
  box) application or Software as a Service (SaaS)
service to minimize customization and thus simplify
future application upgrades

V. CONCLUSIONS

Modern applications need a new groundwork, built on
scalable, resilient and adaptive services. Our Modern
Apps(Microservices) offer an approach for delivering highly-
scalable, cross-platform services that can be developed,
deployed and scaled independently of one another to provide
the modern application product team with the greatest amount
of flexibility and control.

To successfully put into business operation for microservice
architecture, organizations have to meet basic requirements
that require an organizational shift, such as close collaboration
between developers and operations. Areas to focus on include:

1) Rapid Distribution
   With many service containers to manage with
   organizations need to be able to rapidly set up them for
   both to test and production environments.
2) Fast provisioning
   Along with continuous delivery, organizations must
   have automated mechanisms for machine provisioning,
   as well as for deployment, testing, etc.
3) Basic monitoring
   With more moving parts that must be coordinated in
   production, basic monitoring must be in place to detect
   problems quickly. This monitoring is able to include
detecting technical issues such as service availability
and business issues such as a rapid drop in transaction
volume.

By moving to a microservices-based approach, organizations
can address the business needs for operational flexibility,
functional simplicity and continuous change that define
today’s digital economy.

VI. REFERENCES

    [Retrieved: September 2016]
    2015.
    Peterson, “Container-based operating system virtualization: A
    scalable, high-performance alternative to hypervisors,” SIGOPS
    https://docker.com
    performance comparison of virtual machines and Linux
    containers,” IBM Research Division, Austin Research Laboratory,
    based software architectures,” Ph.D dissertation, University of


[13] Gatete Marcel, 2 Dr.N. Vetrivelan QOS-AWARE TRANSMISSION FOR MULTIMEDIA APPLICATIONS IN MANET USING ACO WITH FUZZY LOGIC International Journal of Innovations in Scientific and Engineering Research (IJISER)