Journal of Mines, Metals and Fuels, 71(4): 545-551; 2023. DOI: 10.18311/jmmf/2023/33936

Print ISSN : 0022-2755

Journal of Mines, Metals and Fuels

Contents available at: www.informaticsjournals.com/index.php/jmmf

Improving Business Deliveries for Micro-services-based Systems using CI/CD and Jenkins

Vinay Singh[1], Amarjeet Singh[2], Alok Aggarwal[3*], Shalini Aggarwal[4] and Himanshu Chaudhary[5]

[1,2,3]School of Computer Science, University of Petroleum & Energy Studies, Dehradun, India.
[4]Graphic Era Hill University, Dehradun, India.
[5]Manipal University Jaipur, Jaipur India

Abstract

Micro-services architecture has changed the paradigm of software designing and software development behaviour as they are lightweight, easy adoptable, faster to be built and deploy at business servers. The nature of today's market changes so abruptly that the software industry faces problems or impediments to hit the market demand due to complex, large, monolith clumsy applications which are complex to be broken easily and eventually fails to hit time-to-market approach. In this work, a novel migration approach has been proposed to improve the business deliveries and cover time-to-market approach by using advanced Continuous Integration and Continuous Delivery process and sophisticated Devops and cloud tools like Jenkins, Git and Amazon cloud which has micro-services deployed in the containerization form. Finally, a model and a robust system has been investigated and proposed in terms of decomposition, testing, security, performance, inter service communication, persistence, transaction management aspects. The whole analysis revolved around to propose a novel approach to how to reduce the build and deployment time and make the end business product available to end customer in a faster and rapid way.

Keywords: *Version Control System; Git, Subversion; micro-service; Kubernetes; container; Jenkins; Spring Boot; Monolithic; CI/CD*

1.0 Introduction

Micro-services architecture has changed the paradigm of software designing and software development behaviour as they are lightweight, easily adaptable, and faster to be built and deployed at business servers. The volume and architecture of enterprise applications are very large and complex so they always intend and expected to be built easily, easily scalable, fault-tolerant and work independently. Several organizations strive to achieve better, faster and customer-oriented business applications by developing the fault-tolerant system. However, several industries are facing the problem of meeting the demand of end customers within the time-frame or as per market demand for it. The market is changing and it is volatile in nature. The nature of today's market changes so abruptly that software industry faces problems or impediments to hit the market demand due to complex, large, monolith clumsy applications which are complex to be broken easily and eventually fail to hit the time-to-market approach. It has been observed that there are very few works available on how to improve business deliveries by adopting Micro-services and implementing a better Continuous Integration (CI) and Continuous Delivery (CD) pipelines which makes the end product available to end customer on time. CI emphasizes and supports that the software industry must find out the production errors and

[*]Author for correspondence

software defects at early stage rather than in production environment. In this work, focus has been on how to implement a robust, steady and automated CI/CD pipeline which builds the micro-services faster, error-free, and deploys it to business servers before the market need the product feature. Several inputs have been taken and fed them to measure whether really micro-services and CI/CD are capable enough to meet the business demands before time or not. A comparison between micro-services and monolith applications have been analyzed in terms of build and deploy time, scalability, fault tolerance and time-to-market.

A typical process to improve the business readiness and final end product in the form of micro-services involves the following steps:

- Identify business functional components
- Prioritized and defector components
- Designing the micro-services using Spring Boot framework and architecture
- Setting up continuous integration and continuous deployment pipelines using Jenkins and PCF
- Build the micro-services using Git, Jenkins, Python and Angular
- Deploy the micro-services to Containerized environment
- Measure the different parameters like build and deployment time
- Total time to scale-up and scale-down in peak traffic flow
- Total latency difference between monolith and Micro-services
- Systematic scalability difference between monolith vs Micro-services
- Total time spent in infrastructure readiness for business production servers (time-to-market)

Due to its inherited advantages and benefits, for the past few years the software industry has strived to migrate from monolithic legacy systems to a micro-service architecture. A monolithic application architecture support only a single unified dependable unit while a micro-services architecture entails to smaller, independently deployable services, which are lighter in weight and easily deployable. They consume less CPU resources or memory and are faster in response. micro-services, as opposed to monolithic apps, are compact, autonomous entities that focus on certain functions and collaborate with others to support an application's operation. The benefit of a micro-service architecture is that developers can deploy features that prevent cascading failures[1-5]. Monolithic application is designed in such a way that using a monolithic architecture structure, if production fails in one subcomponent of the architecture, it will ruin and breaks all architectural components altogether. On the other hand the beauty of micro-services is that each subsystem is independently written and with a micro-services architecture, if one service fails, it is much less likely that other parts of the application will fail because each micro-service runs independently. Various results have been achieved, which clearly indicate that micro-services are faster, easier, and error free to end real customer in real-time business environments as they are easy to scale up, faster to build, have low latency, fault-tolerant and hits time-to-market factor[6-12].

Continuous Integration and Continuous Delivery helps the software world to find the bugs in the code issues and fix it faster, which entails to faster application delivery times and decreased time to market factor. Modern applications make use of Dev-Ops practices like CI/CD to reduce several duplicate tasks which can increase development time and consume unnecessary time to deploy to Business servers[13-15]. Figure 1 shows the Dev-Ops eco-system for micro-services. In CI/CD process, CI which means continuous integration merges incremental code changes to the main version control repository on a regular basis as the developer commit the change in code. This code merge or code check indirectly call version control webhooks to trigger an automated build process that runs unit and integration tests to cover the software testing in an automated fashion. CI process ensures that bugs and integration issues are detected early in the development stages and aren't propagated through to production.

Dev-Ops is traditionally based on Agile methodology[16-17]. Each software development environment advocates the AGILE culture where open dialogue between the software developers and the product owners, software testers, architects and finally release managers are involved with deployment, testing (the operations team) and maintenance of the end product which business people use for their own use cases, hence the name Dev-Ops. Dev-Ops approach adds the feature of maintainability to the development lifecycle[18-26]. Continuous integration is another feature that makes Dev-Ops faster and more efficient than agile. Few researchers strive to improve the delivery of several Micro-services, others focus on containerized culture like PCF,

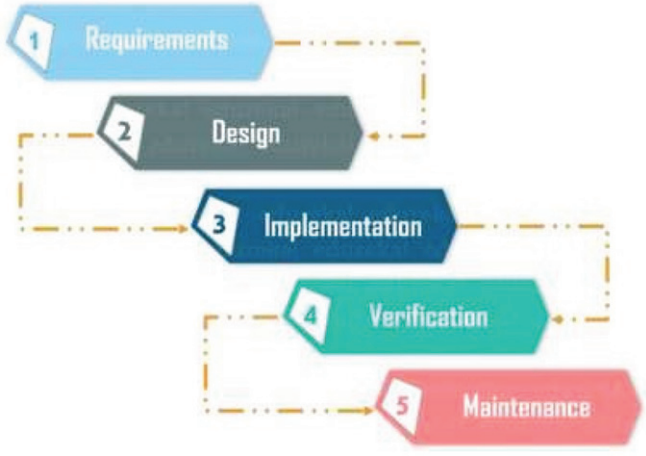

Figure 1: DevOps eco system for micro-services

OpenShift or Kubernetes problems and their real-time challenges, but in these efforts, we never find the relationship between the two different eco-systems.

2.0 Methodology and Experimental Description

2.1 Installing Jenkins and Setting up Continuous Integration and Continuous Delivery Pipeline

Various steps used for installing Jenkins and setting up Continuous Integration and Continuous Delivery pipeline are given below.

Step 1:

Setup Jenkins

Setup docker

Execute these system commands:

systemctl start jenkins

systemctl enable jenkins

systemctl start docker

Step 2:

Open Jenkins console

Click New Item > create your first job

Step 3:

Choose freestyle project with name and save

Step 4:

Go to configuration section > SCM

Link Repository

Step 5:

Select Build option

Place shell script and Execute shell

The architectural flow of build and deploy of micro-services from development to business production servers is shown in Figure 2. Figure 3 shows how to set up Jenkins and CI/CD pipelines to build micro-services. Bash Script written for constructing CI/CD Pipeline for micro-services is shown in Figure 4.

Constructing a rapidly deployable micro-services architecture can bring better adoptability, maintainability and agility, but it entails on the approach of how developers and architects share code and data between micro-services and prepare for Inter-Pods communication. Micro-services must be constructed using design patterns and spring boot

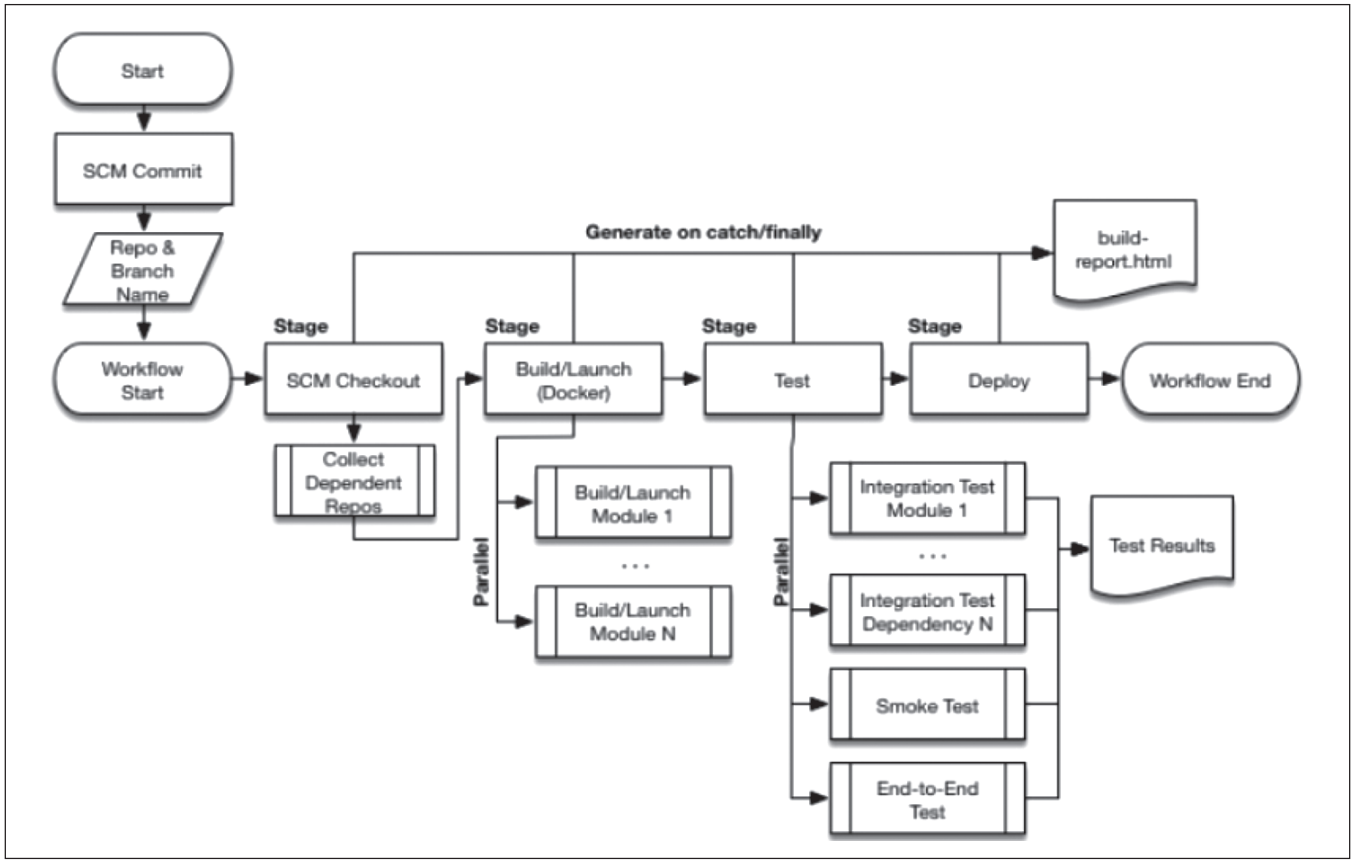

Figure 2: Architectural flow of build and deploy of micro-services from development to business production servers

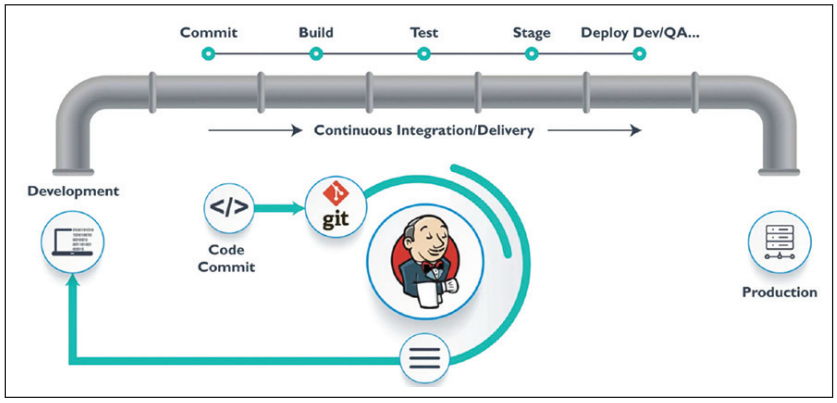

Figure 3: Setting up Jenkins and CI/CD pipelines to build microservices

```
#!/bin/bash
echo "********-Starting CI CD Pipeline Tasks-********"
#-BUILD
echo ""
echo "..... Build Phase Started :: Compiling Source Code :: ...."
cd java_web_code
mvn install

#-BUILD (TEST)
echo ""
echo "..... Test Phase Started :: Testing via Automated Scripts
cd ../integration-testing/
mvn clean verify -P integration-test

RUNNING=$(sudo docker inspect --format="{{ .State.Running }}" $CONTAINER 2> /dev/null)

if [ $? -eq 1 ]; then
  echo "'$CONTAINER' does not exist."
else
  sudo docker rm -f $CONTAINER
fi

    # run your container
    echo ""
    echo "..... Deployment Phase Started :: Building Docker Container :: ....."
    sudo docker run -d -p 8180:8080 --name devops_pipeline_demo devops_pipeline_demo


#-Completion
echo "------------------------------------------------------"
echo "View App deployed here: http://server-ip:8180/sample.txt"
echo "------------------------------------------------------"
```

Figure 4: Bash Script to construct CI/CD Pipeline for Microservices

framework, but no-code platforms can call a web service to request data and return it much more quickly. Micro-services are usually adopted and constructed as backend services where it provides end API to call a service or group of services. Micro-services are designed in such a way that every scalable architecture makes it easy to design and deploy so quickly and consume API backend services. Ultimately, it hits time-to-market and provides business values to the end customers. Web and mobile front-ends are user interfaces that leverage functionality provided by back-end services.

2.2 Software Code Build and Deployment Time

The two most important factors which play major role in the software industry are speed and agility where software code is written and checked into distributed version control system like Git. Obviously, the way developers write and do programming on their own systems is often faster than deploying sharing resources on a web server. Monolith system are complex in nature and due to high dependencies, the build and deploy mechanism in monoliths is more complex and time-consuming and never hits the time-to-market factor. Modules in micro-services are isolated from each other, making it easier to build and deploy. We built several micro-services and found that the build and deployment does not significantly increase if we increase the huge number of Micro-services at once but in the case of monolithic applications, we see a significant amount of time to build it once. This helps us to deploy the application in real-time environments like business production servers are faster and smoother and ultimately hits time-to-market. Figure 5 shows a comparison on build and deployment time between micro-services and monolith.

2.3. Latency in Monolith and Micro-services

The digital transformation or modernization of legacy applications from monolith architecture to micro-service architecture is getting increasingly popular in recent times. Figure 6 shows how the latency increased drastically in the monolith system for a range of 500 to 2500 HTTP calls per second which are very small and incapable in the contemporary business world where an average of trillion transactions are served every day.

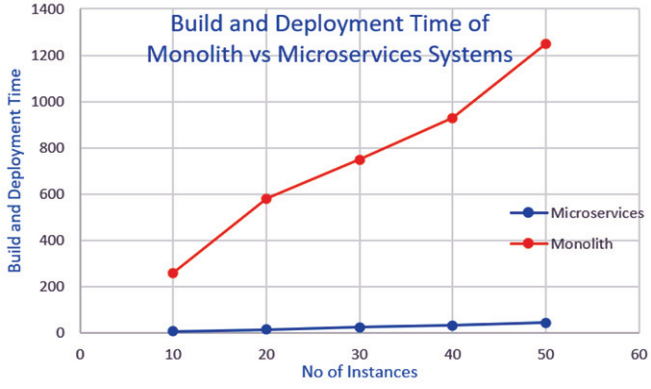

Figure 5: A comparison on Build and Deployment time between microservices and monolith

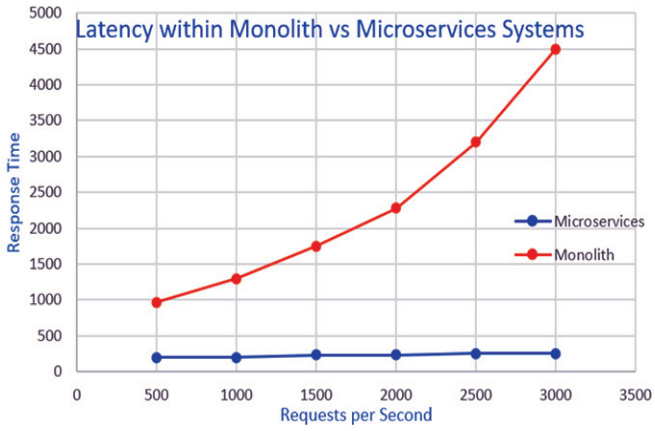

Fig. 6. Latency statistics between Monolith vs Micro-services based systems

2.4 Time to Market: Monolith vs Micro-services

Reducing time-to-market is the first and foremost priority for each organization and business. Reducing the time it takes to get a product to market gives a competitive advantage that allows staying ahead of contemporary competitors, respond faster to market changes, and increase the business with the latest modifications and better customer experience. Monolith systems are typically harder to deploy because of their large size and complex dependencies. It is experimentally proven that micro-services are faster to be deployed and ready for time-to-market, but monolith systems are complex and they are not flexible enough if any change in the system is required and require an entire change in the system hence cannot be easily built and deployed. Figure 7 shows the total time spent in infrastructure readiness for business production servers (time-to-market).

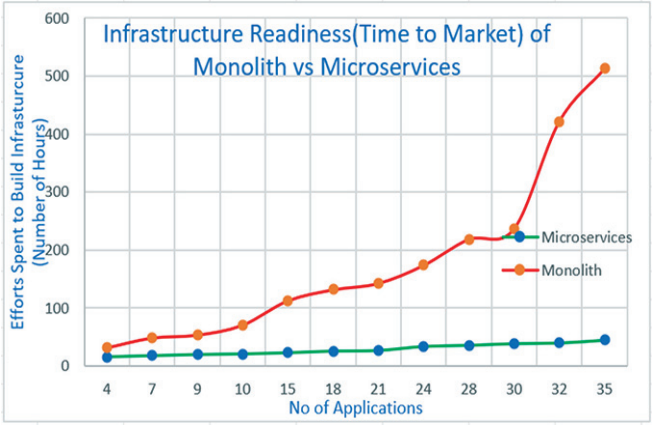

Fig. 7. Total time spent in infrastructure readiness for business production servers (time-to-market)

3.0 Results and Discussion

Micro-service engineering trends tend to result in sub-optimal applications of this architecture fashion, driving an environment in which groups begin to think that micro-services are yet another embraced fad. Dev-Ops technology also has significant business value. The approach increases the quality of enterprise software applications while also reducing delivery times. Dev-Ops technology also improves business team productivity and growth. The Dev-Ops trend has prompted many businesses to automate their software deployment and release processes. Dev-Ops can support the company by enhancing communication and teamwork within the workplace. Frequent communication boosts efficiency, and the more employees connect when working together, the more they can maintain high-quality service. In that situation, it is important to understand why Dev-Ops is a tool for increasing employee productivity and camaraderie. These are some of the reasons why Dev-Ops is one of the most effective tools for team management. Dev-Ops puts together two critical aspects of the company processes, as the name implies. The strategy incorporates operations and creation under one roof. One umbrella could be a single team with shared objectives. The business team takes care of every stage of the Dev-Ops application lifecycle, including growth, testing, and operations, thanks to the convergence of business developments and business goals. There is an option of testing the Dev-Ops approach to its limits. As a result of the business automation enabled by Dev-Ops technology, the business team will collaborate to introduce new practices. Automation is one of the main advantages of Dev-Ops because it allows for accelerated business development by eliminating sluggish and manual processes. Dev-Ops helps companies to grow at a fast and consistent pace. Dev-Ops, on the other hand, can be extremely difficult to implement if done incorrectly.

In a typical scenario, several micro-services are deployed in real-time environments in several pods. Hence we must have a proper design and a well-defined architectural flow of these micro-services so that we can easily and correctly communicate with each other and their backend API services with clean interfaces for services and a convenient database for each service. Finally, the software industries nowadays put several efforts to overcome the problem of monolith applications by fully migrating to micro-services.

Figure 8 shows the Python Script to build and deploy for micro-services in CI/CD pipeline. Results show that micro-services systems take very less time to be built and they are considerably fast in response so eventually they take significantly less amount of time to be built, deployed, and considerably faster in response as compared to monolith systems. Monolith systems are very expensive and hard to scale-up and scale down based on business needs whereas

micro-services-based systems are scalable and highly independent in nature fault-tolerant and continuously deployable.

```
deploy:
    needs: [test]
    runs-on: ubuntu-latest

    steps:
    - name: Checkout source code
      uses: actions/checkout@v2

    - name: Generate deployment package
      run: zip -r deploy.zip . -x '*.git*'

    - name: Deploy to EB
      uses: einaregilsson/beanstalk-deploy@v20
      with:

        // Remember the secrets we embedded? this is how we access them
        aws_access_key: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws_secret_key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

        // Replace the values here with your names you submitted in one of
        // The previous sections
        application_name: django-github-actions-aws
        environment_name: django-github-actions-aws

        // The version number could be anything. You can find a dynamic way
        // Of doing this.
        version_label: 12348
        region: "us-east-2"
```

Fig. 8. Python Script to Build and Deploy for Microservices in CI/CD pipeline

4.0 Conclusion

With some reports published by Gartner, it has been evidenced that today, almost 71% of software companies, many of which are among the major players in the IT market, have already adopted the design pattern of micro-services from their development, SIT, UAT and production business servers. Whereas 6% of large companies, 50% of medium companies, and 24% of small companies are using micro-services in production and development. The Gartner statistics show us that the adoption of micro-services with the containerized approach is generally applied in business-demanding environments where applications are critical and require processing the data in real-time systems at peak hours in the real-time system and within high real-time transactions and even at massive scales a traffic flow allows jobs to be conducted independently. In traditional monolithic application development, subcomponents of monolith applications are so tightly bound together that they become so clumsy to redesign and scale up. Conversely, micro-services are written in a well-designed Spring boot framework that uses a modular design structure that enables software designers and architects to code, test, and debug each small independent component of the application without having a dependency on the entire system.

In this work, a novel migration approach has been proposed to improve the business deliveries and cover the time-to-market approach by using advanced Continuous Integration and Continuous Delivery process and sophisticated Dev-Ops and cloud tools like Jenkins, Git and Amazon cloud which has micro-services deployed in the containerization form. Finally, a model and a robust system has been investigated and proposed in terms of decomposition, testing, security, performance, inter-service communication, persistence, and transaction management aspects. The whole analysis revolved around proposing a novel approach to how to reduce the build and deployment time and make the end business product available to end customers in a faster and more rapid way.

4.0 References

1. Goyal M.K. et al. (2012): QoS based trust management model for Cloud IaaS. In: 2nd Inter. Conf. on PDGC, pp. 843-847.
2. Kumar A., Krishan G. et al. (2016): Design and Analysis of Lightweight Trust Mechanism for Secret Data using Lightweight Cryptographic Primitives in MANETs. *Inter. Jour. of N/w Security;* 18 (1): 1-18.
3. V. Singh, A. Singh, A. Aggarwal and S. Aggarwal (2021): Dev-Ops based migration aspects from Legacy Version Control System to Advanced Distributed VCS for deploying Micro-services. In: 2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), pp.1-5.
4. Kumar A., Krishan G. et al. (2015): A Novel Trusted Hierarchy Construction for RFID-Sensor Based Secure Mobile Ad Hoc NETworks (MANETs) using Error Correcting Codes (ECCs). *Electronics Tele. Research Instt. Journal* 2015, 37 (1):186-196.
5. Kumar Adarsh et al. (2013): Outlier Detection and Treatment for Lightweight Mobile Ad Hoc Networks. Lect. Notes of the Instt.for Comp. Sc., *Social Info.& Tele. Engg,* 115, pp 750-763.
6. Kumar A, Aggarwal A, Charu. (2012): Efficient Hierarchical Threshold Symmetric Group Key Management Protocol for Mobile Ad Hoc Networks. In: Parashar et al. (eds) Contem. Computing, *Commu. in Computer and Infor. Sc*., 306, pp. 335-346.
7. Kumar A, Krishan G, and Aggarwal A. (2017): A Novel Lightweight Key Management Scheme for RFID-Sensor integrated Hierarchical MANET based on Internet of Things. *Inter. Jour. of Adv. Intel. Paradigms*; 9 (2-3): 220-245.
8. Aggarwal Alok, Singh Vinay and Kumar Narendra. (2022): A Rapid Transition from Subversion to Git: Time,

Space, Branching, Merging, Offline Commits & Offline builds and Repository Aspects. *Recent Advances in Computer Science and Communications*, 15(5).

9. Vinay Singh, Alok Aggarwal, Narendra Kumar, A. K. Saini. (2021): A Novel Approach for Pre-Validation, Auto Resiliency & Alert Notification for SVN To Git Migration Using Iot Devices. *PalArch's Journal of Arch. of Egypt/Egyptology*, 17 (9): 7131 - 7145.
10. Kumar A., Krishan G., et al. (2014): Design and Analysis of Lightweight Trust Mechanism for Accessing Data in MANETs. *KSII Trans. on Internet and Infor. Systems*; 8 (3): 1119-1143.
11. V. Singh, A. Singh, A. Aggarwal and S. Aggarwal. (2021): A digital Transformation Approach for Event Driven Micro-services Architecture residing within Advanced vcs. In: 2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON), pp. 100-105.
12. Kumar A, Krishan G, et al. (2014): Simulation and Analysis of Authentication Protocols for Mobile Internet of Things (MIoT). In: Proc. 3rd Inter. Conf. on PDGC, pp. 423-428.
13. Gupta P et al. Trust and reliability based scheduling algorithm for cloud IaaS. In: Lect. *Notes in Elec. Engg.* 2013, 150, pp.603-607.
14. Kumar A., Aggarwal A. (2012): Lightweight Cryptographic Primitives for Mobile Ad Hoc Networks. Recent Trends in Comp. Net. & Distributed Systems Security Comm. in *Computer and Infor. Science*; 335: 240-251.
15. Vinay Singh, Alok Aggarwal, Adarsh Kumar, and Shailendra Sanwal. (2019): The Transition from Centralized (Subversion) VCS to Decentralized (Git) VCS: A Holistic Approach. *Journal of Electrical and Electronics Engineering*, 12 (1):7-15.
16. Kumar A, Aggarwal A, and Charu. (2012): Performance analysis of MANET using elliptic curve cryptosystem. In: 14th Inter. Conf. on Adv. Comm. Tech. (ICACT), pp.201-206.
17. Goyal MK, Aggarwal A. (2012): Composing Signatures for Misuse Intrusion Detection System Using Genetic Algorithm in an Offline Environment. In: N. Meghanathan et al. (eds) Adv. in Compu. and Infor. Tech., *Adv. in Intelligent Systems and Comp*., 176, pp 151–157.
18. Adarsh Kumar et al. (2012): A complete, efficient and lightweight cryptography solution for resource constraints Mobile Ad-Hoc Networks. In; Proc. PDGC, pp. 854-860.
19. Sangeeta Mittal et al. (2012): Application of Bayesian Belief Networks for context extraction from wireless sensors data. In: 14th Inter. Conf. on Advanced Comm. Tech. (ICACT), pp. 410-415.
20. N. Chugh et al. (2016): Security aspects of a RFID-sensor integrated low-powered devices for Internet-of-Things. In: 4th Inter. Conf. on PDGC, pp. 759-763.
21. Singh T. et al. (2017): A novel approach for CPU utilization on a multicore paradigm using parallel quicksort. In: 3rd Inter. Conf. on Compu. Intelligence & Comm. Tech. (CICT), pp. 1-6.
22. Kumar A., Gopal K., and Aggarwal A. (2016): Simulation and Cost Analysis of Group Authentication Protocols. In: Inter. Conf. on Contemporary Computing, pp.1-7.
23. Chakradar M et al. (2021): A Non-invasive Approach to Identify Insulin Resistance with Triglycerides and HDL-c Ratio using Machine learning. *Neural Processing Letters*, 52 (3).
24. Mittal S. et al. (2012): Situation recognition in sensor based environments using concept lattices. In: Proc. Inter. Infor. Tech. Conf., pp.579-584.
25. Kumar A. and Aggarwal A. (2019): An Efficient Simulated Annealing based Constrained Optimization Approach for Outlier Detection Mechanism in RFID-Sensor Integrated MANET. *Inter. Jour. of Computer Infor. Systems and Indus.* Mgt. App; 11: 55-64.
26. V. Singh and A. Aggarwal. (2014): Performance analysis of middleware distributed and clustered systems (PAMS) concept in mobile communication devices using Android operating system. In: 2014 International Conference on Parallel, Distributed and Grid Computing, pp. 345-349.