Deployment of Multi Tier Application using EKS

Prof.Shikha Dwivedi
Department of Computer Engineering
JSPM's JSCOE

Pranit Pramod Kolamkar
Department of Computer Engineering
JSPM's JSCOE

Deepam Himachal Patle
Department of Computer Engineering
JSPM's JSCOE

Shreyash Shivanand Nakhate
Department of Computer Engineering
JSPM's JSCOE

Abstract - The project report titled "3-Tier Application Deployment on AWS using Amazon EKS" focuses on the design, development, and deployment of a highly scalable and secure 3-tier web application architecture using Amazon Elastic Kubernetes Service (EKS). The 3-tier architecture consists of a presentation layer (frontend), an application layer (backend), and a database layer, each deployed in isolated environments for enhanced security and performance. The project demonstrates the use of Kubernetes for container orchestration, enabling efficient management of microservices and load balancing. Additionally, it leverages AWS services like Elastic Load Balancer (ELB), Amazon RDS, and CloudWatch[2] for monitoring and scaling. This deployment model ensures high availability, fault tolerance, and streamlined application management, providing a robust solution for modern cloud- native applications. In this deployment model, the frontend consists of a user interface, typically running as a web application. The backend processes the business logic and APIs, while the database layer stores and manages application data. Each tier is containerized using Docker and deployed on Kubernetes clusters within Amazon EKS[1], which automates scaling, load balancing, and monitoring of containers.

Keywords – 3-tier architecture, AWS (Amazon Web Services), Amazon EKS (Elastic Kubernetes Service), Frontend, Backend, Database, Elastic Load Balancing (ELB), Amazon RDS (Relational Database Service), AWS Cloud Formation, VPC (Virtual Private Cloud), Role-Based Access Control (RBAC).

I. INTRODUCTION

This report outlines the deployment of a 3-tier application architecture on Amazon Web Services (AWS) using Amazon Elastic Kubernetes Service (EKS)[1]. A 3-tier application architecture traditionally consists of three layers: the presentation tier (UI), the application tier (business logic), and the database tier (data storage). Each of these tiers can be independently managed, scaled, and optimized, which provides a modular approach for developing, deploying, and maintaining modern applications[12].

Kubernetes automates the deployment, scaling, and management of containerized applications, providing a strong foundation for the continuous integration and continuous delivery(CI/CD)pipeline. This report details the steps and methodologies used in deploying the application components across the three tiers on AWS, leveraging various services such as Elastic Load Balancing,[15] Amazon RDS, and AWS Identity and Access Management (IAM) for security and scalability.Additionally, we discuss the benefits of this architecture, such as high availability, fault tolerance, and cost efficiency, along with potential challenges encountered during the deployment process. In this project, the deployment was carried out using Amazon EKS, a managed Kubernetes service that automates the provisioning, scaling, and management of containerized applications in AWS[2]. Kubernetes, being the industry- standard orchestration platform, allows us to efficiently manage containers and provides features such as load balancing, scaling, self-healing, and automated rollouts.

II. LITERATURE SURVEY

Cloud-native architectures[19], driven by Kubernetes, have transformed the deployment and management of multi-tier applications. Amazon Elastic Kubernetes Service (EKS) simplifies Kubernetes cluster management while offering scalability and integration with AWS services. Research emphasizes how multi-tier applications (comprising frontend, backend, and database tiers) benefit from Kubernetes' container orchestration, including fault tolerance, auto-scaling, and CI/CD pipelines. Several studies highlight the reduction in deployment complexity and improved system reliability achieved using Kubernetes-based workflows.

EKS builds upon Kubernetes' capabilities, leveraging AWS's managed infrastructure[13] to enhance performance. Studies show that EKS significantly reduces operational overhead for managing Kubernetes clusters by automating node provisioning, load balancing, and monitoring[17]. Research also illustrates EKS's seamless integration with AWS services like RDS, S3[5], and IAM, which supports robust backend and database functionalities. These integrations simplify secure application development and deployment for multi-tier applications.

While Kubernetes and EKS offer numerous advantages, deploying multi-tier applications comes with challenges, as detailed in existing research. Common issues include network complexity, inter-tier communication latency[13], and the need for advanced monitoring tools. Studies focus on using tools like AWS CloudWatch, Prometheus[18], and Grafana to ensure observability and maintain application performance. Further, authors have discussed strategies to manage service discovery and traffic routing effectively, including the use of AWS Application Load Balancers (ALBs) and Ingress controllers[14].

The deployment of multi-tier applications using Amazon Elastic Kubernetes Service (EKS) has been extensively explored in contemporary research due to its ability to provide scalable, secure, and highly available solutions for cloud-native applications[20]. Studies emphasize leveraging Kubernetes' orchestration capabilities to manage the lifecycle of multi-tier architectures, including the frontend, backend, and database layers, while integrating[2] AWS-native services like Elastic Load Balancing, IAM for security, and CloudWatch for monitoring. Research highlights the benefits of deploying applications in EKS clusters, such as containerized scalability, fault tolerance, and streamlined CI/CD workflows using tools like Jenkins or GitOps practices[21].

Security remains a critical focus in multi-tier application deployment. Research emphasizes securing application workloads through Kubernetes network policies, AWS security groups, and role-based access control (RBAC)[15]. Studies also highlight implementing secure data storage with encrypted AWS services (e.g., EBS and S3) and ensuring identity management using AWS IAM and Secrets Manager. Findings suggest that these practices enhance the integrity and confidentiality of multi-tier systems in production[13].

Literature on resource optimization for EKS discusses autoscaling and workload optimization techniques, such as using Horizontal Pod Autoscalers (HPA)[20] and Spot Instances. Researchers have analyzed how EKS supports cost-effective scaling for varying workloads in multi-tier applications. Studies on serverless databases and container optimization reveal how combining Kubernetes with AWS Fargate[14] can minimize idle resource costs.

Best practices highlighted in research include adopting Infrastructure as Code (IaC)[13] with tools like Terraform or AWS CloudFormation to manage cluster configurations. There is also significant emphasis on CI/CD pipelines using AWS CodePipeline and GitOps frameworks for continuous deployment. Research underscores leveraging Helm charts[7] for application configuration and reducing downtime through blue/green deployments or canary releases.

The deployment of multi-tier applications using EKS addresses scalability, security, and operational efficiency. Research indicates that adopting cloud-native principles and leveraging AWS ecosystem tools can streamline development workflows and improve overall application resilience. Future studies could focus on evolving EKS features[15], such as AI- driven workload optimization and hybrid cloud scenarios, to further enhance the deployment of multi-tier applications.

The deployment of multi-tier applications using Amazon Elastic Kubernetes Service (EKS) has become a critical area of research and practice in cloud computing. This survey provides a comprehensive overview of existing research and practical insights into deploying multi-tier applications using Amazon EKS. It highlights the integration of Kubernetes' orchestration capabilities with AWS-managed services to create scalable, secure, and maintainable cloud-native applications.

III METHODOLOGY

Problem Definition: Deploying scalable and secure 3 tier application remains challenging in dynamic environments. This project aims to streamline this process by utilizing AWS and Amazon EKS[1] to enhance scalability, security, and efficiency in application deployment.

Existing System Architecture : In a typical three-tier application architecture deployed on AWS using Amazon Elastic Kubernetes Service (EKS), the system is divided into three main layers: the presentation layer, the application (or business logic) layer, and the data layer. Each tier serves a distinct purpose and is hosted on separate, dedicated infrastructure, ensuring modularity[21], scalability, and security.

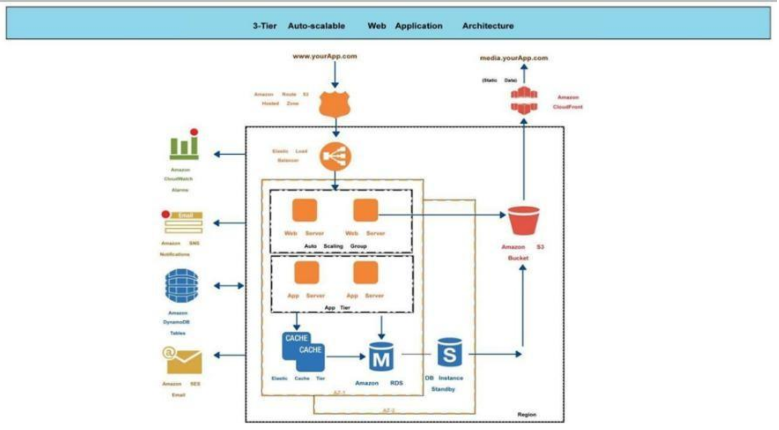


Fig.01 Existing System Architecture

1. Presentation Layer: This tier, responsible for user interaction, is often hosted as a set of containerized services on EKS[1]. It handles requests from end users and typically includes web servers, such as NGINX or Apache, that serve front-end components of the application. In this layer, users' requests are routed through an Application Load Balancer (ALB)[20], which distributes incoming traffic to the containers in the EKS cluster. The ALB provides HTTP/HTTPS access and can perform SSL termination, protecting traffic to and from the presentation layer[5].

2. Application Layer: The business logic of the application resides in this layer, also deployed on EKS as a set of microservices or containerized applications. Kubernetes manages these services for high availability and auto-scaling, distributing them across multiple worker nodes within the EKS cluster[22]. This layer communicates with both the presentation and data layers, processing user requests, executing business logic, and retrieving or storing data as needed. Managed by Kubernetes, this layer benefits from advanced container orchestration features like rolling updates, load balancing, and self-healing, which help ensure continuous availability and resilience[10].

3. Data Layer: The data tier holds the application's persistent data, which is typically stored in a managed database service like Amazon RDS or DynamoDB. This tier is isolated within its own VPC subnet[15], with restricted access controlled by network policies and IAM roles to ensure security. Data backup, maintenance, and scalability are managed by AWS, which offloads these operational burdens from the application team, allowing them to focus on development. The application layer can securely connect to this database via Kubernetes secrets and VPC endpoints, safeguarding data in transit[21].

By deploying this architecture on AWS EKS, organizations gain the benefits of Kubernetes' robust orchestration and AWS- managed services, offering a scalable, resilient, and secure foundation for running e-commerce or other web applications. The current architecture enables the 3-tier application to achieve scalability, security, and availability by using Amazon EKS along with other AWS services[4].

A. MATHEMATICAL MODEL

The mathematical model for a 3-Tier Application deployment on AWS using Amazon EKS involves defining the system in terms of its states, inputs, outputs, and relationships between these elements. The model incorporates factors such as request handling, resource allocation, and response times, which are influenced by traffic, load balancing, and scaling algorithms[20].

Mathematical Model Components

1. System Definition :

Let the system be represented as S, where :
$S=\{U,R,F,D,Q,O\}$
U: Set of users interacting with the system.
R: Set of resources allocated to each layer (frontend, backend, database).
F: Set of functions executed by the application, which include processing user requests, executing business logic, and managing data.
D: Set of data stores, which includes databases and caches.
Q: Quality metrics for performance and reliability (e.g., latency, throughput, availability).
O: Set of outputs, including processed data and responses sent to users.

2. User Requests and Load Distribution :

• Let λ represent the rate of incoming requests from users (Poisson distribution for request arrivals), where:
$|U|$
$\lambda = \sum \lambda i\ i=1$
• Let μ_F be the service rate of the frontend, μ_A be the service rate of the application layer (backend), and μ_D be the service rate of the data layer (database).

3. Load Balancing Function :

Requests are distributed across frontend instances by the Application Load Balancer (ALB) :

$$\text{Load Distribution (LD)} = \lambda / \ |RF|$$

where RFR_FRF is the set of frontend resources (pods), so each frontend pod receives approximately LD requests.

4. Auto Scaling Function:

• The system uses horizontal scaling based on resource usage. If the usage of a resource exceeds a threshold T, new instances are added.
• Let C_R represent the capacity of a resource RRR.
• If $\lambda / \mu R > T \cdot CR$, then scale-out by adding new instances.
• If $\lambda / \mu R < (1-T) \cdot CR$, then scale-in by reducing instances.
• Scaling formula:

$$Rnew=Rcurrent+\text{round } (\lambda - T \cdot CR)/ \mu R$$

5. Latency and Response Time

• Total response time T_R is the sum of times taken at each layer: $TR=TF+TA+TD$
• TF: Time taken at the frontend layer.
• TA: Time taken at the application (backend) layer.
• TD: Time taken at the data layer (database).

6. Caching Efficiency

• Let C_hit represent the cache hit rate, where: Chit = Cache Hits/ Total Cache Accesses
• Data retrieval time T_D can be optimized as:

$$TD=Chit \times Tcache + (1-Chit) \times Tdatabase$$

where T_cache is the retrieval time from cache, and T_database is the retrieval time from the database.

Algorithm

1) Presentation Layer:

• Step 1: User Request: A user initiates a request by interacting with the frontend application (web or mobile). This could be an HTTP request to access a specific resource or perform an action (e.g., login, view data).
• Step 2: Traffic Routing[15] via Load Balancer: The request is received by the Application Load Balancer (ALB), which is responsible for distributing the request to the appropriate frontend pod running in Amazon EKS. The ALB ensures that traffic is evenly distributed to prevent overload on any single pod.
•Step 3: Secure Communication: The communication between the client and the frontend layer is secured using TLS/SSL to encrypt data in transit, ensuring privacy and data integrity[21].
• Step 4: Frontend Processing: The frontend pod processes the request, preparing data for presentation or user interaction, such as calling backend services for more complex operations or fetching data[16].

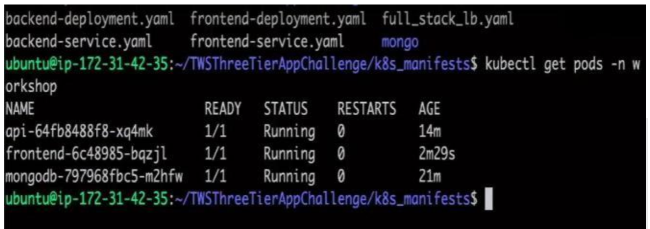
2) Application Layer:

•Step 1: API Call from Frontend: If the request requires data processing or business logic, the frontend communicates with the backend by calling an API endpoint exposed by the backend services running in the application layer[13]. These backend services are encapsulated in Kubernetes pods.
• Step 2: Request Routing to Backend: The request is routed to the correct backend pod using Kubernetes services or an Ingress Controller, ensuring the proper service endpoint is accessed.
• Step 3: Authentication & Authorization: The backend services authenticate and authorize[18] the request using credentials, tokens, or session data (e.g., JWT tokens), ensuring the user has permission to access or modify the requested data.
• Step 4: Business Logic Execution: The backend service processes the request by executing business logic, such as querying the database, performing calculations, or triggering other processes[9].
• Step 5: Cache Lookup (Optional): Before making a database call, the backend service checks the Amazon ElastiCache (e.g., Redis or Memcached) for frequently accessed data to reduce latency and avoid unnecessary database queries.[8] If data is not found in the cache, the backend proceeds to query the database layer.
• Step 6: Return Response to Frontend: Once the backend service has completed its task (whether it's retrieving data, performing calculations, or modifying resources), it sends the response back to the frontend via the API.

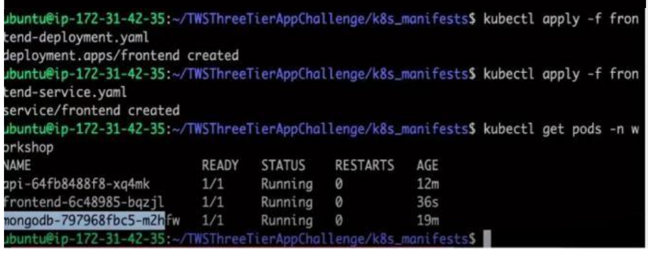
3) Data Layer:

• Step 1: Database Query: When the backend service requires data, it makes a request to the database in the data layer (e.g., Amazon RDS for SQL databases[23] or Amazon DynamoDB for NoSQL databases).
• Step 2: Data Retrieval: The database layer processes the query and retrieves the requested data, applying any necessary security checks (e.g., IAM roles, VPC peering, and security groups) to ensure secure access.

• Step 3: Data Processing (Optional): If data modification is required (e.g., inserting, updating, or deleting records), the backend service sends the modification request to the database. The database



```
backend-deployment.yaml  frontend-deployment.yaml  full_stack_lb.yaml
backend-service.yaml     frontend-service.yaml     mongo
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ kubectl get pods -n w
orkshop
NAME                        READY   STATUS    RESTARTS   AGE
api-64fb8488f8-xq4mk        1/1     Running   0          14m
frontend-6c48985-bqzjl      1/1     Running   0          2m29s
mongodb-797968fbc5-m2hfw    1/1     Running   0          21m
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ 
```



```
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ kubectl apply -f fron
tend-deployment.yaml
deployment.apps/frontend created
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ kubectl apply -f fron
tend-service.yaml
service/frontend created
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ kubectl get pods -n w
orkshop
NAME                        READY   STATUS    RESTARTS   AGE
api-64fb8488f8-xq4mk        1/1     Running   0          12m
frontend-6c48985-bqzjl      1/1     Running   0          36s
mongodb-797968fbc5-m2hfw    1/1     Running   0          19m
ubuntu@ip-172-31-42-35:~/TWSThreeTierAppChallenge/k8s_manifests$ 
```

handles transactions, ensuring data integrity, consistency, and rollback capabilities if needed.

• Step 4: Data Caching (Optional): Frequently accessed or critical data may be cached in Amazon ElastiCache[2] to reduce load on the database and improve response times.

• Step 5: Return Data to Backend: Once the data is retrieved or modified, it is sent back to the application layer (backend services) for further processing or direct response to the frontend.

4) Overall Workflow:

• Step 1: User Interaction: The user interacts with the frontend application, triggering a request.

• Step 2: Request Handling: The ALB[22] forwards the request to the appropriate frontend pod.

• Step 3: Backend Processing: The frontend communicates with backend services through APIs, which may involve accessing the data layer for storage or retrieval of data.

• Step 4: Data Layer Interaction: The backend interacts with the database to fetch, modify, or delete data.

• Step 5: Response Return: Once the backend processing is complete, the data is sent back to the frontend, which then presents it to the user.

IV. Result

Cloud computing not only creates issues but also advances security. The improvement is demonstrated in three areas: economic advancements, technological developments, and security regulation strategies. Customers, service vendors, and even government authorities should all have legitimate security needs, according to the development of technical concepts. Both cloud service providers and users have varied security requirements. The deployment of the 3-tier application on AWS using Amazon EKS successfully demonstrated the advantages of cloud-native technologies and container orchestration for modern application architecture. By leveraging Amazon EKS, Docker, and Kubernetes, we achieved a scalable, fault-tolerant, and automated system that efficiently manages containerized services in a dynamic cloud environment.

Snapshots of Output

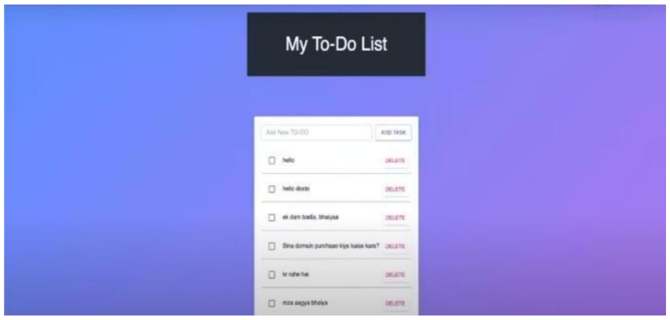
1. Successful Deployment of all 3 Tiers on AWS Server:

Fig.2 Successful Deployment of all tiers on AWS

2. Checking the status of running all the 3:

Fig.3 Checking the status of running of all three tiers

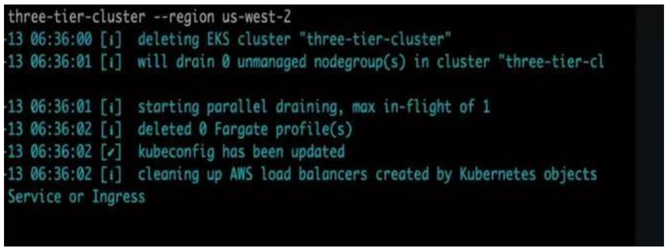
GUI of the application after the deployment and search on port 8080:



hello
hello dosto
ek dam badia, bhaiyaa
Bina domain purchase kiye kaise kare?
kr rahe hai
mza aagya bhaiya

Fig.4 GUI of the running application after Deployment

3. After all the process successfully done the last step is of garbage collection:



```
three-tier-cluster --region us-west-2
-13 06:36:00 [i]  deleting EKS cluster "three-tier-cluster"
-13 06:36:01 [i]  will drain 0 unmanaged nodegroup(s) in cluster "three-tier-cl

-13 06:36:01 [i]  starting parallel draining, max in-flight of 1
-13 06:36:02 [i]  deleted 0 Fargate profile(s)
-13 06:36:02 [✓]  kubeconfig has been updated
-13 06:36:02 [i]  cleaning up AWS load balancers created by Kubernetes objects
Service or Ingress
```

Fig.5 Garbage Collection after successful deployment

V. Conclusion

In conclusion, deploying a three-tier application architecture on AWS using Amazon EKS provides a highly resilient, scalable, and secure framework for supporting complex, user-driven applications. This architecture divides the application into three distinct layers—Web, Application, and Database Tiers—each serving specific roles in user interaction, business logic, and data storage. By running this structure on Amazon EKS[1], we leverage Kubernetes' powerful orchestration capabilities, enabling automatic scaling, load balancing, and self-healing features that ensure continuous availability and robust performance, even under high demand or in cases of hardware failure. AWS's extensive infrastructure and services complement this by offering high availability and multi-region redundancy, further minimizing downtime and enhancing user experience. Automation and efficiency are core benefits of this setup, with Infrastructure as Code (IaC) facilitating easy, repeatable deployments and environment provisioning. CI/CD pipelines streamline the application lifecycle, from code integration to deployment, allowing the team to deliver new features, updates, and fixes rapidly and consistently. Managed services like Amazon RDS for the Database Tier offload significant operational tasks such as backups, patching, and scaling, reducing manual management requirements and freeing up resources to focus on development[26]. AWS's comprehensive security measures, combined with Kubernetes' network policies and IAM roles, provide a multi-layered security approach that protects the application, data, and infrastructure from unauthorized access and potential vulnerabilities.

VI. Acknowledgement

We would like to express our special thanks of gratitude to our guide Prof. Shikha Dwivedi for their able guidance and support. We would also like to extend our gratitude to the Principal Mr .R.D. Kanphade for providing us with all the facilities that was required. We would also like to acknowledge with much appreciation the role of our staff. We sincerely thank our parents and our friends who have been always Helping and encouraging us in every situation.

VII. References

[1] G. Aldering, G. Adam, P. Antilogus, P. Astier, R. Bacon,S. Bongard, C. Bonnaud,
Y. Copin,D. Hardin, F. Henault, D.A. Howell, J. Lemonnier, J. Levy, S.C. Loken, P.E. Nugent,R. Pain, A.Pecontal, E. Pecontal, S. Perlmutter,R.M. Quimby, K. Schahmaneche, G. Smadja andW.M.Wood-Vasey, Overview of the Nearby Supernova Factory, in: TheSociety of Photo-Optical Instrumentation Engineers (SPIE) Conference, J.A. Tyson and S. Wolff, eds, Society of Photo-Optical Instrumentation Engineers(SPIE) Conference Series, Vol. 4836, SPIE, Bellingham, WA, 2002, pp. 61– 72.

[2] Amazon Web Services, http://aws.amazon.com.

[3] Amazon EBS, http://aws.amazon.com/ebs.

[4] Amazon EC2, http://aws.amazon.com/ec2.

[5] Amazon S3, http://aws.amazon.com/ebs/s3.

[6] E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney,
J. Du Croz, S. Hammarling,J. Demmel, C. Bischof and D. Sorensen, LAPACK: a portable linear algebra library for highperformance computers, in: Proceedings of Supercomputing' 90, IEEE, New York, NY, 2002,pp. 2–11.

[7] C. Aragon, S. Poon, G. Aldering, R. Thomas and R. Quimby, using visual analytics to develop situation awareness in astrophysics, Information Visualization 8(1) (2009), 30–41.

[8] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski,G. Lee, D. Patterson, A.Rabkin, I. Stoica et al., Above the clouds: a Berkeley view of cloud computing, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, CA, 2009.

[9] CFIT SIO, http://heasarc.nasa.gov/docs/software/fitsio/fitsio.html.

[10] http://www.klientsolutech.com/importance-of-cloud- computing-worldwide.

[11] https://www.itproportal.com/2013/05/02/top-10-tips-why-

file:///C:/Users/chakrmx2/Downloads/BTS%20Mobile%20 App% 20Security%20 Guidelines.pdf.

[12] https://www.stratoscale.com/blog/cloud/9-ways-cloud-improves-productivity.

[13] Fig1: https://aws.amazon.com/blogs/apn/cloud-deduplication- on-demand- storreduce-anapn-technology-partner.

[14]
you- should-use-the- cloud-andhow-to-do-it-securely.

Fig2:https://docs.aws.amazon.com/storagegateway/latest/usergu ide/StorageGatew ayConcepts.html

[12] https://aws.amazon.com/choosing-a-cloud-platform/#content_distribution

[13]Fig3: https://aws.amazon.com/backup.

[14] Fig4: https://aws.amazon.com/blogs/big-data/automating-analytic-workflows-on- aws.

[15] Fig5: https://aws.amazon.com/blogs/architecture/optimizing-a- lift-and-shift-for- cost.

[16]Fig7: https://aws.amazon.com/autoscaling.

[17]Fig 8 & 9:
https://media.amazonwebservices.com/AWS_Disaster_Recovery. pdf

[18] Mukherjee, S. (2019). How IT allows E-Participation in Policy- Making Process. arXiv preprint arXiv:1903.00831.

[19] Mukherjee, S. (2019). Popular SQL Server Database EncryptionChoices:preprint
[20]