Journal of Advances in Developmental Research (IJAIDR)

E-ISSN: 0976-4844 ● Website: www.ijaidr.com ● Email: editor@ijaidr.com

Implementing Serverless Architectures for Ultra-Low Latency Data Pipelines in Multiplayer Gaming Environments

Urvangkumar Kothari

Data Engineer
Las Vegas, NV, USA.
Email: urvangkothari87@gmail.com

Abstract:
Real-time, competitive gaming has also resulted in higher demands of ultra-low latency in multiplayer gaming environments. Scalability and performance of the Traditional client-server models are a problem when the number of players involved and the speed of interactions is high, which introduces problems of latency, lag and synchronization errors. The current paper discusses the usage of serverless architectures as the means of addressing such issues. Serverless computing fully manages the infrastructure, providing developers with an opportunity to write event-driven code, automatically scaling the resources depending on the demand. AWS Lambda, Azure Functions, and Google Cloud Functions are services that offer low latency and scalable solutions to real-time multiplayer games. This paper presents an architectural overview, design, and implementation of serverless systems, analyses the performance monitor, and finally states the benefits of serverless computing when used in serving dynamical gaming workloads. It also discusses how serverless technologies may affect multiplayer gaming industry in future by improving scalability, cutting down operational costs and ensuring uninterrupted gaming experience to the players.

Keywords: Serverless Architecture, Low-Latency, Multiplayer Games, Real-Time Data Pipelines, AWS Lambda, Azure Functions, Google Cloud Functions, Cloud Computing, Scalability, Event-Driven Code.

I. INTRODUCTION

Real-time, competitive games and multiplayer experiences have driven the demand of ultra-low latency in multiplayer gaming. Any lag among the players, even a little bit, can greatly interfere with the game process, causing irritation. Synchronization problems due to latency issues such as lag, jitter and packet loss may lead to a poor quality of the game and players quitting, which means lost engagement to the developers [1].

The classic client-server architectures are suitable on the small-scale games but have trouble with the scale in the high speed and high-volume multi-player interactions. A server can easily turn into a bottleneck, which is incapable of consuming the intensive volumes of real-time data produced by numerous players. Also, dedicated servers can cause over-provisioning, underutilization, and high latency at peak demand, with the additional maintenance expense. Serverless computing and cloud-native architectures can provide scalability and flexibility to meet these challenges.

Serverless computing provides abstractions over infrastructure and developers need to write event-driven code rather than acquire and operate servers. Cloud providers such as AWS, Azure, and Google Cloud provision automatic resource adjustments according to the demand, which eliminates the task of capacity planning. This strategy reduces operating expenses and offers the flexibility required by elastic gaming workload.

Serverless compute (functions as a service), e.g., AWS Lambda, Azure Functions, and Google Cloud Functions, can be used to run low-latency game data processing, e.g., player interaction and real-time updates. The services can be combined with other cloud services such as data streaming and storage to create effective, low latency data pipelines. Due to this fact, serverless architectures provide the scalability, flexibility and performance required in modern multiplayer games, alleviating most of the drawbacks of the traditional models.

This paper presents serverless systems in multiplayer games with the emphasis on ultra-low latency data pipelines. It talks about the architecture, components and challenges of building serverless solutions and how it can be used to enhance the gaming experience by making it scalable, cost effective and responsive. In the long run, serverless architectures can transform the development of multiplayer games and create immersive experiences across the globe without any hiccups.

II. BACKGROUND

A. Traditional Architecture vs Serverless Architecture

The multiplayer games in the classic architecture pattern are based on a client-server architecture, in which a central server is utilized to store and calculate all the game data, such as players interacting with each other, game world, and real-time updates. This centralized model mandates provisioning and administering of servers at all times to make them available and performant. The server is the primary processing device, which processes the request of multiple clients, and provides the synchronization among all players. Nevertheless, this model is associated with a number of shortcomings, especially in terms of scale and latency [2].

The main disadvantage with the older server-based architectures is the inability to scale easily in response to different degrees of demand. When the number of players grows, the server may be unable to handle all the requests, causing high response time and latency. Game developers must frequently over-provision server capacity in order to handle peak loads, which results in unnecessary expenses during off-peak times to handle these challenges. Also, with a more complex game environment, the management infrastructure to support real-time data streams, player actions, and game state changes in a conventional server architecture becomes more complex to manage which can be time consuming and expensive. Moreover, when one of the servers fails, players may be interrupted, which can influence the process of gaming greatly.

Serverless computing, in turn, is a more efficient and scalable solution. Serverless architecture takes away servers that developers are supposed to manage and provision. Rather, the game backend is broken down into small, event-driven functions, invoked by particular events, e.g., player interactions or changes in the game state. The cloud provider automatically executes these functions as they are required and scales the underlying infrastructure according to the requirement. Such a strategy means that resources are only utilized when required, and optimal performance can be achieved without having to have an idle infrastructure in place during times of low demand.

The concept of serverless computing is more reasonable and less expenditure in its nature than the conventional server-based systems. Because the developers do not have to be concerned with infrastructure, they can work on the game logic code only. Serverless automatically perform scaling, load balancing, and fault tolerance. In addition, they enable game developers to scale in response to player activity peaks without excessive provisioning of resources, which lowers operational expenses. Besides, serverless platforms are highly compatible with other cloud-based services like real-time data streaming, storage services, and edge computing, which means that they can easily manage the requirements of multiplayer gaming environments.

B. AWS Lambda, Azure Functions, Google Cloud Functions

The broader point to note about the realization of serverless architectures is the utilization of cloud-agnostic serverless compute offerings including “AWS Lambda, Azure Functions, and Google Cloud Functions”. They are event-driven computing services that enable developers to run code in response to different events, without having to think about infrastructure [3].

1) AWS Lambda: Very popular serverless compute service that automatically scales with events sent to it. It enables developers to execute backend code in reaction to definite provokes, e.g., player actions, game state changes, or database modifications. Lambda is flexible enough to work with many different programming languages and it can be easily integrated with other AWS services, including “AWS DynamoDB” low-latency storage or “Amazon Kinesis” data streaming. The pay-per-use pricing model of AWS Lambda makes it cost-efficient as it only charges the compute time consumed by the functions during their execution.

2) Azure Functions Serverless compute: It’s the Microsoft equivalent of serverless compute, which functions in the same way as AWS Lambda. It helps developers to develop event-driven applications, where the cloud provider executes and scales functions automatically. Azure Functions is available in many programming languages and can easily be connected to other Azure services like “Azure Cosmos DB” to store data globally distributed and in real-time with “Azure Event Hub”. It also provides such features as durable functions that enable stateful serverless applications, which are best suited in the case of complex gaming situations.

3) Google Cloud Functions: The serverless compute platform provided by Google, enabling developers to execute small pieces of code in response to events, at scale and paying only for what it calculation time. Google Cloud Functions scales well to event-driven applications, which serves as a suitable option in multiplayer gaming. It fits perfectly into the ecosystem of Google services, such as “Google Pub/Sub” to message and “Google Cloud Firestore” to store NoSQL, to offer a comprehensive solution to developing serverless data pipelines in games.

Combining AWS Lambda, Azure Functions, and Google Cloud Functions, offers the infrastructure required to develop and operate at scale, low-latency data pipelines in multiplayer games. With these serverless compute offerings, developers are able to bypass the issues of scale and latency created by more traditional server-based systems to provide real-time gaming experiences with significantly reduced infrastructure costs.

III. SERVERLESS ARCHETECTURE DESIGN

The dynamic and scalable solution A serverless design of ultra-low latency data pipelines in multiplayer gaming environments is an effective way to handle the requirements of contemporary gaming applications. Gaming systems can be made responsive and performant enough to support real-time, multiplayer interactions by taking advantage of serverless computing, data streaming, low-latency storage systems and edge computing. We propose a detailed architecture and the technologies used in building a serverless multiplayer gaming environment below [2].

A. Architecture

Serverless multiplayer gaming data pipelines A common implementation of serverless architecture is modular, with each module hosting a specific part of the real-time data processing pipeline. This architecture can be divided into few fundamental layers:

1) Player Interaction Layer: This layer is responsible towards sending events (game actions, movement, chat messages, game state changes) to the cloud backend by the game clients (players). This information is picked up through real-time messaging protocols such as WebSockets, HTTP-based APIs, or REST APIs. With every interaction of the players, an event is generated and handled by the serverless functions.

2) Event Streaming Layer: Events occurring on the players are pushed into the event streaming system where they are captured, queued and processed in real-time. This guarantees the system not to lose any event and the capability to process data upon arrival. The data streaming platforms like Apache Kafka, Amazon

MSK, AWS Kinesis, Azure Event Hub, or Google Pub/Sub are used to stream and capture this data to downstream systems.

3) *Game State Processing Layer:* This layer consists of serverless functions, e.g., AWS Lambda, Azure Functions or Google Cloud Functions that process game data. These functions have the capability of processing game logic, changing game states and computations, including player positioning, player interaction and environmental interactions. Functions are invoked automatically whenever there are new events in the data stream, so there is little latency between events and updates.

4) *Storage Layer:* After the data has been processed it must be stored so that it can be retrieved and manipulated later. This layer often consists of low-latency, serverless databases such as “AWS DynamoDB, Azure Cosmos DB, and Redis” to store game state, player data, and temporary information. These databases are optimized to support high reads and writes throughput, which means that the information about players is immediately available and in real-time.

5) *Edge Computing Layer:* Further optimization of the latency and performance can be achieved with the help of edge computing solutions. Game data and content can be processed more locally to the player with AWS Lambda or Azure CDN or Cloudflare Workers to minimize the distance and time it takes data to travel. That makes the content delivered to the player more quickly and overall improves the player experience, decreasing latency and making the game state sync faster.

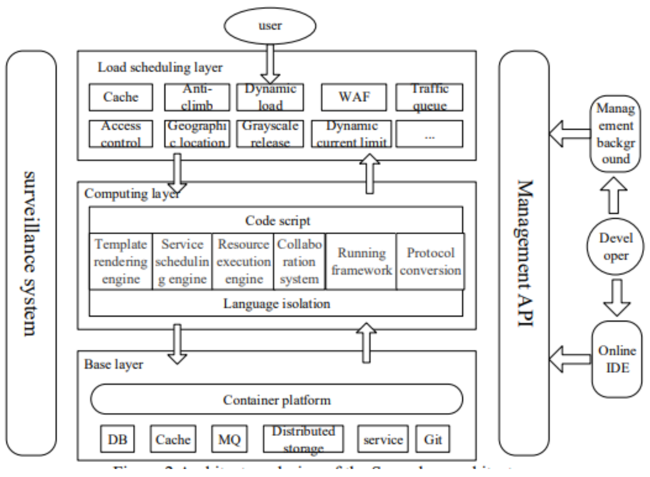

Fig.1. Serverless architecture [2]

All these layers collaborate to make sure that the interactions between the players, processing of the game state, and real-time updates can be efficiently and scale to a large number of players. This architecture is serverless, which makes it dynamic in the allocation of resources, thus cost-effective and able to scale with respect to the varying demands of players.

B. Data Streaming

In the case of a multiplayer gaming environment, data streaming is an essential component because it allows real-time transfer of game-related information amongst players and the backend servers. Serverless architecture necessitates the use of a data streaming service with high throughput whereby the action of each player is recorded and processed in real-time.

1) *Apache Kafka:* Kafka is a distributed event streaming platform that is based on open source and can process high throughput streams of real-time data. Because it is able to process a high number of messages and is durable and scalable, it is commonly utilized in serverless architectures to process gaming events. Commonly, Apache Kafka is used as a managed service, like “Amazon MSK” which is a fully managed Apache Kafka service that lets you simplify deployment and scale.

2) *Amazon MSK (Managed Streaming for Kafka):* MSK is the AWS managed service based on Kafka that can be used to gain a scalable and low-latency streaming platform to be used in multiplayer games. It also

makes sure that player and game actions are delivered to downstream processing systems with limited latency [4].
3) AWS Kinesis: Kinesis is the streaming service provided natively by AWS and meant to process data in real-time. It provides three primary services; Kinesis Data Streams, Kinesis Data Firehose and Kinesis Data Analytics which can be utilized to stream player actions and game events which are subsequently processed by AWS Lambda functions to provide real time updates to the game state. Kinesis can integrate with other AWS services freely, making the flow of data through the architecture unproblematic.
4) Azure Event Hub: Azure Event Hub is an event streaming platform (cloud native) with a throughput capacity of millions of events per second. It is best suited to multi-player games, where events throughput is demanding. Azure Event Hub can be simply combined with Azure Functions to process the game data in real-time, and this gives a durable serverless approach to gaming workloads.
5) Google Pub/Sub: Google Cloud Pub/Sub service is a messaging system that is distributed across the world and which can be used to establish real-time communication between players and backend systems. One can use it to stream game data transferred by players in real-time, without the need to consider scaling of the infrastructure.
These data streaming platforms would guarantee the smooth, low latency processing of player events, making real time communication possible between players in a multi-player game.

C. Low-Latency Storage Products

Storage is an essential factor in making sure that game data, player progress, and game state are stored with low latency. Serverless databases and caching the serverless databases and caching are best used in gaming applications where low-latency access is critical [5].
1) AWS DynamoDB: DynamoDB is a fully managed NoSQL database, which delivers high and predictable performance at scale. It is capable of high velocity and high-volume workloads and it is well suited to multiplayer games which require player profiles, game state, and other real time information to be persisted. Low-latency reads and writes that are guaranteed by DynamoDB are critical in making the update of games available to players faster.
2) Azure Cosmos DB: Cosmos DB is a multi-model, globally distributed database which supports low-latency data storage. It is also bound to scale easily and support mission-critical workloads, which makes it a decent choice in a multiplayer gaming environment. Cosmos DB with its low-latency reads and writes makes sure that the game state updates are immediately visible to players.
3) Redis: it is an open-source, in-memory key-value dataset utilized as a cache of regularly requested information. It is very high-speed and can process bulky data with very low latency. Multiplayer games use Redis to cache player data, game states and session information, in order to avoid making repeated database queries and to make the game resume faster.

D. Edge Computing Factors

Edge computing is a method of handling data nearer to the origin - i.e., nearer to the players. This reduces the time data takes to travel to and for the central server, thus, minimizing latency [6].
1) AWS Lambda @Edge: Lambda @Edge brings AWS Lambda to the global content delivery network (CDN) of AWS. This enables developers to deploy serverless functions nearer to the player, minimizing the distance between the client and the source of data. The edge helps to process data, which can then be used to update game data with little latency, enhancing the player experience.
2) Azure CDN: Azure Content Delivery Network (CDN) may be utilized together with Azure Functions to serve content and process data nearer to the player. This is critical particularly in gaming environments where the media and game resources need to be delivered fast to customers globally.
3) Cloudflare Workers: Cloudflare Workers is serverless computing platform, which allows developers to execute code at the edge, nearer to end-users. It is especially helpful with multiplayer games, where the real-time updates and the low latency of the content delivery are imperative. Edge computing lowers latency greatly, and players encounter fewer delays during interactions.

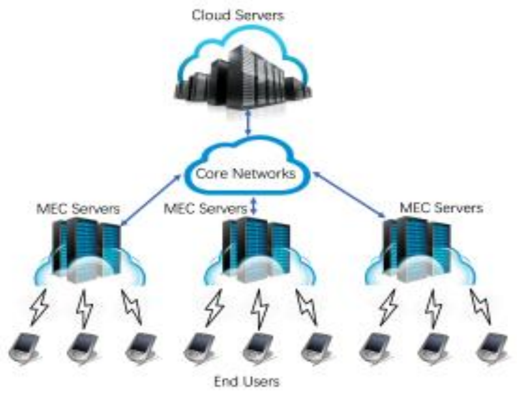

Fig.1. Edge Computing Architecture [6]

IV. IMPLEMENTATION PROBLEMS AND RESOLUTIONS

Despite the numerous advantages of serverless architectures in terms of scaling, cost-efficiency, and simplicity of deployment, there are some challenges that are also linked to them. Among the key challenges in using serverless architectures to realize ultra-low latency data pipelines in multiplayer gaming environments are cold start latency, and infrastructure management.

A. Solutions Cold Start Latency problems

One of the most evident issues in the usage of serverless functions is associated with a cold start latency problem. Cold start Cold start is the situation where serverless function is being started and it has not been started in a while, or there is a burst in the traffic that cannot be accommodated by an existing instance, and a new one must be created to respond to a request [7]. The process of provisioning of resources and initialization of functions may take some time, which is regrettable in the scenario of real-time applications, such as multiplayer games, where a lag in milliseconds can result in an observable deterioration of user experience. To reduce the cold start latency, we can do the following:

1) Pre-Warming: One specific way of addressing the cold start issue is to do pre-warming. The process of occasionally invoking serverless functions at predetermined timers (even without a request being made by a user) in order to keep them in a state of being warm is called pre-warming. The invocation of the functions periodically makes them be prepared to receive the incoming events with minimal latency. E.g., to maintain the function warm, it might simply be needed to establish a scheduled task that will periodically (e.g., each few minutes) generate a dummy event. This will result in less time being taken in initialization of functions when actual game events arrive leading to faster response.
2) Redis Optimized Caching: Another way of minimizing the cold start latency is caching. A fast in-memory data structure store such as Redis can be implemented as a cache layer to reduce the number of invocations of serverless functions. Most requested data, such as player profiles or game state data, can be cached in Redis so that 0ms reads can be achieved without invoking serverless functions on every request. By caching the commonly accessed data, Redis can considerably reduce both the frequency and duration of the cold starts, and thus, provide reduced latency in multiplayer games. Additionally, there is the option of using Redis as a session manager which implies that data concerning a player would be readily available without invoking other serverless functions.

B. Infra as Code (IaC)

Serverless architectures rely on the underlying infrastructure, which can be complex to configure, especially in the case of large-scale deployments. The solution to this issue is the Infrastructure as Code (IaC), which enables developers to manage, automatize the infrastructure setup, deployment, and scaling of serverless applications by using the code [8].

1) *Terraform:* One of the most well-known IaC providers that enable the management of cloud resources through a declarative configuration file. Terraform allows developers to describe the infrastructure in code and thereby provision and manage resources across many environments in a similar way. With Terraform, developers can identity serverless functions (e.g., AWS Lambda, Azure Functions) and other infrastructure (e.g., databases, messaging queues). This simplifies the deployment process and turns the environment into consistent and reproducible, which reduces the likelihood of misconfigurations and human errors.
2) *Serverless Framework:* An open-source tool that is preferable in serverless architecture. It allows developers to define serverless functions and other resources (e.g., "APIs, databases, storage") in "YAML config files". The Serverless Framework handles the deployment of serverless functions and ensuring that all the necessary resources are deployed and linked. This architecture simplifies the deployment workflow and can be used to orchestrate across multiple cloud providers, which means it is simple to achieve a serverless experience.
3) *AWS CloudFormation:* An IaC provider provided by Amazon Web Services to define and provision AWS infrastructure as templates. CloudFormation allows developers to use templates which describe serverless resources such as AWS Lambda functions, API Gateway endpoints, DynamoDB tables, etc. These templates are version controlled and hence infrastructure turns out to be repeatable, manageable and alike at different stages of development and deployment. CloudFormation makes it easier to manage the infrastructure of serverless apps and also provides automated rollbacks in the event that some deployment fails.

Challenge	**Solution**
Cold Start Latency	Pre-Warming
Cold Start Latency	Optimized Caching with Redis
Infrastructure Management	Terraform
Infrastructure Management	Serverless Framework
Infrastructure Management	AWS CloudFormation

Table 1: Implementation Challenges & Solutions

V. PERFORMANCE ANALYSIS AND CASE STUDY

A. Monitoring/Observability Tools

In order to achieve serverless architecture that is running in the real time multiplayer gaming environment, there arises a need to have the functionality of monitoring the performance of the various parts of the system. This kind of monitoring enables making sure that the potentially emerging issues could be noted and resolved instantly, not to interrupt the game process [9]. The following monitoring and observability tools are crucial in the tracking and the analysis of whether the system is healthy:
1) *AWS CloudWatch:* A powerful monitoring service, which provides real time understanding of the performances of applications including resource utilization, error rate, and latency of AWS services like Lambda, DynamoDB, and Kinesis. CloudWatch can be implemented in the instance of multiplayer gaming to keep track of the Lag functions including duration of functions execution, number of requests, and percentage of errors to ensure that the serverless functions are working as envisaged. Another feature of CloudWatch is automated alarms that notifies the developers when any of the performance bounds are crossed that helps in maintaining a responsive system.
2) *Azure Monitor:* Similar to AWS CloudWatch, and it gives full-stack monitoring of Azure services. It collects application, network resources and infrastructure information to provide a profound image of the performance. Azure monitor helps in the monitoring of real time performance of functions as Azure Functions, Cosmos DB as well as Event Hubs. Based on the knowledge offered by the Azure Monitor, game developers will get a proactive chance to identify potential bottlenecks, latencies, or resource bottlenecks, thus, ensuring that the system is scalable and prepared to respond to player actions.

3) Grafana: An open-source visualization platform which can be connected with a large number of various data sources, including CloudWatch and Azure Monitor, to provide interactive dashboards and visualizations. Grafana can as well be used in multiplayer games to give a visualization of Response time, Function execution time, Error percentage, and User activity. Grafana enables developers to understand more about the performance of their serverless functions in real time by providing detailed and customizable visualizations to them so they can decide how to optimize them.

B. Comparative Visualizations, Performance Metrics Analysis

To measure the effectiveness of the serverless architecture, as well as to ensure the optimal operation, one has to focus on the main performance indicators, which consist of latency, throughput, scalability, and cost efficiency. Such numbers are used to investigate the degree of archive suitability to the demands of real-time multiplayer games [10].

1) Latency: The most significant multiplayer gaming statistic. Latency is a metric which indicates the round-trip time (e.g., player actions, game state updates) between the client and server. Latency is the devil of the gaming process that should be monitored and minimized. Dynamical scalability Serverless architectures can achieve lower latency by means of edge computing and caching strategies.

2) Throughput: The number of events (e.g., player actions) that the serverless system can handle within a second is the throughput. It must be of big throughput to support the numerous player interactions and game events in real-time. With the help of such metrics as a number of messages processed by streaming services (Kinesis or Event Hub), throughput can be measured.

3) Scalability: Serverless systems are known to be highly scalable thereby enabling the automatic scaling of the resource provisioning based on the demand. When pushed to the limits, scaling capabilities of the system can be pointed out by the performance figures. This is necessary in case of multiplayer games, in peak time or probably when some new feature has been introduced.

4) Cost Efficiency: The costs model that comes with serverless architectures (pay-per-use) is among its greatest assets since it gets rid of the issue of resource utilization. The cost-based measures help to keep track of the spending especially during the retrograde or expansion of the resources during the peak demands. Comparative visualizations can be used to gain a picture of cost saving by using serverless solutions as opposed to the traditional server-based system.

These comparisons can help developers to understand which cloud provider (e.g., AWS vs. Azure), or serverless framework (e.g., Lambda vs. Azure Functions) can offer the best performance when handling specific multiplayer gaming scenarios.

VI. FUTURE INTEGRATIONS

The future of serverless architectures is yet to be written, and several integration opportunities are thrilling to explore, which can Space Invaders additionally enhance the performance of multiplayer gaming space.

1) AI-Enhanced Edge Computing: Another opportunity to reduce the latency even more and optimize the real-time game data processing is artificial intelligence and edge computing. AI models at the edge (e.g., on AWS Lambda Edge or Cloudflare Workers) can help game developers execute player data more efficiently, make predictions concerning player behaviors, and enhance in-game experiences without relying on centralized cloud infrastructure. Examples of this would be AI being used to predict field player movement, or optimizing real time state transitions of the game based on player actions.

2) Advance CI/CD: By their very definition, multi-player games are regularly updated, and with new features, which means that more sophisticated Continuous Integration (CI) and Continuous Deployment (CD) tools will need to be incorporated into the development pipeline. The process of code deployment, testing and monitoring is automated, such that the updates can be deployed in a short time frame without interfering with the game play.

A. CI/CD: Jenkins, GitHub Actions, AWS Code Pipeline

1) Jenkins: Jenkins is an open-source automation server very prevalent in the CI/CD pipelines. It helps in automating tasks such as compilation and testing of codes, and deployment of codes. serverless architectures

Game developers can create serverless architectures using Jenkins by using Jenkins's plugins to integrate it with AWS Lambda, Azure Functions and other cloud services. Jenkins help to automate build and deployment process that helps to make release of game updates regular and quick without any human participation.

2) GitHub Actions: GitHub Actions is the service which enables the creation of the CI/CD pipelines directly in the GitHub repositories. It allows developers to offer automation in testing, building, and deployment of serverless applications. With multiplayer games, one option is GitHub Actions, which can automatically deploy the game to the cloud (AWS, Azure, etc.) whenever a change has been committed to the codebase, making the updates faster and reducing the amount of downtime players experience.

3) AWS Code Pipeline: AWS Code Pipeline is a fully managed continuous integration/continuous delivery (CI/CD) pipeline service that automates the build, test, and deployment of serverless applications on AWS. Code Pipeline allows game developers to define workflow that will be incorporated in continuous delivery and it is integrated with AWS Lambda, Amazon API Gateway among other AWS services. Using Code Pipeline, developers can publish game updates, bug fixes and new feature releases with minimal downtimes to ensure that the gaming experience is not affected.

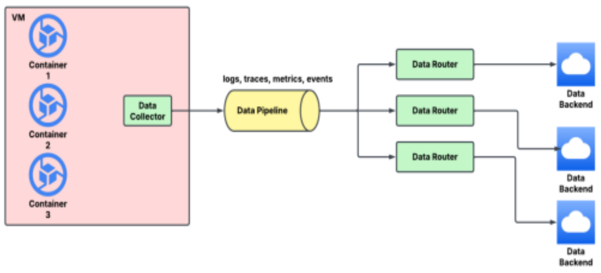

Fig.2. A Layered Observability Framework Depicting Data Collection, Storage, And Analysis [9]

CONCLUSION

Multiplayer gaming relies upon the ability to scale efficiently and operate real time data with lower latency in the future. Serverless usage and the adherence to the use of powerful monitoring and observability tools, as well as CI/CD practices, are needed to manage the heavy traffic demands of the modern gaming world. AI and edge computing combined with advanced CI/CD tools will ensure further performance and innovation in the multiplayer game as it will provide a smooth, lag-free game to players around the world.

REFERENCES:

[1] M. G. S. a. T. A. Aslanpour, " Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research.," Internet of Things, 12, p.100273., 2020.

[2] L. P. Y. a. Z. J. Jiang, "Overview of serverless architecture research.," In Journal of Physics: Conference Series (Vol. 1453, No. 1, p. 012119). IOP Publishing., 2020.

[3] M. G. A. Z. A. B. B. a. F. K. Malawski, "Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions. Future Generation Computer Systems, 110, pp.502-514.," 2020.

[4] J. a. L. X. Peng, "Disease and death monitoring on Amazon managed streaming for Apache Kafka. In Proceedings of the 2nd International Symposium on Artificial Intelligence for Medicine Sciences (pp. 24-27).," 2021, October.

[5] F. Gessert, "Low latency for cloud data management (Doctoral dissertation, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky).," 2018.

[6] W. L. F. H. X. H. W. L. C. L. J. a. Y. X. Yu, "A survey on the edge computing for the Internet of Things. IEEE access, 6, pp.6900-6919.," 2017.

[7] J. a. J. S. Gope, "A survey on solving cold start problem in recommender systems. In 2017 International Conference on Computing, Communication and Automation (ICCCA) (pp. 133-138). IEEE," 2017, May.

[8] S. Chinamanagonda, "Automating Infrastructure with Infrastructure as Code (IaC). Available at SSRN 4986767.," 2019.

[9] A. .. Gogineni, "Observability Driven Incident Management for Cloud-native Application Reliability. Int. J. Innov. Res. Eng. Multidiscip. Phys. Sci, 9(2).," 2021.

[10] S. M. S. D. F. K. E. a. L. T. .. Murugesan, " Deepcompare: Visual and interactive comparison of deep learning model performance. IEEE computer graphics and applications, 39(5), pp.47-59.," 2019.