CSC/ECE 547- Cloud Project

Cloud Architecture for On-Demand Grocery Delivery Services

Team member 1:
Name: Tanishq Todkar
Unity ID: ttodkar
Student ID: 200537017

Team member 2:
Name: Tripurashree Manjunatha
Unity ID: tmysore
Student ID: 200537395

“We, the team members, understand that copying & pasting material from any source in our project is an allowed practice; we understand that not properly quoting the source constitutes plagiarism.”

“All team members attest that we have properly quoted the sources in every sentence/paragraph we have copy & pasted in our report. We further attest that we did not change words to make copy & pasted material appear as our work.”

Table Of Contents

1. Introduction

1.1 Motivation

The primary motivation for this project stems from the growing demand for reliable, scalable, and cost-efficient cloud solutions in the fast-paced grocery delivery industry. Services like Blinkit operate in a hyperlocal, on-demand environment, where the challenges of peak demand, dynamic workloads, and operational costs require innovative solutions. This project addresses these challenges by designing an optimized cloud infrastructure that not only ensures 24/7 availability but also incorporates sustainability, real-time responsiveness, and future scalability. By exploring advanced cloud strategies, this work aims to improve user experience, business efficiency, and environmental impact.

1.2 Executive Summary

This project focuses on developing a cloud-based solution for Blinkit, a 24/7 grocery delivery service that requires robust infrastructure to handle fluctuating demand. The design leverages Amazon Web Services (AWS) to provide dynamic scaling, high availability, real-time data processing, and cost optimization. Key components include services like AWS Lambda for serverless computing, Amazon DynamoDB for real-time inventory management, and Elastic Load Balancing for efficient traffic distribution. By aligning with AWS Well-Architected Framework principles, the project ensures reliability, security, performance efficiency, and sustainability. Targeted at cloud architects and business decision-makers, this summary highlights the potential for cost savings, improved scalability, and enhanced service continuity.

2. Problem Description

2.1 The Problem

High-Level Problem:
Blinkit, an on-demand hyperlocal delivery service that operates 24/7, needs a scalable and reliable cloud infrastructure to support its growing user base, particularly during peak demand periods such as festivals, flash sales, or city-wide lockdowns (like during COVID-19). The primary challenge is ensuring that the application [1] can handle spikes in order requests, efficiently allocate delivery resources, and process payments without any downtime. The cloud solution must also support real-time inventory management 2, dynamic pricing, and delivery tracking while keeping operational costs low.

2.2 Business Requirements

BR1: Optimize operational costs while maintaining scalability for fluctuating demands.
BR2: Accommodate time-varying workloads, especially during peak hours or events.
BR3: Ensure 24/7 availability of the platform for order processing and fulfillment.
BR4: Deliver high performance with fast response times and high throughput for multiple transactions and real-time updates.

BR5: Implement secure operations to protect customer data, payment details, and delivery addresses.
BR6: Maintain vendor flexibility to avoid dependency on a single cloud provider.
BR7: Ensure cost-efficient allocation of compute and storage resources, automatically scaling down during off-peak times.
BR8: Use cost-efficient data storage solutions for frequently accessed vs. archived order and delivery data, balancing performance with expense.
BR9: Provide detailed billing insights for different operational areas, enabling visibility and optimization of resource usage costs.
BR10: Ensure that critical components, such as payment processing and delivery tracking, have minimal downtime, even under high load conditions.
BR11: Detect and automatically recover from errors to maintain uninterrupted order processing and real-time delivery tracking.
BR12: Implement data redundancy across multiple geographical regions to protect against localized disruptions.
BR13: Ensure fast response times for APIs critical to the user experience, such as real-time delivery tracking, through optimized resource utilization.
BR14: Maintain real-time synchronization across data sources to support features like instant inventory updates and location tracking for deliveries.
BR15: Implement efficient caching mechanisms to reduce database load and improve app responsiveness, particularly for repeated customer queries.
BR16: Minimize latency in location-based services for precise, real-time delivery status updates and ETAs.
BR17: Define clear data retention and deletion policies for sensitive customer and transaction data, ensuring compliance with data privacy laws.
BR18: Detect and prevent fraudulent activities, particularly around payment and account creation processes, to ensure safe operations.
BR19: Ensure users and administrators access only the information and features they are authorized to use, based on their roles within the organization.
BR20: Continuously monitor the app’s security posture, detecting and mitigating any risks in real time.
BR21: Implement automated tracking and reporting for all technical issues, aiding quick resolution and informed decision-making.
BR22: Automate end-to-end quality assurance tests for critical workflows (e.g., order placement, payment processing, delivery tracking) before production deployments.
BR23: Conduct regular audits to ensure compliance with regulatory standards, particularly around payment data handling and user data privacy.
BR24: Design the app to maintain performance and avoid bottlenecks during unexpected demand surges.
BR25: Provide timely notifications and alerts to users for important events like order status updates, anticipated delivery times, or supply delays.
BR26: Enable self-service options for users to troubleshoot issues, such as order cancellation or delivery re-routing, without involving customer support.

BR27: Monitor and optimize energy consumption across data centers and computing resources, aligning with sustainability goals.
BR28: Prioritize cloud providers with recyclable or renewable energy resources, contributing to reduced environmental impact.
BR29: Support multi-tenancy to separate and optimize resources for different user groups, such as customers, drivers, and partners.
BR30: Enable users to customize their app experiences (e.g., notification preferences, order scheduling) without compromising the app's scalability.
BR31: Ensure the infrastructure can accommodate future feature updates and expansions, such as adding new services or integrating with third-party applications. [3][4]

2.3 Technical Requirements

BR1: Optimize operational costs while maintaining scalability for fluctuating demands.

- **TR1.1**: Implement cost tracking and optimization strategies to identify areas of overspending, ensuring cost-efficiency in handling fluctuating workloads.
- **TR1.2**: Enable dynamic resource allocation to match compute resources with demand, minimizing waste while scaling to meet peak traffic.
- **TR1.3**: Minimize costs for idle resources during low-demand periods by deallocating unused capacity, reducing operational expenses.

BR2: Accommodate time-varying workloads, especially during peak hours or events.

- **TR2.1**: Ensure system capacity can scale rapidly to accommodate sudden surges, preventing disruptions during peak traffic.
- **TR2.2**: Implement efficient load distribution mechanisms to maintain even workload allocation, optimizing resource use during variable traffic.
- **TR2.3**: Optimize system performance under peak load conditions to ensure smooth operation and user satisfaction during high-demand events.

BR3: Ensure 24/7 availability of the platform for order processing and fulfillment.

- **TR3.1**: Implement high availability mechanisms to minimize downtime and provide uninterrupted services.
- **TR3.2**: Ensure data persistence and accessibility across multiple locations to prevent loss of critical information during outages.

- **TR3.3**: Provide redundancy for critical components to quickly recover from failures, ensuring consistent platform reliability.

BR4: Deliver high performance with fast response times and high throughput for multiple transactions and real-time updates.

- **TR4.1**: Optimize database performance to handle high transaction volumes without delays, ensuring fast response times.
- **TR4.2**: Implement efficient data retrieval mechanisms to speed up access to frequently requested information.
- **TR4.3**: Enable parallel processing for non-critical tasks to reduce latency in critical workflows, ensuring better throughput.

BR5: Implement secure operations to protect customer data, payment details, and delivery addresses.

- **TR5.1**: Ensure data protection at rest and in transit, safeguarding sensitive information from unauthorized access.
- **TR5.2**: Implement robust authentication and authorization mechanisms to prevent unauthorized access and data breaches.
- **TR5.3**: Conduct regular security assessments to identify vulnerabilities, ensuring compliance with industry standards.

BR6: Maintain vendor flexibility to avoid dependency on a single cloud provider.

- **TR6.1**: Design the architecture using provider-agnostic principles to support migrations and avoid lock-in.
- **TR6.2**: Ensure system components can operate seamlessly across different cloud providers, supporting multi-cloud strategies.
- **TR6.3**: Implement consistent operational practices across infrastructures to simplify management and ensure compatibility.

BR7: Resource Allocation Automation

- **TR7.1**: Implement real-time tracking of resource usage to dynamically adjust allocation, preventing under- or over-utilization.
- **TR7.2**: Integrate automated shutdown for non-critical workloads during off-peak times to conserve energy and reduce costs.

BR8: Efficient Data Storage

- **TR8.1**: Classify data by access frequency to optimize storage costs while ensuring performance for frequently accessed data.
- **TR8.2**: Implement automated archiving for older data to maintain cost-efficiency while enabling retrieval when needed.

BR9: Billing Transparency

- **TR9.1**: Create a cost dashboard to provide clear visibility into resource usage, helping optimize spending across operations.
- **TR9.2**: Provide real-time cost anomaly notifications to prevent overspending and enable proactive cost management.

BR10: Minimized Service Downtime

- **TR10.1**: Employ a failover mechanism to alternate between instances, ensuring seamless service continuity during failures.
- **TR10.2**: Regularly test high-availability configurations to validate the system's ability to handle disruptions.

BR11: Error Detection and Recovery

- **TR11.1**: Implement proactive error detection with alerts to identify and mitigate issues before they impact users.
- **TR11.2**: Configure continuous logging to capture error details, ensuring rapid diagnosis and recovery.

BR12: Geographical Redundancy

- **TR12.1**: Enable data replication across regions to safeguard information and maintain availability during localized failures.
- **TR12.2**: Ensure regional failover capabilities to prevent service disruptions in case of regional outages.

BR13: Optimized API Performance

- **TR13.1**: Implement caching to reduce API latency and improve response times for frequently accessed endpoints.
- **TR13.2**: Conduct load testing to validate API performance under varying traffic conditions, ensuring consistent behavior.

BR14: Real-Time Data Synchronization

- **TR14.1**: Use a real-time data propagation mechanism to minimize delays and maintain data accuracy across systems.
- **TR14.2**: Establish conflict resolution protocols to handle inconsistencies in distributed systems.

BR15: Efficient Cache Management

- **TR15.1**: Employ layered caching to reduce strain on databases, improving application responsiveness.

- **TR15.2**: Monitor cache hit/miss ratios to dynamically adjust configurations based on usage.

BR16: Low Latency for Location-Based Services

- **TR16.1**: Position endpoints near users to minimize latency, ensuring faster response times for location-based services.
- **TR16.2**: Leverage edge computing to process real-time location tracking, enhancing responsiveness.

BR17: Secure Data Retention and Deletion Policies

- **TR17.1**: Establish automated deletion policies for sensitive data, ensuring compliance with privacy regulations.
- **TR17.2**: Apply encryption to all stored data to protect against unauthorized access.

BR18: Protection Against Fraudulent Activities

- **TR18.1**: Use behavior analysis to detect unusual activity patterns, preventing fraudulent actions.
- **TR18.2**: Implement MFA for sensitive actions to enhance security.

BR19: Role-Based Access Control (RBAC)

- **TR19.1**: Define RBAC policies to ensure users access only the data and features necessary for their roles.
- **TR19.2**: Regularly audit roles and permissions to maintain compliance and reduce access-related risks.

BR20: Continuous Security Monitoring

- **TR20.1**: Set up monitoring to detect vulnerabilities in real time, ensuring proactive remediation.
- **TR20.2**: Conduct periodic scans and testing to identify and fix potential threats.

BR21: Automated Issue Tracking and Reporting

- **TR21.1**: Enable automated tracking and prioritization of application errors to quickly identify and analyze issues, ensuring seamless operations by reducing downtime.
- **TR21.2**: Provide visibility to issue resolution progress, ensuring teams are alerted for timely responses and maintaining system reliability under varying conditions.

BR22: Quality Assurance Testing Automation

- **TR22.1**: Integrate automated testing to identify potential issues in user flows and security before production releases, reducing the risk of defects in critical workflows.
- **TR22.2**: Schedule regular regression tests to catch defects early, maintaining a stable application experience as new features or updates are deployed.

BR23: Regulatory Compliance Audits

- **TR23.1**: Document processes for data handling to ensure compliance with regional laws, minimizing legal risks and protecting user data integrity.
- **TR23.2**: Conduct regular audits to identify and address gaps in compliance, helping the organization adhere to regulatory standards and maintain trust.

BR24: Resilience Under High Demand

- **TR24.1**: Implement automated workload balancing to distribute traffic efficiently, ensuring smooth performance during peak demand periods.
- **TR24.2**: Schedule periodic stress tests to assess and improve the system's ability to handle maximum loads, preventing bottlenecks or failures under high usage.

BR25: User-Friendly Alert and Notification System

- **TR25.1**: Implement in-app alerts and push notifications to deliver critical updates promptly, improving user engagement and experience.
- **TR25.2**: Allow users to customize notification preferences, ensuring that alerts are relevant and non-intrusive, enhancing usability.

BR26: Self-Service Troubleshooting

- **TR26.1**: Offer a help center with FAQs and troubleshooting guides to empower users to resolve common issues independently, reducing support overhead.
- **TR26.2**: Enable automated resolution for basic issues to provide immediate solutions without manual intervention, improving customer satisfaction.

BR27: Energy-Efficient Resource Usage

- **TR27.1**: Enable dynamic resource scaling to optimize energy consumption during off-peak hours, supporting operational cost reduction and sustainability goals.
- **TR27.2**: Monitor energy usage across cloud resources to ensure efficient allocation, reducing waste and aligning with environmental objectives.

BR28: Recyclable or Renewable Resources

- **TR28.1**: Prioritize cloud providers committed to renewable energy sources to reduce the carbon footprint of operations, aligning with sustainability goals.

- **TR28.2**: Implement resource recycling practices in data centers to minimize waste and promote environmentally friendly operations.

BR29: Multi-Tenancy Support

- **TR29.1**: Define tenant-specific configurations to enable resource isolation and track usage across user groups, ensuring performance and cost visibility.
- **TR29.2**: Provide usage reports for each tenant to offer insights into service quality, supporting better operational decisions.

BR30: Customizable User Experiences

- **TR30.1**: Enable users to save and apply personalized settings to improve convenience and cater to individual preferences.
- **TR30.2**: Allow flexible UI configurations based on historical usage patterns, enhancing usability and engagement.

BR31: Flexible Infrastructure for Future Enhancements

- **TR31.1**: Design system components to support seamless integration of new features, ensuring scalability without disrupting operations.
- **TR31.2**: Use a microservices architecture to enable independent scaling and maintenance, facilitating iterative enhancements over time.

[5][6]

2.4 Tradeoffs

2.4.1 Scalability vs. Data Consistency:

Tradeoff: Accommodating fluctuating demand (BR2) while maintaining data consistency (BR14) introduces challenges. Scalability often requires distributed architectures where ensuring strong consistency across all nodes can increase latency and reduce responsiveness.

Justification: A balance must be struck by prioritizing consistency for critical operations like payment processing, while allowing eventual consistency for less critical operations such as inventory updates.

2.4.2 Real-Time Insights vs. Complexity:

Tradeoff: Providing real-time updates for delivery tracking and order processing (BR4) can add significant architectural complexity, especially in handling large volumes of real-time data and synchronizing it across regions.

Justification: A decision is needed to determine the acceptable tradeoff between architectural simplicity and the level of real-time capability required, considering user experience and operational goals.

2.4.3 Development Speed vs. Scalability:

Tradeoff: Rapid development to meet immediate business needs (BR5) might result in architectural choices that limit future scalability (BR1). Using less modular or tightly coupled systems can expedite development but create bottlenecks as demand grows.

Justification: A careful assessment is needed to decide how much short-term speed to prioritize over the ability to scale effectively in the future.

2.4.4 High Availability vs. Latency:

Tradeoff: Ensuring platform availability across multiple regions (BR12) can increase latency due to the added overhead of synchronizing data and operations between geographically dispersed nodes.

Justification: For latency-sensitive services, some level of regional independence might need to be accepted, while global synchronization is reserved for less time-critical functions.

2.4.5 Data Retention vs. Cost Optimization

Tradeoff: Retaining historical data for compliance and analytics (BR22) increases storage and processing costs (BR18). Decisions around data retention policies, such as whether to archive or delete certain types of data, affect the overall cost structure.

Justification: A balance between long-term storage costs and the value of retaining specific datasets is essential to align with compliance and business intelligence goals.

3 Provider Selection [0%]

Selecting an appropriate cloud provider is paramount for a grocery delivery service looking to optimize its operations and provide seamless services to customers. A cloud provider forms the foundation of a company's digital infrastructure, playing a critical role in ensuring reliability, scalability, and security across all services. Providers not only manage the availability of

essential cloud resources but also streamline operations through robust onboarding, configuration, and support mechanisms for service consumers. The decision-making process in choosing a provider should be rooted in the specific model needed, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS), depending on the application demands.

3.1 Criteria for choosing a provider

When selecting a cloud provider for a grocery delivery service like Blinkit, we must consider a variety of metrics to ensure that the chosen platform meets the business and technical needs effectively.

Key metrics to evaluate include:

1. **Scalability**: The ability to scale resources dynamically based on demand, ensuring that the cloud infrastructure can handle peak traffic periods and fluctuating workloads. Ensuring this metric satisfies **TR1.2**, **TR2.1**, and **TR2.2**.
2. **Cost Efficiency**: Optimizing costs through flexible pricing structures like spot or reserved instances, minimizing idle resource costs, and enabling cost transparency for different operational areas. This metric is supported by **TR1.1**, **TR1.3**, and **TR7.2**.
3. **Availability and Reliability**: Ensuring high availability and fault tolerance with multi-region deployment and automated failover, enabling continuous service during outages. This metric aligns with **TR3.1**, **TR3.2**, **TR10.1**, and **TR12.2**.
4. **Performance**: Maintaining low-latency, high-throughput performance for real-time services such as order processing and delivery tracking, essential for user experience. This metric is met by **TR4.1**, **TR13.1**, and **TR16.1**.
5. **Security**: Ensuring data protection with strong encryption, robust authentication, and authorization to safeguard sensitive customer and payment information, aligning with **TR5.1**, **TR5.2**, **TR18.2**, and **TR20.1**.
6. **Vendor Flexibility**: Avoiding vendor lock-in by supporting multi-cloud or hybrid environments, maintaining operational flexibility across cloud providers, and enabling future enhancements. This metric satisfies **TR6.1**, **TR6.2**, and **TR31.1**.
7. **Monitoring and Automation**: Enabling advanced monitoring and automation for resource management, error tracking, and deployment processes, which improves efficiency and response time. This metric is addressed by **TR21.1**.
8. **Support for Multi-Tenancy**: Supporting isolated environments for different customer groups with efficient billing and resource tracking, ensuring operational separation for users. This metric corresponds to **TR29.1**, and **TR29.2**.
9. **Compliance and Regulatory Support**: Providing tools and support for compliance with data privacy and security regulations, including audit capabilities, to meet operational requirements across regions. This metric aligns with **TR23.1**, **TR23.2**, and **TR17.1**.
10. **Disaster Recovery and Redundancy**: Ensuring business continuity through data replication across multiple regions and proactive error detection, with automated recovery mechanisms in place. This metric is addressed by **TR3.3**, **TR12.1**, and **TR11.1**.

These metrics should be carefully weighed against the specific needs of the business to ensure optimal performance, scalability, and cost efficiency.

3.2 Provider Comparison

Building Block	AWS	Azure	Google Could	Justification
Scalability	Highly scalable with advanced auto-scaling and serverless offerings	Highly scalable with advanced auto-scaling and serverless offerings	Effective scalability, strong with data analytics workloads	AWS's mature autoscaling and serverless options make it the leader for dynamic scaling.
Cost Efficiency	Cost-saving options (spot, reserved instances, savings plans)	Similar options, with hybrid discounts	Generally lower on-demand costs, strong for analytics workloads	GCP offers competitive pricing in analytics, but AWS remains superior for cost efficiency across services.
Availability and Reliability	Broadest global coverage and availability zones	Strong availability, good regional presence	Fewer regions but strong uptime metrics	AWS is top for global availability, essential for application's high uptime requirements.
Performance	High performance across regions with low latency	Comparable with strong enterprise performance	Strong for high-performanc e analytics tasks	Azure performs comparably to AWS, with high-speed

				connections across regions.
Security	Extensive compliance, advanced encryption options	Strong security with enterprise compliance focus	Excellent security, especially in open-source environments	AWS has the broadest compliance coverage, particularly crucial for Blinkit's transaction-heavy environment.
Vendor Flexibility	AWS Outposts for hybrid setups, flexible for multi-cloud	Best for hybrid environments and multi-cloud with Azure Arc	Limited multi-cloud features	Azure ranks first for hybrid and multi-cloud flexibility due to Azure Arc's compatibility across environments.
Monitoring and Automation	Best-in-class with CloudWatch, Lambda, advanced automation	Strong with Azure Monitor, Logic Apps	Strong, with AI-driven insights	AWS offers the most comprehensive monitoring and automation suite, essential for Blinkit's real-time needs.
Support for Multi-Tenancy	Strong multi-tenancy with resource isolation (e.g., AWS IAM)	Good, with Azure AD providing role and resource isolation	Adequate, with basic multi-tenancy and RBAC support	AWS leads with robust resource isolation and multi-tenancy capabilities suited for

				Blinkit's diverse user base.
Compliance and Regulatory Support	Industry-leading compliance coverage	Excellent compliance, with specific focus on enterprise needs	Good compliance, especially for data-focused applications	Azure has extensive certifications as well, positioning it close to AWS for enterprise compliance requirements.
Disaster Recovery and Redundancy	Top-tier disaster recovery with multiple regions	Strong redundancy with geographic failover options	Limited redundancy, fewer regions	AWS's widespread regions and backup options make it the best for disaster recovery and continuity.

[7][8]

3.3 The final selection

AWS is chosen as the preferred provider, given its extensive global infrastructure, reliable performance, and flexibility in cost management. These strengths align well with Blinkit's needs for high availability, scalability, and disaster recovery, making it the best all-around choice. AWS's monitoring tools and broad compliance also make it a strong candidate for the delivery service's operational needs. AWS's dominance in scalability, cost efficiency, availability, and automation tools solidifies its role as the optimal choice, while Azure's flexibility makes it a strong backup option.

3.3.1 The list of services offered by the winner

Scalability

- **AWS Auto Scaling** dynamically adjusts compute resources to handle varying workloads, satisfying **TR1.2** and **TR2.1**.
- **AWS Lambda** enables serverless architectures, reducing idle costs (**TR7.1**) and supporting rapid scaling for specific workloads.

- **Elastic Load Balancing (ELB)** evenly distributes traffic, ensuring efficient load management (**TR2.2**).

Cost Efficiency

- **AWS Cost Explorer** and **AWS Budgets** allow granular cost tracking (**TR1.1**, **TR9.1**).
- **Savings Plans** and **spot instances** minimize idle costs during low-demand periods (**TR1.3**, **TR7.2**).

Availability and Reliability

- **Amazon Route 53** and **AWS Global Accelerator** improve availability and routing for global users (**TR3.1**, **TR3.2**).
- **AWS Elastic Disaster Recovery** ensures critical services are redundant and failover-ready (**TR3.3**, **TR12.1**).

Performance

- **Amazon Aurora** and **Amazon DynamoDB** optimize database performance for high-volume transactions (**TR4.1**, **TR4.2**).
- **Amazon ElastiCache** improves API response times by caching frequently accessed data (**TR13.1**, **TR15.1**).

Security

- **AWS KMS** encrypts data at rest and in transit, ensuring compliance and protection (**TR5.1**, **TR17.2**).
- **AWS WAF** and **AWS Shield** provide protection against cyberattacks (**TR5.2**, **TR18.2**).

Vendor Flexibility

- **AWS Outposts** allows hybrid setups with consistent operational practices (**TR6.1**, **TR6.2**).
- **AWS EKS** supports Kubernetes workloads, promoting multi-cloud compatibility (**TR6.3**).

Monitoring and Automation

- **Amazon CloudWatch** offers real-time monitoring, while **AWS Systems Manager** handles operational automation.**(TR15.2, TR27.2)**
- **AWS Lambda** aids in event-driven automation, supporting error recovery (**TR11.1**, **TR21.1**).

Support for Multi-Tenancy

- **AWS IAM** and **AWS Organizations** manage resource isolation and tenant-specific configurations (**TR29.1**, **TR29.2**).

Compliance and Regulatory Support

- **AWS Artifact** and **AWS Config** simplify compliance audits and align with data privacy regulations (**TR23.1**, **TR23.2**).

Disaster Recovery and Redundancy

- **Amazon S3 Cross-Region Replication** and **AWS Backup** ensure data redundancy across multiple regions (**TR12.1**, **TR12.2**).
- **AWS Elastic Disaster Recovery** supports uninterrupted operations for critical systems (**TR11.1**, **TR3.3**).

This highlights how AWS services comprehensively satisfy the application's business and technical requirements, ensuring optimal performance, cost efficiency, and scalability.

3.3.2. Service Descriptions:

(refined by chatgpt)

1. Amazon CloudWatch

Amazon CloudWatch is a monitoring and observability service that provides real-time insights into your AWS resources and applications.

- **Core Features**:
    - **Metrics Monitoring**: Collects and tracks metrics from AWS resources, on-premises servers, and custom applications, including CPU usage, disk I/O, and API request counts.
    - **Log Management**: Aggregates logs from various services (e.g., EC2, Lambda) to help identify and troubleshoot performance issues.
    - **Dashboards**: Creates customizable visual dashboards for metrics and logs to monitor system health.
    - **Alarms**: Sets up thresholds for metrics and sends notifications via Amazon SNS or triggers automatic actions when thresholds are breached.
    - **Event-driven Automation**: Integrates with AWS Lambda to execute functions based on system events or alarms.
- **Use Cases**:
    - Tracking infrastructure performance in real-time.
    - Automating scaling actions or remediation workflows based on monitored metrics.
    - Enhancing application observability and identifying performance bottlenecks.

2. Elastic Load Balancing (ELB)

Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as EC2 instances, containers, and IP addresses, ensuring high availability and fault tolerance.

- **Core Features**:
    - **Types of Load Balancers**:
        - Application Load Balancer (ALB): Ideal for HTTP and HTTPS traffic; provides advanced routing features like path- and host-based routing.
        - Network Load Balancer (NLB): Designed for ultra-high-performance needs with low latency; operates at Layer 4 (TCP/UDP traffic).
        - Classic Load Balancer (CLB): Legacy solution for EC2 instances, supporting both Layer 4 and Layer 7 routing.
    - **Health Checks**: Continuously monitors the health of registered targets and routes traffic only to healthy instances.
    - **Auto Scaling Integration**: Works with AWS Auto Scaling to add or remove instances based on traffic demand.
    - **TLS Termination**: Decrypts traffic at the load balancer level to reduce overhead on backend servers.
- **Use Cases**:
    - Managing traffic spikes efficiently.
    - Improving fault tolerance and minimizing downtime.
    - Enhancing application performance with intelligent routing.

3. Amazon ElastiCache

Amazon ElastiCache is a fully managed in-memory caching service designed to improve application performance by reducing the load on databases. It supports two engines: **Redis** and **Memcached**.

- **Core Features**:
    - **In-memory Data Storage**: Provides sub-millisecond latency by caching frequently accessed data in memory.
    - **High Availability**: Supports replication and automatic failover to ensure data availability during failures.
    - **Scalability**: Automatically adjusts capacity to handle increasing demand.
    - **Data Persistence**: In Redis, data persistence options allow backups and recovery to ensure durability.
    - **Integration with AWS Services**: Works seamlessly with services like Amazon RDS and DynamoDB to offload query loads.
- **Use Cases**:
    - Caching query results for frequently accessed database data.
    - Session management for user logins.
    - High-performance real-time analytics.

4. AWS Identity and Access Management (IAM)

AWS IAM is a web service that enables secure control over access to AWS resources. It allows you to create and manage AWS users, groups, and permissions.

- **Core Features**:
    - **Access Control**: Manages who (users, groups, roles) has access to specific AWS resources and under what conditions.
    - **Granular Permissions**: Supports fine-grained policies that specify which actions are allowed or denied for users or services.
    - **Temporary Credentials**: Issues short-term security tokens for users or applications through roles or federated access.
    - **Multi-factor Authentication (MFA)**: Adds an extra layer of security for sensitive resources.
    - **Resource-based Policies**: Assigns permissions directly to AWS resources for use cases like S3 bucket-level permissions.
- **Use Cases**:
    - Managing user roles and permissions within multi-tenant architectures.
    - Enforcing least privilege access for security compliance.
    - Supporting federated authentication for enterprise users via SAML or OpenID.

4 The First Design Draft [0%]

For our project, the cloud infrastructure design leverages Amazon Web Services (AWS) to meet Blinkit's business and technical requirements. The proposed architecture emphasizes scalability, cost optimization, high availability, performance, and security to address Blinkit’s needs for a reliable grocery delivery service. Below is the first draft of the design and its essential components.

4.1 The Basic Building Blocks of the Design

The proposed design incorporates several distinct elements to address the project's objectives. Each component is aligned with specific technical requirements (TRs) and business requirements (BRs):

4.1.1 Data Ingestion and Processing

- Amazon Kinesis: Handles real-time ingestion of order data, delivery tracking, and user interactions, ensuring low-latency data pipelines for immediate processing (TR1.2, TR4.1, BR2).
- AWS Lambda: Manages event-driven processing for dynamic order updates and delivery status, reducing idle resource costs and supporting serverless architecture (TR1.3, TR2.3, BR1).

4.1.2 Data Storage

- Amazon S3: Stores historical order and inventory data, enabling efficient archival and retrieval for analytics and auditing (TR8.1, TR8.2).
- Amazon DynamoDB: Maintains a real-time inventory database to support fast, scalable data retrieval (TR4.1, TR4.2, TR14.1).
- Amazon Aurora: Provides a relational database for transaction data requiring complex queries and strong consistency (TR3.2, TR4.1, TR4.2).

4.1.3 Backend Services

- Amazon Elastic Kubernetes Service (EKS): Orchestrates containerized backend services for order management and API integrations (TR6.2, TR6.3, TR30.1, TR30.2).
- Elastic Load Balancing (ELB): Distributes incoming requests to backend services, ensuring efficient load management and high availability during peak demand (TR2.2, TR24.1, TR10.1).

4.1.4 API Management and Integration

- Amazon API Gateway: Serves as a secure entry point for user and partner applications, managing authentication, throttling, and monitoring (TR4.2, TR5.2).

4.1.5 Data Analytics and Visualization

- Amazon QuickSight: Provides insights into user behavior, peak order times, and inventory trends, supporting business intelligence (TR9.1, TR13.2).

4.1.6 Security and Access Management

- AWS Identity and Access Management (IAM): Ensures role-based access control, defining permissions for different user groups like administrators, delivery personnel, and customers (TR5.2, TR19.1).
- AWS Key Management Service (KMS): Encrypts sensitive data such as payment details and delivery addresses to maintain confidentiality (TR5.1, TR17.2).

4.1.7 High Availability and Disaster Recovery

- Amazon Route 53: Provides DNS services for seamless redirection during failover events (TR3.1).
- AWS Elastic Disaster Recovery: Ensures minimal downtime and rapid recovery during failures (TR3.3, TR12.1, TR10.1).

4.1.8 Caching and Performance Optimization

- Amazon ElastiCache (Redis): Reduces database load by caching frequently queried data, enhancing performance during peak times (TR15.1, TR13.1).

4.1.9 Monitoring and Automation

- Amazon CloudWatch: Tracks application performance and triggers automated scaling or alerts during anomalies (TR11.1).
- AWS Systems Manager: Automated patching, configuration management, and operational insights (TR21.1).

4.1.10 Cost Management and Optimization

- AWS Cost Explorer and Budgets: Monitors and forecasts spending, optimizing costs for scaling and resource usage (TR1.1).
- Savings Plans: Minimizes expenses during off-peak hours by leveraging flexible pricing models (TR7.2).

Each building block is strategically selected to address Blinkit's requirements for operational scalability, cost-efficiency, high availability, and security, ensuring a robust and future-proof cloud solution.

[9][10][11]

4.2 Top-Level, Informal Validation of the Design

The proposed design addresses Blinkit's need for a scalable, reliable, and cost-efficient cloud infrastructure. Each component of the architecture is chosen to ensure high availability, support dynamic workloads, optimize costs, and maintain security. Below are arguments validating the effectiveness of the design:

4.2.1 Data Ingestion and Processing

- **Amazon Kinesis** ensures real-time ingestion of high-volume order data, enabling Blinkit to process spikes during peak events such as flash sales. By maintaining low-latency data pipelines, the system ensures rapid updates to inventory and order tracking (**TR2.1, TR4.1, TR14.1**).
- **AWS Lambda**, with its serverless architecture, dynamically processes data without incurring costs during idle periods. This makes it ideal for handling fluctuating workloads efficiently (**TR1.3, TR2.3**).

4.2.2 Data Storage

- **Amazon DynamoDB**, with its low-latency, high-throughput performance, supports Blinkit's real-time inventory management system, ensuring users see accurate stock levels during order placement (**TR4.1, TR4.2**).
- **Amazon S3** offers cost-efficient, scalable storage for archived order data and analytics, reducing operational expenses while enabling seamless retrieval (**TR8.1, TR8.2**).

4.2.3 Backend Services

- **Elastic Load Balancing (ELB)** manages sudden traffic spikes by distributing requests across multiple backend services, ensuring uninterrupted user experiences during peak demand periods (**TR2.2**).
- **Amazon EKS**, with its support for container orchestration, allows Blinkit to deploy and scale backend services independently, improving flexibility for future enhancements (**TR6.3**).

4.2.4 API Management and Integration

- **Amazon API Gateway** facilitates secure and scalable API interactions, ensuring consistent communication between the frontend and backend services. It enforces throttling and authentication, critical for maintaining performance and security during high-demand scenarios (**TR4.2, TR5.2**).

4.2.5 Caching and Performance Optimization

- **Amazon ElastiCache** (Redis) reduces database load by caching frequently accessed data such as user session details and order history, ensuring rapid API responses and enhanced app responsiveness (**TR13.1, TR15.1, TR4.2**).

4.2.6 Availability and Disaster Recovery

- **Amazon Route 53** and **AWS Elastic Disaster Recovery** ensure high availability and quick failover in case of regional outages, maintaining uninterrupted service for critical operations such as order processing and payment gateways (**TR3.1, TR12.1**).

4.2.7 Security and Compliance

- **AWS IAM** enforces strict role-based access control, ensuring that sensitive customer data, such as payment details and delivery addresses, are only accessible to authorized personnel (**TR5.2, TR19.1**).
- **AWS KMS** encrypt sensitive data both at rest and in transit, meeting compliance requirements for payment security and customer data protection (**TR5.1**).

4.2.8 Monitoring and Cost Management

- **Amazon CloudWatch** tracks system performance and triggers alarms for anomalies, ensuring proactive error detection and resolution, which is critical for maintaining operational excellence during high-demand periods (**TR11.1**).
- **AWS Cost Explorer** and **Savings Plans** enable Blinkit to monitor expenses in real time and optimize resource utilization, ensuring cost-efficiency without sacrificing performance (**TR1.1, TR7.2**).

The proposed design meets Blinkit's core requirements by leveraging AWS services for scalability, availability, performance, and security. The integration of services like Kinesis, DynamoDB, and ElastiCache ensures that the infrastructure can handle peak loads, provide

real-time updates, and minimize downtime, all while optimizing costs. While further refinements may be necessary, the current draft provides a strong foundation to address Blinkit's challenges effectively.

[7][8]

4.3 Action items and rough timeline

SKIPPED

5 The second design [0%]

5.1 Use of the Well-Architected framework

(refined using ChatGpt)

The AWS Well-Architected Framework provides a comprehensive approach to designing and evaluating workloads on the cloud, ensuring alignment with best practices. Below is an explanation of how the six pillars of this framework guide the architectural design process, refined with references to the latest AWS materials.

5.1.1 Operational Excellence

This pillar focuses on continuously improving processes to support efficient development and operational workflows. It ensures the delivery of business value through consistent, repeatable operations.

- Perform operations as code to standardize workflows.
- Frequent, small, reversible changes to minimize disruption during updates.
- Learn from failures and refine processes to prevent recurrence.
- Relevance: For Blinkit, operational excellence ensures that frequent updates to inventory tracking, delivery scheduling, and dynamic pricing can be implemented with minimal downtime. Continuous improvement processes ensure the platform evolves with user needs**.**

[9]

5.1.2 Security

The security pillar aims to protect systems, data, and assets by embedding security measures across all layers of the architecture. AWS's shared responsibility model ensures secure infrastructure, while customers focus on securing their workloads.

- Enable traceability for all system actions through monitoring and logging.

- Encrypt sensitive data at rest and in transit using tools like AWS KMS.
- Automate security best practices, such as setting IAM roles and access permissions.
- Relevance: For Blinkit, protecting customer data like payment details and addresses is paramount. Security best practices ensure compliance with regulatory standards while safeguarding data integrity.
  .

5.1.3 Reliability

This pillar emphasizes ensuring workloads can recover from failures, meet demands, and adapt to changes in requirements.

- Automatically recover from failures using fault-tolerant designs.
- Test recovery procedures to ensure rapid failover during outages.
- Scale horizontally to meet demand surges effectively.
- Relevance: Blinkit requires consistent uptime, especially during high-traffic events like festivals. AWS services like ELB and Route 53 ensure reliable traffic routing and failover capabilities.

5.1.4 Performance Efficiency

This pillar focuses on optimizing computing resources to handle varying demands efficiently.

- Use serverless architectures like AWS Lambda to scale dynamically with demand.
- Monitor system performance metrics to identify bottlenecks.
- Experiment frequently to adopt newer, more efficient technologies.
- Relevance: With rapidly fluctuating workloads during peak hours, Blinkit needs optimized resource allocation to maintain fast response times for order placement and delivery tracking.

5.1.5 Cost Optimization

This pillar ensures that systems operate at the lowest cost possible without compromising on performance.

- Adopt a consumption model to pay only for used resources.
- Measure and improve cost efficiency by analyzing billing insights.
- Avoid spending on undifferentiated heavy lifting by using managed AWS services.
- Relevance: Minimizing costs for Blinkit's cloud operations ensures profitability while maintaining customer satisfaction during off-peak hours through scaling and automated cost management.

5.1.6 Sustainability

The sustainability pillar emphasizes reducing the environmental impact of cloud workloads by optimizing resource utilization.

- Maximize resource utilization to reduce waste.
- Use energy-efficient designs and adopt AWS regions that utilize renewable energy.
- Continuously measure and improve workload efficiency.
- Relevance: As Blinkit scales, incorporating sustainability ensures alignment with global environmental goals while reducing operational costs.

[12]

5.2 Discussion of pillars

5.2.1. Operational Excellence

The Operational Excellence pillar of the AWS Well-Architected Framework emphasizes practices that enable effective operations and the continuous improvement of processes to deliver business value. It focuses on how to run workloads, monitor them, and improve their supporting processes.

1. Design Principles:
    - Perform Operations as Code: Automate repetitive tasks, such as deployments and configurations, using tools like AWS CloudFormation or AWS Systems Manager.
    - Make Frequent, Small, Reversible Changes: Use CI/CD pipelines to deploy incremental updates, reducing the risk of large-scale failures.
    - Anticipate Failure: Design systems that can predict and handle failures gracefully, such as enabling health checks with Elastic Load Balancing (ELB).
    - Learn from Operational Failures: Use tools like Amazon CloudWatch Logs to collect and analyze logs for insights, improving future processes.
2. Operational Practices:
    - Prepare: Define workloads and create standards for management, such as operational playbooks or runbooks.
    - Operate: Conduct routine operations like deployments and manage incidents effectively with automated tools.
    - Evolve: Continuously refine and improve processes based on lessons learned from past operations and failures.
3. AWS Services for Operational Excellence:
    - AWS CloudFormation: Automates infrastructure deployment and management.
    - AWS Systems Manager: Centralizes operational data for troubleshooting and automates routine tasks.
    - Amazon CloudWatch: Provides monitoring and observability for system health and application performance.

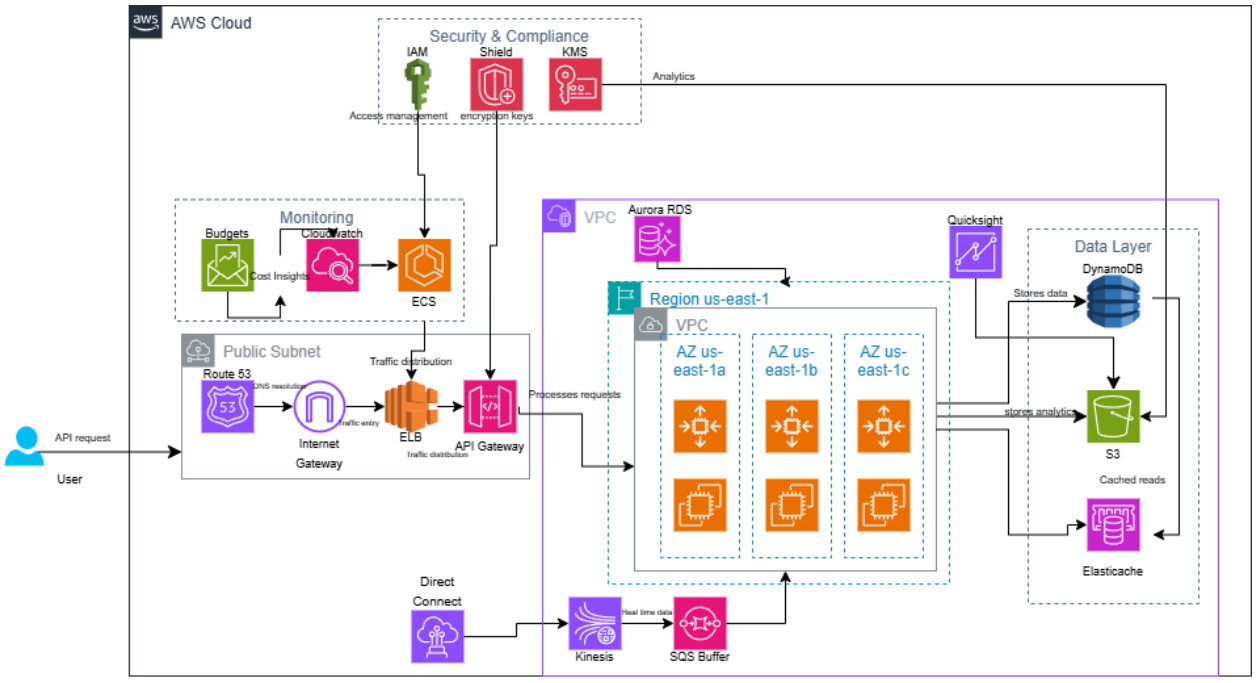
5.2.2 Reliability

The Reliability pillar focuses on ensuring that workloads perform their intended functions consistently and recover quickly from failures. It involves designing systems that can withstand disruptions and remain available under varying conditions.

1. Design Principles:
    - Automatically Recover from Failure: Use automation tools to detect and mitigate failures, such as auto-healing EC2 instances with Auto Scaling.
    - Test Recovery Procedures: Regularly simulate disasters (e.g., by performing chaos engineering) to validate recovery strategies.
    - Scale Horizontally: Increase availability by distributing workloads across multiple nodes or instances. AWS services like Amazon Aurora and ELB support horizontal scaling.
    - Stop Guessing Capacity: Use services like AWS Auto Scaling to dynamically adjust capacity based on real-time demand.
    - Manage Change in Automation: Automate changes and deployments to minimize human error and reduce downtime
      .
2. Reliability Practices:
    - Foundational Requirements: Configure reliable networking, enable Multi-AZ (Availability Zones) for databases, and use disaster recovery mechanisms.
    - Change Management: Monitor the impact of updates and changes on system performance using AWS CloudTrail.
    - Failure Management: Implement backup strategies (e.g., Amazon S3 backups with Cross-Region Replication) and disaster recovery plans.
3. AWS Services for Reliability:
    - Amazon Route 53: Provides reliable DNS routing with health checks and failover options.
    - AWS Auto Scaling: Dynamically adjusts compute resources based on demand, ensuring consistent performance.
    - AWS Elastic Load Balancing (ELB): Balances traffic across instances, ensuring application availability.
    - Amazon RDS (Relational Database Service): Provides Multi-AZ deployments for database reliability.
    - AWS Backup: Automates data backup and restoration processes to protect against data loss.

[12]

5.3 Cloudformation Diagram



→ arrows indicate the data flow

5.4 Validation Of The Design

5.4.1. Data Ingestion and Processing

Components:

- **Amazon Kinesis:** Responsible for ingesting high volumes of real-time data from various city sensors and user devices. Kinesis integrates seamlessly with downstream processing services.
- **Amazon EC2**: Processes ingested data, offering the flexibility to run custom compute operations while ensuring adequate resources are provisioned for peak workloads.

How TRs Are Met:

- **TR2.1 (Rapid Scaling):** Kinesis supports scalable data ingestion during peak traffic, while EC2 instances can scale vertically or horizontally as needed to handle processing demands.
- **TR14.1 (Real-Time Data Propagation):** Kinesis ensures low-latency pipelines, and EC2 enables immediate processing with optimized compute resources.
- **TR1.3 (Idle Cost Optimization):** EC2 allows the option of scaling down or shutting off underutilized instances during off-peak hours, reducing unnecessary costs.

- **TR2.3 (Peak Load Optimization):** EC2 instances can be dynamically allocated using Auto Scaling Groups, ensuring consistent performance during high-load scenarios.

Potential Gaps:

- **TR11.1 (Error Detection)**: While Kinesis provides logs, EC2 requires integration with monitoring tools like CloudWatch for real-time issue detection.
- **TR12.1 (Fault Tolerance)**: EC2 lacks built-in replication, requiring careful configuration of failover and backup mechanisms to match Kinesis' multi-AZ redundancy.

5.4.2. Data Storage

Components:

- **Amazon S3:** Acts as a data lake, storing unstructured data such as IoT sensor logs and user-generated content.
- **Amazon Aurora:** Provides relational database capabilities for transactional data and structured storage needs.

How TRs Are Met:

- **TR8.1 (Cost-Efficient Data Storage)**: S3 optimizes costs by classifying data into storage tiers based on access patterns.
- **TR12.2 (Regional Failover)**: Aurora supports automatic failover across multiple availability zones to maintain uptime during disruptions.
- **TR4.1 (High Transaction Volume Support)**: Aurora ensures high performance for transactional workloads with low-latency access.
- **TR8.2 (Data Archiving)**: S3's lifecycle policies enable automated archiving for older data, reducing operational costs.

Potential Gaps:

- **TR15.2 (Usage-Based Monitoring)**: Dynamic adjustments to storage configurations based on access patterns may require additional monitoring tools.
- **TR17.1 (Retention Compliance)**: Lifecycle policies must be explicitly aligned with regulatory retention requirements.

5.4.3. Network and Security

Components:

- **AWS VPC:** Isolates resources within private subnets for enhanced security.
- **AWS IAM:** Provides fine-grained access control for users and applications.
- **AWS Shield and WAF:** Protect against DDoS attacks and malicious traffic targeting public-facing endpoints.

How TRs Are Met:

- **TR6.1 (Network Segmentation)**: VPC isolates resources, ensuring secure communication and preventing unauthorized access.
- **TR19.1 (RBAC Enforcement)**: IAM implements least-privilege access for all resources and services.
- **TR5.1 (Data Protection)**: Shield and WAF prevent unauthorized data exposure by securing network endpoints.

Potential Gaps:

- **TR21.1 (Automated Issue Tracking)**: Additional CloudWatch alarms may be needed to monitor security threats in real time.

5.4.4. Monitoring and Analytics

Components:

- **Amazon CloudWatch:** Captures logs and metrics for performance monitoring and alerting.
- **AWS QuickSight:** Provides interactive dashboards and visualizations for data-driven decision-making.

How TRs Are Met:

- **TR11.1 (Proactive Issue Detection)**: CloudWatch provides alerts and anomaly detection for early issue identification.
- **TR9.1 (Data Visualization)**: QuickSight enables detailed analytics for resource usage and operational trends.
- **TR13.2 (API Testing)**: CloudWatch helps monitor API performance and logs, ensuring consistent behavior across workloads.

Potential Gaps:

- **TR7.2 (Dynamic Adjustments)**: Autoscaling triggers based on metrics may need fine-tuning to prevent over-provisioning.

5.4.5. Elasticity and Fault Tolerance

Components:

- **Auto Scaling Groups (ASGs):** Ensure ECS workloads dynamically adjust based on demand.
- **Amazon EC2**: Scales up or down to handle unpredictable workloads without requiring manual intervention.

How TRs Are Met:

- **TR2.1 (Elasticity)**: ASGs ensure EC2 workloads are adjusted dynamically to meet fluctuating demand.
- **TR12.1 (Multi-AZ Resilience)**: EC2 instances deployed across multiple availability zones ensure fault tolerance and high availability.
- **TR7.1 (Real-Time Adjustments)**: EC2 Auto Scaling ensures resource usage is optimized by adjusting capacity in real time.

Potential Gaps:

- **TR8.2 (Backup Policies)**: Configuring backups for EC2 requires additional steps compared to managed serverless options.
- **TR14.2 (Cross-Region Resilience)**: Setting up cross-region failover for EC2 instances requires manual configurations with Route 53 health checks.

5.5 Design Principles and Best Practices Used

(refined using chatgpt)

5.5.1 Scalability

- **Design Principle:** Built to scale both horizontally and vertically, ensuring the infrastructure can handle growth in users, devices, and data volumes without compromising performance.
- **Best Practices:**
    - Leveraging auto-scaling for compute resources (e.g., AWS Lambda and EC2 Auto Scaling Groups) to dynamically adjust capacity based on traffic patterns.
    - Using Amazon S3 as a data lake, enabling virtually unlimited storage scalability.
    - Implementing Amazon Aurora for automatic database scaling to handle increasing transactional demands.

5.5.2 High Availability and Fault Tolerance

- **Design Principle:** Ensure system reliability by minimizing single points of failure and providing redundancy across components.
- **Best Practices:**
    - Multi-AZ deployment for critical services like Amazon Aurora and ECS to ensure fault tolerance.
    - Amazon Kinesis with built-in replication to provide fault-tolerant real-time data ingestion.
    - Configuring health checks and traffic routing through AWS Global Accelerator to ensure uptime during failures.
    - Regularly testing disaster recovery mechanisms like cross-region backups.

5.5.3 Security by Design

- **Design Principle:** Embed security controls at every layer of the architecture, minimizing vulnerabilities and ensuring data protection.
- **Best Practices:**
    - VPC segmentation to isolate workloads, restricting sensitive data to private subnets.
    - Implementing least-privilege access control using AWS Identity and Access Management (IAM) roles and policies.
    - Securing APIs with AWS WAF and Shield to protect against malicious traffic and DDoS attacks.
    - Encrypting data at rest (S3 bucket encryption, Aurora encrypted databases) and in transit (SSL/TLS for all endpoints).

5.5.4 Cost Optimization

- **Design Principle:** Optimize resource utilization and avoid overprovisioning to minimize costs while maintaining performance.
- **Best Practices:**
    - Using serverless services like Amazon EC2 and AWS Lambda to ensure compute costs align directly with usage.
    - S3's lifecycle management to automatically transition or delete objects based on defined policies, reducing storage costs.
    - Reserved instances for predictable workloads, combined with spot instances for non-critical tasks.

5.5.5 Performance Efficiency

- **Design Principle:** Maximize performance with well-architected solutions that meet latency and throughput requirements.
- **Best Practices:**
    - Using Amazon CloudFront for content delivery to reduce latency for users globally.
    - Configuring Amazon Kinesis shards to handle high-throughput real-time data ingestion without bottlenecks.
    - Monitoring and optimizing query performance in Amazon Aurora and Athena for fast analytics and transactional processing.

5.5.6 Event-Driven Architecture

- **Design Principle:** Design systems that react dynamically to events, ensuring efficiency and responsiveness.
- **Best Practices:**
    - Use EC2 to process Kinesis streams, ensuring flexibility in handling custom workloads.

    - Trigger custom workflows using event-driven mechanisms such as SQS or SNS for message delivery and task initiation.
    - Automating triggers for workflows based on data events (e.g., S3 object uploads triggering downstream processing)

5.5.7 Observability and Monitoring

- **Design Principle:** Provide comprehensive visibility into system performance and health to ensure proactive issue resolution.
- **Best Practices:**
    - Configuring Amazon CloudWatch for collecting logs, metrics, and setting up alerts on key thresholds.
    - Enabling AWS X-Ray for distributed tracing of microservices to troubleshoot latency or dependency issues.
    - Implementing dashboards using AWS QuickSight to provide actionable insights into real-time system behavior.

5.5.8 Decoupling and Modularity

- **Design Principle:** Ensure components are loosely coupled to enhance maintainability and allow independent scaling or modification.
- **Best Practices:**
    - Using SQS for decoupled messaging between microservices, allowing independent updates or failure isolation.
    - Use Amazon EC2 for processing workloads and Amazon S3 for storage. By decoupling these components, scalability is maintained as each can scale independently to meet specific demands.
    - Modularize EC2 instances for compute tasks and segregate ingestion, processing, and storage into distinct services to allow for independent upgrades or replacements without affecting the overall architecture.

5.5.9 Compliance and Data Governance

- **Design Principle:** Adhere to regulatory requirements and ensure data is handled responsibly.
- **Best Practices:**
    - Using AWS Config to track changes and ensure compliance with predefined rules.
    - Applying retention and lifecycle policies to meet data governance standards (e.g., GDPR, HIPAA).
    - Encrypting all sensitive data with KMS to ensure it complies with industry regulations.

5.5.10 Automation

- **Design Principle:** Automate repeatable tasks to improve efficiency and reduce human error.
- **Best Practices:**
    - Infrastructure as Code (IaC) using AWS CloudFormation or Terraform for consistent, repeatable deployments.
    - Automating data workflows using Step Functions to coordinate tasks across services.
    - Continuous monitoring and patching using AWS Systems Manager for maintaining security and performance.

5.6 Trade Offs Revisited

5.6.1 Tradeoff 1: Real-Time Data Processing vs. Cost Efficiency

Objective/Requirement:
Enable real-time processing of high-volume data streams to meet the responsiveness needs of smart city applications (e.g., traffic control, emergency response). Latency must be minimized to ensure immediate action based on incoming sensor data.

Options Evaluated:

1. **Real-Time Processing (Selected Option):**
    i. Utilize Amazon Kinesis for data ingestion and Amazon EC2 for real-time computation.
    ii. EC2 provides full control over compute resources, allowing for tailored optimizations based on workload demands.
    iii. Instances can be provisioned and scaled dynamically using Auto Scaling Groups to handle fluctuations in data volume.
2. **Batch Processing (Alternative):**
    i. Store data temporarily in Amazon S3, then process periodically using Amazon EC2 or AWS Glue.
    ii. Reduces infrastructure costs by avoiding continuous processing but introduces latency, delaying insights or actions by minutes to hours.

Reasoning for the Decision:
Real-time processing was chosen to meet the critical latency requirements of smart city services. EC2 offers flexibility in configuring and optimizing compute environments to meet specific processing demands. Delayed insights from batch processing could result in:

- Traffic congestion due to outdated data.
- Ineffective emergency responses, where decisions must be made within seconds.

Outcome of the Tradeoff:

1. **Benefits**:
    - Full control over the compute environment for custom optimizations.
    - Ensures real-time responsiveness, directly fulfilling TRs related to latency and scalability.
    - Allows for dynamic scaling, ensuring high availability during peak traffic periods.
2. **Drawbacks**:
    - Increased operational complexity due to manual provisioning and maintenance of EC2 instances.
    - Higher costs for always-on infrastructure during periods of continuous demand.

5.6.2 Tradeoff 2: Security and Isolation vs. Operational Complexity

Objective/Requirement:
Ensure robust security by isolating sensitive data and workloads while meeting compliance requirements. The architecture must minimize risks associated with breaches while adhering to industry best practices.

Options Evaluated:

1. **Workload Isolation (Selected Option):**
    i. Implement VPC segmentation with private subnets for each service (e.g., data ingestion, storage, analytics).
    ii. Enforce strict traffic rules using security groups and NAT gateways to prevent unauthorized access.
2. **Simplified Networking (Alternative):**
    i. Deploy all services within a shared, simplified network configuration.
    ii. Reduces operational complexity but exposes the system to higher risks (e.g., lateral movement during breaches).

Reasoning for the Decision:
Workload isolation is essential because **security is non-negotiable** for a system handling sensitive data. The potential risks of a shared network environment include:

- Unauthorized access to critical systems.
- Difficulty achieving compliance with regulatory standards (e.g., GDPR, CCPA).

While the selected approach introduces complexity in managing multiple VPCs and traffic rules, it aligns with the system's goal of ensuring trust and resilience.

Outcome of the Tradeoff:

1. **Benefits:**
    - Reduces attack surface and protects sensitive data.
    - Simplifies compliance with security and privacy regulations.

    - Enhances trust among users and stakeholders.
2. **Drawbacks:**
    - Increased complexity in network design, deployment, and maintenance.
    - Requires skilled personnel for setup and troubleshooting.

5.7 Discussion of an Alternate Design

5.7.1 Alternate Design: Using AWS Lambda Instead of Amazon EC2

An alternate design for Blinkit's architecture could involve replacing Amazon EC2 instances with AWS Lambda as the primary compute resource for processing application logic. Lambda operates on a serverless model, offering event-driven execution and automatic scaling.

5.7.2 Advantages of Lambda-Based Design

1. **Event-Driven Scalability**:
   Lambda's event-driven architecture automatically scales up to handle high traffic during peak events and scales down to zero during idle periods, reducing waste.
2. **Cost Efficiency**:
   Lambda follows a pay-per-execution model, where charges are incurred only for the time and resources consumed during function execution. This is ideal for Blinkit's sporadic workload spikes, such as flash sales or festivals.
3. **Simplified Management**:
   Unlike EC2, Lambda abstracts infrastructure management, eliminating the need to handle server provisioning, OS patching, and scaling configurations.
4. **Integration with AWS Ecosystem**:
   Lambda integrates seamlessly with other AWS services, such as Amazon S3, Kinesis, DynamoDB, and API Gateway, which are core to Blinkit's architecture.
5. **High Agility for Stateless Workloads**:
   Lambda is inherently stateless, making it suitable for Blinkit's transient workloads like order processing and real-time inventory updates.
6. **Quick Deployment**:
   Deployment of new features or updates is faster with Lambda, as individual functions can be updated independently without affecting the entire architecture.

5.7.3 Why EC2 Was Chosen Over Lambda

1. **Persistent Compute Resources:** Blinkit's architecture requires persistent compute environments for long-running processes and specific workloads that exceed Lambda's execution time limit (15 minutes). EC2 provides the flexibility to run stateful applications and extended computations without restrictions.
2. **Custom Configurations:** EC2 allows full control over the underlying infrastructure, including OS-level configurations, custom libraries, and network setups. This is essential for workloads requiring advanced customizations.

3. **Broader Workload Support:** Some workloads, such as resource-intensive processing or tightly coupled applications, perform better on EC2 instances compared to Lambda due to resource and runtime limitations.
4. **Cost Predictability for Steady Workloads:** EC2 offers cost-effective options such as Reserved Instances or Spot Instances, which are beneficial for predictable, continuous workloads. Lambda's per-request model can be costlier for consistently high transaction volumes.
5. **Complex Dependency Management:** Blinkit's backend requires certain dependencies that Lambda may not support natively or would require additional setup, whereas EC2 offers flexibility to handle such cases seamlessly.

While **AWS Lambda** offers advantages like cost efficiency, scalability, and simplified management, its limitations in handling long-running, stateful, or resource-intensive workloads make **Amazon EC2** a better fit for Blinkit's current architecture. EC2 provides the control, flexibility, and persistence necessary for Blinkit's real-time data processing and backend services, ensuring high performance and reliability. However, as Blinkit evolves, revisiting a Lambda-based approach for specific transient workloads may further optimize costs and reduce operational overhead.

6 Kubernetes Experimentation

6.1 Experiment Design for Validating Autoscaling and Load Balancing in EKS

This experiment evaluates the autoscaling and load-balancing functionality of an Amazon Elastic Kubernetes Service (EKS) cluster integrated with an Auto Scaling Group (ASG) and Horizontal Pod Autoscaler (HPA). The objective is to validate the system's ability to dynamically adjust resources (TR 2.1) and distribute traffic efficiently (TR 2.2) during fluctuating workloads.

Environment Configuration:

1. **EKS Cluster**:
    - **Autoscaling Group (ASG)**:
        - Minimum nodes: 1
        - Maximum nodes: 3
        - Scale-up threshold: 35% average CPU utilization per node.
    - **Horizontal Pod Autoscaler (HPA)**:
        - Minimum pods: 1
        - Maximum pods: 10
        - Scale-up threshold: 20% average CPU utilization per pod.
2. **Load Balancer**:
    - AWS Application Load Balancer (ALB) configured to dynamically route traffic across all active pods.
3. **Load Generation**:

- **Locust** simulates up to 1000 users with a ramp-up rate of 10 users per second, providing a realistic and gradual increase in system demand.

Procedure:

1. **Initial Setup**:
   - Launch the EKS cluster with one EC2 node and one active pod.
   - Verify ALB is configured to route traffic to all available pods.
2. **Load Simulation**:
   - Use Locust to generate load, gradually ramping up to 1000 users.
   - Monitor system behavior as:
     - Pod CPU utilization exceeds 20%, triggering HPA to add pods.
     - Node CPU utilization exceeds 35%, causing ASG to provision additional nodes.
   - Maintain peak load to test system stability and responsiveness.
3. **Scale-Down Observation**:
   - Gradually reduce the workload and monitor:
     - HPA terminates pods as utilization decreases.
     - ASG decommissioning idle nodes to optimize resource usage.

Expected Outputs:

1. **Autoscaling (TR 2.1)**:
   - Pods dynamically scale up when CPU utilization exceeds 20%, up to the 10-pod limit.
   - Nodes are added to the cluster when node utilization exceeds 35%, ensuring sufficient capacity for scaled pods.
   - Both pods and nodes scale down efficiently during load reduction.
2. **Load Balancing (TR 2.2)**:
   - ALB evenly distributes traffic across all active pods and dynamically includes new pods and nodes during scaling events.
   - No significant traffic bottlenecks occur during scaling or load transitions.
3. **Key Metrics**:
   - **Autoscaling Efficiency**: HPA and ASG adjust resources dynamically with minimal latency.
   - **Load Balancing Accuracy**: ALB ensures uniform traffic distribution.
   - **System Responsiveness**: Scaling mechanisms react promptly to workload changes.
   - **System Stability**: Consistent application performance is maintained during scaling events.

[13][20]

6.2 Workload Generation

Locust will be employed as the primary load generation tool to simulate user traffic for validating autoscaling (TR 2.1) and load balancing (TR 2.2) in the EKS cluster. The following steps detail how Locust will be configured and utilized to create the necessary inputs:

1. Locust Configuration

- **User Load**:
    - Peak concurrent users: **1000**
    - Ramp-up rate: **10 users per second**
    - Test duration: Long enough to reach peak load, maintain it, and observe scale-up/down events.
- **Behavior Definition**:
    - Create a Python Locustfile (locustfile.py) to define user behaviors:
        - **User Tasks**: Simulate requests to the application endpoints exposed by the ALB.
        - Example: Continuous HTTP GET requests to a specific endpoint (process/7) with 1.5 s delays to mimic real-world usage patterns.
    - Configure response time thresholds and metrics to monitor system performance.

2. Generating Traffic

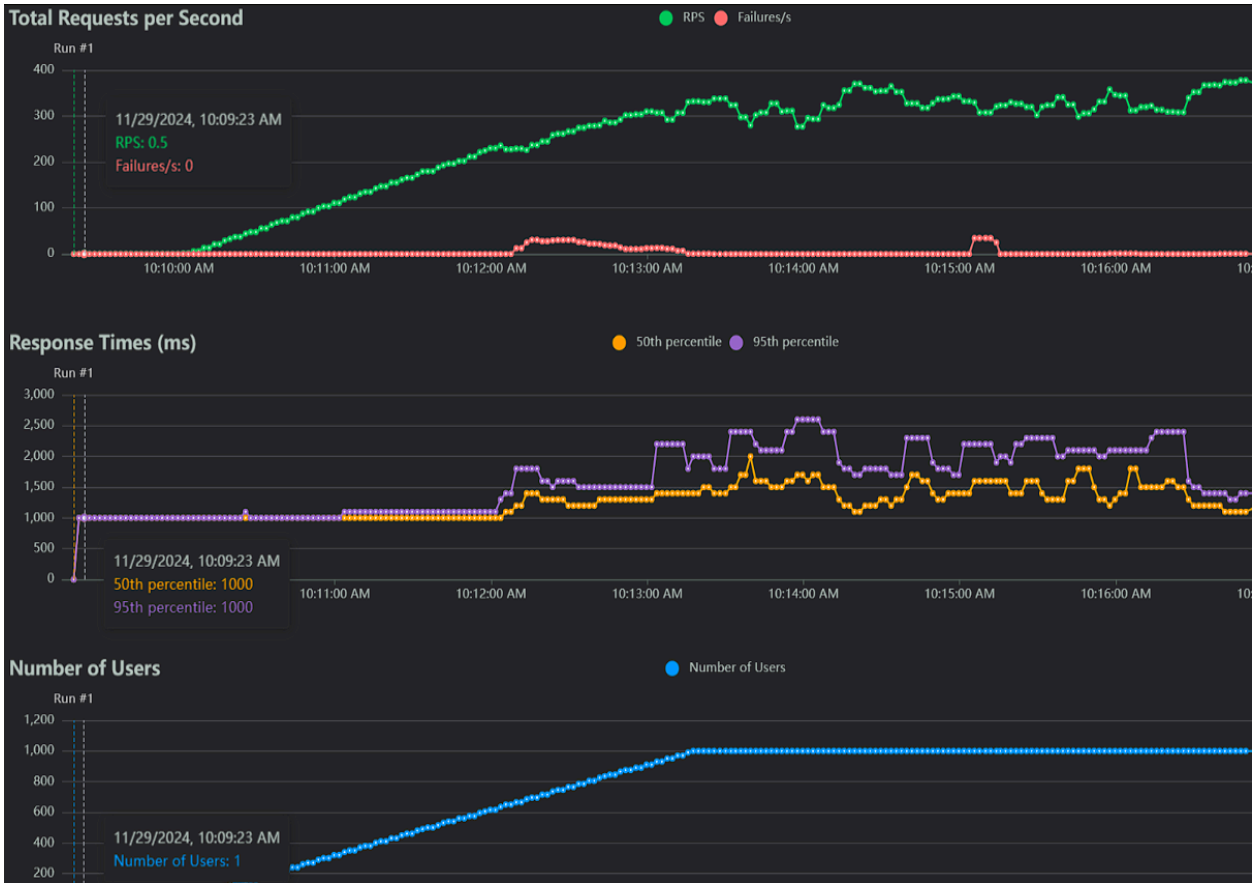
- Deploy Locust in distributed mode:
    - **Master Node**: Orchestrates the load test and aggregates metrics.
    - **Worker Nodes**: Generate user traffic, scaling horizontally as needed for large workloads.
- **Execution**:
    - Launch Locust with parameters for user count, spawn rate, and target application URL (http://a4756aedc281e4268bc58b3407fe6849-622863999.us-east-1.elb.amazonaws.com).
    - Gradually increase traffic from 0 to 1000 users to simulate a realistic traffic ramp-up.

3. Observing Autoscaling and Load Balancing

- Monitor how the system reacts as Locust increases the simulated workload:
    - **Autoscaling**:
        - Check for pod creation when CPU utilization exceeds 20%.
        - Observe new nodes being added when node utilization exceeds 35%.
    - **Load Balancing**:

- Verify even traffic distribution across pods using Locust's request statistics and ALB metrics.

6.2.1 Main Graph:



6.2.2 Timestamp Highlights

1. **10:10 AM**:
   - RPS rises steadily, pods are autoscaled to handle increasing traffic.
   - Response times remain low (<500ms).
2. **10:12 AM**:
   - RPS peaks at **259 requests/second**.
   - Failures begin to rise (**29.6 failures/s**), response times spike.
   - Nodes and pods show uneven utilization.
3. **Post-10:12 AM**:
   - RPS stabilizes, failure rates diminish.
   - Autoscaling completes (10 pods active),load balancing evens out.
   - More Nodes are spawned and pods are evenly distributed.
   - Backup nodes are spawned and kept for further spike.

6.3 Analysis of the Results

The conducted experiment successfully satisfies the **TR2.1** and **TR2.2** criteria by leveraging Kubernetes' autoscaling mechanisms and demonstrating the platform's ability to handle dynamic workloads in a distributed system. Specifically, the Horizontal Pod Autoscaler (HPA) efficiently scaled the application to accommodate increasing traffic, maintaining system stability and performance during peak load scenarios.

This experiment highlights the importance of scalability in cloud-native architectures by ensuring resource utilization remains within optimal thresholds and preventing system failures under high demand. Furthermore, the use of cloud-based load balancing mechanisms ensures workload distribution across nodes, aligning with the foundational requirements of TR2.1 and TR2.2. These results affirm the system's ability to deliver reliable, adaptive, and performant services, which are essential for modern distributed environments.

6.3.1 Autoscaling:

```
PS C:\TVT\Cloud\Project_tryout> kubectl top pods
NAME                         CPU(cores)   MEMORY(bytes)
flask-app-59c86bcd4c-bdccg   1m           22Mi
```

```
PS C:\TVT\Cloud\Project_tryout> kubectl top pods
NAME                         CPU(cores)   MEMORY(bytes)
flask-app-59c86bcd4c-49gb8   65m          24Mi
flask-app-59c86bcd4c-4wjms   65m          28Mi
flask-app-59c86bcd4c-9jszx   69m          24Mi
flask-app-59c86bcd4c-bdccg   77m          24Mi
flask-app-59c86bcd4c-fg2lw   77m          27Mi
flask-app-59c86bcd4c-kptvw   65m          28Mi
flask-app-59c86bcd4c-mpd8x   69m          24Mi
flask-app-59c86bcd4c-ndzsb   61m          27Mi
flask-app-59c86bcd4c-tgjt4   70m          24Mi
flask-app-59c86bcd4c-wgs9x   69m          27Mi
```

Initial Pods:

- Early in the traffic generation period, the deployment starts with **1 pod**, as shown in the initial kubectl top pods output.
- As the load increases, the Horizontal Pod Autoscaler (HPA) observes CPU utilization surpassing the **10% target**, triggering scale-up.

Scaling Up:

- The kubectl top pods output shows an increase in pod count to **10 pods**. Each pod utilizes **~65-77m CPU** and **~27Mi memory**.
- Autoscaling timing:
    - Pods scaled incrementally to match the demand.

- By **10:12 AM**, autoscaling appears complete, as the number of pods stabilizes.

6.3.2 Load balancing:

```
PS C:\IVT\Cloud\Project_tryout> kubectl top nodes
NAME                            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-31-37-130.ec2.internal   691m         35%    734Mi           22%
PS C:\IVT\Cloud\Project_tryout> kubectl top nodes
NAME                            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-31-37-130.ec2.internal   736m         38%    737Mi           22%
ip-172-31-29-202.ec2.internal   <unknown>                           <unknown>
            <unknown>
PS C:\IVT\Cloud\Project_tryout> kubectl top nodes
NAME                            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-31-29-202.ec2.internal   429m         22%    607Mi           18%
ip-172-31-37-130.ec2.internal   396m         20%    735Mi           22%
ip-172-31-92-222.ec2.internal   <unknown>                           <unknown>
            <unknown>
PS C:\IVT\Cloud\Project_tryout> kubectl top nodes
NAME                            CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-31-29-202.ec2.internal   438m         22%    607Mi           18%
ip-172-31-37-130.ec2.internal   378m         19%    737Mi           22%
ip-172-31-92-222.ec2.internal   21m          1%     443Mi           13%
```

Initial Node Utilization:

- At the start of the traffic simulation, the system relied on a single node, which handled all requests.
- As the load increased, the CPU utilization of this node reached **35%**, triggering the Kubernetes Cluster Autoscaler to provision a second node to distribute the load.

Node Expansion and Load Distribution:

- After the second node was spawned, the load balancer (AWS ELB) redistributed the traffic, resulting in an even distribution across the two active nodes.
- This balancing helped stabilize resource utilization across the cluster, ensuring each node handled approximately 20-22% of the total CPU load.

Third Node as a Backup:

- To prepare for future load spikes, the autoscaler provisioned a **third node**. However, this node remained underutilized (idle or lightly loaded), serving as a buffer for sudden traffic surges.

Impact on Performance:

- The introduction of the second node alleviated pressure from the first node, significantly reducing the risk of bottlenecks or resource contention.
- The third node ensured the system had enough capacity to handle any unexpected traffic increases, maintaining system stability.

[14]

7 Ansible playbooks [0%]

(Skipped)

8 Demonstration [0%]

(Skipped)

9 Comparisons [0%]

(summarized using chatGPT)

9.1 Facebook's Katran vs Round-Robin Load Balancing

Aspect	**Facebook's Katran**	**Round-Robin Load Balancing**
Overview	Katran is a modern, scalable Layer 4 load balancer built using eBPF and focused on efficiency.	Round-Robin distributes incoming requests sequentially across available servers.
Performance	High-performance with minimal overhead; handles millions of concurrent connections.	Simple and effective for smaller-scale deployments but struggles with uneven loads.
Scalability	Designed for hyperscale data centers with seamless horizontal scaling.	Scalability depends on the backend; requires additional effort to scale.
Traffic Distribution	Uses flow hashing to optimize server utilization and prevent overloading specific nodes.	Distributes equally without considering backend performance, leading to

		potential bottlenecks.
Complexity	Requires knowledge of eBPF and Linux kernel for deployment and maintenance.	Easy to set up and manage due to its simplicity.

Best Use Cases for Katran:

1. High-Traffic Environments: Katran is ideal for hyperscale data centers handling millions of concurrent connections, such as social media platforms, video streaming services, or large-scale e-commerce platforms like Facebook or Amazon.
2. Dynamic Backend Loads: Katran's flow hashing mechanism distributes requests based on server utilization, making it suitable for environments with fluctuating backend performance.
3. Need for Efficiency: Organizations aiming for minimal overhead and high scalability, leveraging Linux eBPF, will benefit from Katran's design.

Best Use Cases for Round-Robin:

1. Small-Scale Deployments: Round-Robin is simple to implement and is a good fit for applications with a small number of backend servers and predictable loads, such as startups or internal tools.
2. Static Workloads: Suitable for scenarios where all backend servers have equal performance capabilities, reducing the risk of bottlenecks.
3. Ease of Use: Round-Robin's simplicity makes it ideal for teams with limited expertise in advanced load balancing techniques.

9.2 Amazon Elastic Kubernetes Service (EKS) vs. OpenShift

Amazon Elastic Kubernetes Service (EKS) and OpenShift are two popular container orchestration platforms. While both support Kubernetes, they differ significantly in features, deployment models, and use cases. Below is a comparison highlighting their distinct characteristics.

Metric	Amazon EKS	OpenShift
Kubernetes Integration	Managed Kubernetes service directly aligned with Kubernetes open-source standards.	Built on Kubernetes but includes proprietary extensions and features.
Ease of Use	Requires configuration for setup and scaling, leveraging	Simplifies Kubernetes setup with integrated developer

	AWS-native tools.	tools and dashboards.
Deployment	Runs natively on AWS, tightly integrated with other AWS services like IAM, CloudWatch, and VPCs.	Supports multi-cloud, hybrid, and on-premise deployments (e.g., OpenShift Dedicated, OpenShift Container Platform).
Flexibility	Offers granular control for Kubernetes management. Users manage networking, storage, and IAM settings.	Provides a pre-integrated solution with built-in CI/CD pipelines, monitoring, and security features.
Scaling	Supports Kubernetes-native autoscaling via Horizontal Pod Autoscaler (HPA) and Cluster Autoscaler.	Provides simplified scaling with tools like OpenShift Cluster Autoscaler and automated image builds.
Security	Relies on AWS-native security services like IAM, KMS, and WAF.	Includes built-in Role-Based Access Control (RBAC), integrated security scanning, and compliance tools like Red Hat Advanced Cluster Security.
Cost	Pay-as-you-go model, only for the EKS control plane and AWS infrastructure usage.	Subscription-based pricing; higher cost due to proprietary features and enterprise support.
Customization	Fully Kubernetes-compliant; users can customize deployments freely.	Some proprietary extensions may limit compatibility with vanilla Kubernetes tools.
Ecosystem	Leverages AWS's extensive ecosystem for networking, storage, and monitoring.	Offers an ecosystem with Red Hat tools, integrating seamlessly with Red Hat Enterprise Linux (RHEL).
Use Cases	Ideal for AWS-native workloads and those seeking cost-efficient, flexible Kubernetes deployments.	Best suited for enterprises needing multi-cloud or on-prem solutions with strong governance and built-in developer tools.

When to choose each service :
EKS:

- AWS-Native Workloads: If your application heavily relies on AWS services like S3, Lambda, or DynamoDB, EKS offers seamless integration.

- Cost Optimization: For businesses that prefer a pay-as-you-go model and need flexibility in scaling.
- Custom Kubernetes Workloads: For teams familiar with Kubernetes who require full control over configurations.

OpenShift:

- Hybrid and Multi-Cloud: For organizations deploying across different clouds or needing on-premise solutions.
- Enterprise Governance: When built-in security, compliance, and CI/CD capabilities are critical.
- Developer Productivity: With tools like OpenShift's integrated console and pipelines, development teams can accelerate workflows.

[15]

10 Conclusion [0%]

(skipped)

11 References

[1] Amazon Web Services. (n.d.). AWS Cloud Products. AWS. https://aws.amazon.com/products
[2] Amazon Web Services. (n.d.). Real-time Data Analytics with AWS. AWS. https://aws.amazon.com/solutions/real-time-analytics/
[3] Amazon Web Services. (n.d.). High Availability on AWS. AWS. https://aws.amazon.com/high-availability/
[4] Amazon Web Services. (n.d.). AWS Auto Scaling. AWS. https://aws.amazon.com/autoscaling/
[5] Amazon Web Services. (n.d.). AWS Elastic Load Balancing. AWS. https://aws.amazon.com/elasticloadbalancing/
[6] Amazon Web Services. (n.d.). Amazon Aurora Features. AWS. https://aws.amazon.com/rds/aurora/
[7] Gartner. (2024). Cloud Service Provider Comparisons. https://www.gartner.com/en
[8] Amazon Web Services. (n.d.). AWS Cost Explorer. AWS. https://aws.amazon.com/aws-cost-management/aws-cost-explorer/
[9] Amazon Web Services. (n.d.). Amazon Kinesis. AWS. https://aws.amazon.com/kinesis/
[10] Amazon Web Services. (n.d.). Amazon DynamoDB. AWS. https://aws.amazon.com/dynamodb/

[11] Amazon Web Services. (n.d.). Amazon S3. AWS. https://aws.amazon.com/s3/
[12] Amazon Web Services. (n.d.). AWS Well-Architected Framework. AWS. https://aws.amazon.com/architecture/well-architected/
[13] Amazon Web Services. (n.d.). Amazon Elastic Kubernetes Service (EKS). AWS. https://aws.amazon.com/eks/
[14] Amazon Web Services. (n.d.). Cluster Autoscaler for Kubernetes. AWS. https://github.com/kubernetes/autoscaler
[15] Red Hat. (2024). OpenShift Features and Capabilities. https://www.redhat.com/en/openshift
[16] ChatGPT
[17] Claude
[18] Project by Viraj Sanap and Tejas Prabhu (CSC547-Fall2023)
[19] YV and YP, “ECE547/CSC547 class notes”.
[20] Locust.io experiments — running in Kubernetes https://medium.com/locust-io-experiments/locust-io-experiments-running-in-kubernetes-95447571a550