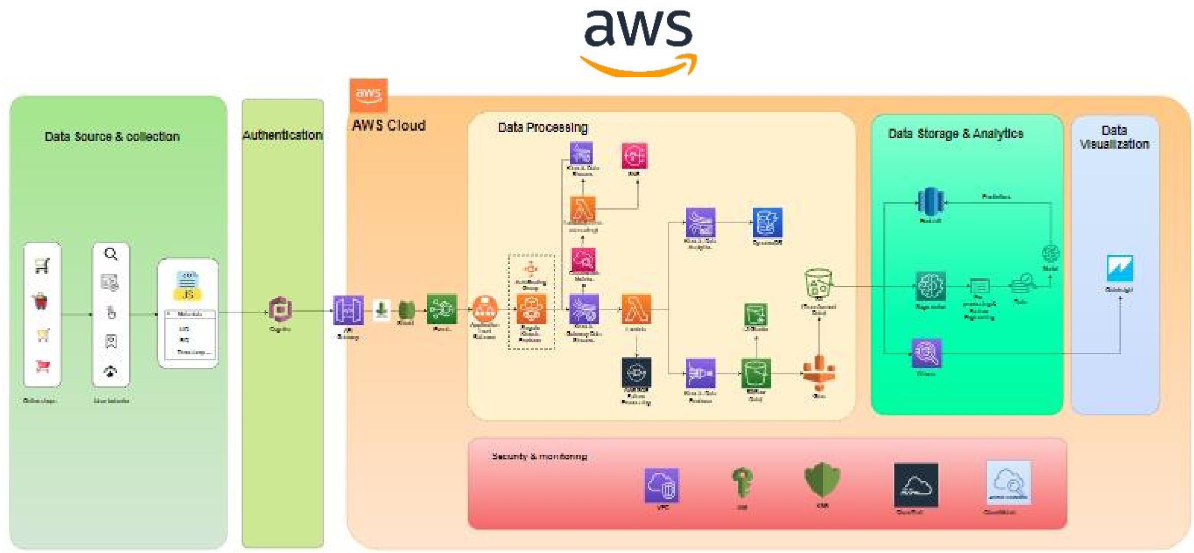
E-commerce Behavioral AWS Architecture Analysis

Task link: Shopware user behavior task

Architecture link: User behavior AWS data pipeline architecture

CICD workflow: user behavior datapipeline CICD

Github link: ecom-aws-infrastructure



Author: Shweta Shivamogga Dattatri

Date: 23.08.2023

Version: 1.0

Client: Shopware

Table of Contents

Introduction:

As e-commerce platforms evolve, there's a burgeoning demand to harness the underlying currents of user interactions. These interactions, subtle yet significant, offer a window into the consumer's mindset, preferences, and decision-making process.

In our project, **E-commerce Behavioral AWS Architecture**, we aim to access this vast pool of user dynamics. We've created a sophisticated pipeline that methodically records, processes, and deciphers these user interactions across a variety of online purchasing platforms. This pipeline is based on the strong ecosystem of AWS.

The vision has several facets:

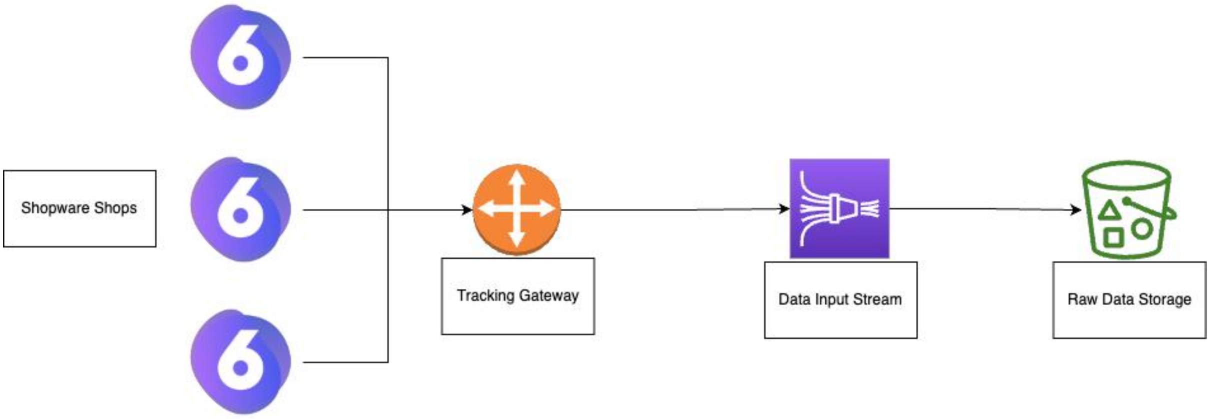
- Equip e-commerce platforms with deep, data-driven user insights.
- Personalize and elevate the shopping experience based on genuine user interactions.
- Guide strategic marketing initiatives with empirical data, promoting informed decision-making.

This document serves as a compass, guiding readers through the labyrinth of our architectural design, pinpointing challenges, and shining light on the optimal pathways we recommend.

Task 1:

Architecture and services on AWS

Scenario: We are tracking data regarding onsite user behavior in thousands of online shops using our product. As a Data Engineer, you must decide which services to use and how those services fit into our target architecture. We have already built and implemented a so-called tracking gateway connected with an AWS Kinesis Firehose service to persist incoming data stream from those online shops into a S3 Bucket.



Question1:

What are the potential challenges and risks of using this architecture and services in the given scenario? Hint: Traffic will be very high,
and it will increase exponentially.

Solution:

Challenges & Risks in Current Architecture:

1) **Scalability:**

Challenge: Kinesis Firehose's default limits might be surpassed with growing user data.
Example: On high-traffic days like Black Friday, the user activity could exceed Firehose's shard limits.

2) **Data Ingestion Delays:**

Challenge: Occasional latency from high traffic may delay data delivery to S3.
Example: Data events from sales spikes might experience a 30-minute delay.

3) **Data Security & Privacy:**

Challenge: User behavior data may contain sensitive or personally identifiable information (PII), which poses security and privacy risks.
Example: User mistakenly inputs email in a shop's search bar.

4) **Cost Management:**

Challenge: Rising costs with increased data traffic.

Example: A sudden user activity surge leads to a 5x cost increase.

5) Integration with Other Systems:
Challenge: Difficulty in integrating user data with external analytics and CRM systems.
Example: Client wishes to reflect user behavior data in both Google Analytics and Salesforce.

6) Data loss:
Challenge: Risk of data loss in processing, transfer, and storage.
Example: Malfunction in Lambda leads to unprocessed events.

7) Latency:
Challenge: Potential for processing delays causing data flow issues.
Example: Buffering delays in Kinesis Firehose slow down data transfer to S3.

8) API Gateway Limits:
Challenge: Navigating the built-in constraints of API Gateway.
Example: Traffic surges might exceed API Gateway's rate limits.

9) Data Integrity:
Challenge: Maintaining data accuracy and validation.
Example: Potential injection of corrupted data from online shops.

10) Security:
Challenge: Safeguarding sensitive data and controlling data ingestion.
Example: Malicious injections through API Gateway or unauthorized S3 access.

11) Data Retention and Backup:
Challenge: Ensuring proper data retention and availability for recovery.
Example: Accidental deletion of crucial data from S3.

Question2:

What architecture or services do you recommend for further processing the data to check data quality and build a feature store used for machine learning? Please visualize your solution and explain the advantages and disadvantages.

Solution:

Recommended Architecture: User behavior data pipeline architecture

Infrastructure Overview:

1.Networking & Regions:

- VPC (Virtual Private Cloud): Provides an isolated cloud section to launch resources.
- Subnets: Designated as public or private, they determine resource accessibility.
- Regions & Availability Zones: Define geographic areas for distributing resources to ensure availability.

- NAT Gateway: Manages outbound traffic for private subnets, blocking unsolicited inbound traffic.
- VPC Endpoints: Allows secure VPC connections to AWS services without public internet traversal.
- VPC Peering: Facilitates direct network communication between different VPCs.

2. Data Security:

- Encryption:

  In Transit: Uses TLS/SSL protocols for data movement.

  At Rest: Encrypts stored data across AWS services.

- Security Groups & NACLs: Virtual firewalls for instance and subnet-level traffic.
- IAM Policies: Manage access to AWS resources.
- WAF & Shield with API Gateway: Defend against malicious requests.
- AWS Cognito: Manages user identity and authentication.

3. **Data Ingestion:**

- API Gateway: Serves as the secured entry point for user behavior data.
- AWS Cognito: Facilitates user interactions through authentication and authorization.

4. **Data Processing:**

- Lambda: Provides real-time data processing.
- Kinesis Data Streams & Firehose: Manage, buffer, and transport streaming data.
- AWS Glue: Handles ETL tasks and data transformation.

5. **Data Storage:**

- S3 Buckets: Offers scalable and durable storage for raw and processed data.
- Redshift: Functions as a data warehouse for structured data analytics.

6. **Machine Learning & Analytics:**

- SageMaker: Trains and predicts using machine learning models.
- Athena & QuickSight: Enable data querying, visualization, and reporting.

7. **Monitoring & Management:**

- CloudWatch: Monitors AWS resources, performance, and logs.
- AWS Budgets & Cost Explorer: Assists in cost management, budgeting, and usage pattern insights.

This Architecture addresses the challenges previously mentioned:

1. Scalability: AWS Cognito, Kinesis Data Analytics, and DynamoDB inherently support scaling. Configurations like read/write capacity of DynamoDB are set appropriately.

2. Data Ingestion Delays: Lambda, Kinesis Data Streams, and Firehose are optimized for real-time processing. Proper buffering settings in Kinesis can mitigate delays.

3. Data Security & Privacy: AWS Cognito ensures user authentication. Using AWS WAF with API Gateway can safeguard against malicious requests. Ensure data at rest (in S3, DynamoDB, and Redshift) and in transit is encrypted.

4. Cost Management: By using AWS Glue for ETL and then storing transformed data in S3, we can avoid continuously running more expensive database operations. Make sure to set lifecycle policies on S3 and monitor AWS service usage.

5. Integration with Other Systems: Lambda and AWS Glue provide ample flexibility for integrating with various systems and transforming data as needed.

6. Data Integrity: By using AWS Glue, can transform and clean the data before saving it in the transformed S3 bucket.

7. Latency: Kinesis is designed for real-time data flow, minimizing latency.

8. API Gateway Limits: We might still need to watch for API Gateway limits, especially with high traffic spikes. Consider caching and throttling.

9. Data Integrity: Lambda ensures real-time validation before data proceeds further down the pipeline.

10. Security: AWS WAF with API Gateway, encryption in S3, and IAM policies handle most security concerns. Ensure least-privilege principles in IAM.

11. Data Retention and Backup: S3's versioning and potential cross-region replication ensure data retention and backup.

Additional considerations:

Monitoring and Alarms: Implement CloudWatch metrics, alarms, and logging across services to ensure to get alert on any issues in the architecture.

Resilience: Consider multi-AZ deployments for DynamoDB, Redshift to ensure high availability.

Advantages:

1.Scalability: This architecture is highly scalable. Kinesis Data Streams can handle a massive influx of data, and Fargate scales the producer application automatically.

2.Real-time Processing: Kinesis Data Analytics allows for immediate insights from the streaming data.

3.Flexibility: By using Lambda for transformations, you have the power and flexibility to process and transform data as it streams in, making downstream processing more straightforward.

4.Durability & Availability: Services like S3, DynamoDB, and Redshift are designed for high durability and availability.

5.Integrated Analysis: With tools like Redshift, Athena, and SageMaker directly connected to the data, it allows for a wide range of analytics, from SQL queries to machine learning.

6.Visualization: QuickSight integration provides a ready solution for visualizing the data insights without much additional setup.

7.Security: With VPC, IAM, and KMS, the data is secure, and access is controlled, ensuring only authorized access.

Disadvantages:

1.Complexity: This architecture involves many AWS services, which can complicate management, monitoring, and troubleshooting.

2.Cost: While each service in isolation might seem cost-effective, the combined cost, especially with high traffic, can escalate. Kinesis Data Streams and Firehose costs can be especially significant.

3.Maintenance: Keeping track of updates, changes, and best practices for each service can be time-consuming.

4.Potential Latency: While many operations are real-time, introducing Lambda for transformations can add some latency to the streaming data, especially under high loads or if complex transformations are involved.

5.Lock-In: This architecture is deeply integrated with AWS services. If ever there's a need to migrate to another cloud provider or an on-premises solution, it would be a considerable effort.

6.Monitoring Overhead: Given the distributed nature of the services, setting up effective monitoring and alerting across all components can be challenging.

Task 2:

Infrastructure-as-Code (IaC)

Question1:

Why is IaC so important and what tools do you know to implement it on AWS? Can you explain the difference of those tools in a nutshell?

Solution:

Importance of IaC:

1. Uniformity & Consistency: Ensures standard configurations across various environments, promoting predictable deployments.

2. Decreased Risk: Minimizes human errors that arise from manual setups, mitigating potential issues.

3. Faster Deployment & Increased Speed: Accelerates the setup process, allowing quick launches or scaling of infrastructure.

4. Strong Security & Faster Recovery: Infrastructure blueprints can include security best practices, and recovery from failures becomes more predictable.

5. Financial Efficiency: Optimized and standardized resource provisioning can lead to cost savings.

6. Team Collaboration: Simplifies change management with version-controlled infrastructure blueprints.

7. Optimal Resource Allocation: Teams can focus on building features and applications rather than managing infrastructure.

8. Boosted Software Development Efficiency: Infrastructure alignment with application needs can expedite the development process.

9. Reliability: Standardized setups mean fewer unexpected behaviors.

10. Enhanced CI/CD Process: Seamless integration with DevOps processes, allowing consistent and reliable application delivery.

11. Cost-effectiveness: Proactive resource management leads to better financial outcomes.

12. Change Management & Versioning: Infrastructure changes are versioned, making rollbacks and audits easier.

Key AWS IaC Tools:

AWS CloudFormation: Native to AWS, it uses templates to define infrastructure.

Terraform: An open-source, multi-cloud solution with a platform-agnostic design.

AWS CDK: Allows infrastructure creation using familiar programming languages (like Python).

Recommendation for Our Project Using IaC:

1. Immutable Infrastructure: IaC ensures a consistent and reproducible deployment, vital for high-stakes e-commerce operations.

2. Dynamic Scalability: Adjust resources based on demand, vital for varying e-commerce loads, without manual intervention.

3. Collaboration & Versioning: IaC facilitates teamwork on infrastructure and maintains version histories, eliminating discrepancies.

4. Enhanced Security: Define security within the infrastructure code, ensuring a consistent and secure environment.

5. Cost Control: Precise scripting of resources with IaC prevents unexpected costs.

6. Fast CI/CD: Integrate IaC into CI/CD pipelines for rapid, reliable deployments.

7. Disaster Recovery: Rapidly redeploy infrastructure in emergencies, minimizing downtimes.

For our specific project's architecture, AWS offers tools like AWS CloudFormation and Terraform.

AWS CloudFormation: Specific to AWS, it offers deep integration with AWS services. It's a mature service provided by AWS that ensures resources are provisioned in the way AWS recommends. However, it doesn't offer the flexibility or multi-cloud capabilities of Terraform.

Terraform: Open-source and cloud-agnostic, terraform provides flexibility. It's not restricted to AWS and can be used across different cloud providers, offering a standardized way to deploy resources. It's especially helpful if there are plans to incorporate multi-cloud strategies in the future.

Question2:

Based on your solution in Task 1 (2nd Question), what might a Terraform implementation look like? You don't have to write code! It's enough if you show the structure of the code repo and explain what part of the infrastructure you define where.

Solution:

Note: This is a sample project repository structure provided for demonstration purposes. The contents and configurations within are dummy representations.

Repository Link: https://github.com/ShwetaProTest/ecom-aws-infrastructure

Terraform Structure for E-Commerce Behavioral Data Processing:

- Root (`ecom-aws-infrastructure/`): The main project directory.

- `.github/workflows/`: Houses CI/CD configurations for Terraform.

- `js/`: Contains JavaScript for capturing user behavior on online shops.

- `lambdas/preprocessing/`: Holds Lambda functions for data preprocessing.

- `scripts/`: Utility scripts aiding in deployment and environment setup.

- `terraform/`: The core of our IaC. Contains main configurations, variables, provider setups, and outputs.

- `terraform/environments/`: Environment-specific variable files to tailor configurations for development, staging, and production.

- `terraform/modules/`: Modular configurations for AWS services, ensuring organized and maintainable code. Each module focuses on a specific AWS service.

Overview: Our Terraform design emphasizes modularity, environment-specific customizations, and clear service boundaries. This ensures flexibility and easy maintainability as the project evolves.

Question3:

How can you define the infrastructure for a staging and production environment without duplicating the Terraform code?

CICD Deployment flow: User behavior data pipeline CICD

Solution:

Defining Infrastructure for Staging & Production without Duplication:

1. Modules: Our Terraform code is modularized, ensuring reusable components and avoiding redundancy.

2. Environment Variable Files: We use. tfvars files for each environment, supplying specific configurations while keeping the main logic consistent.

3. Doppler Integration: For secrets and configurations, we leverage Doppler. Depending on the deployment stage, Terraform dynamically fetches the required environment-specific secrets.

4. Terraform Workspaces: Separate state management for each environment is achieved using Terraform workspaces, allowing the same codebase to deploy different configurations based on the environment.