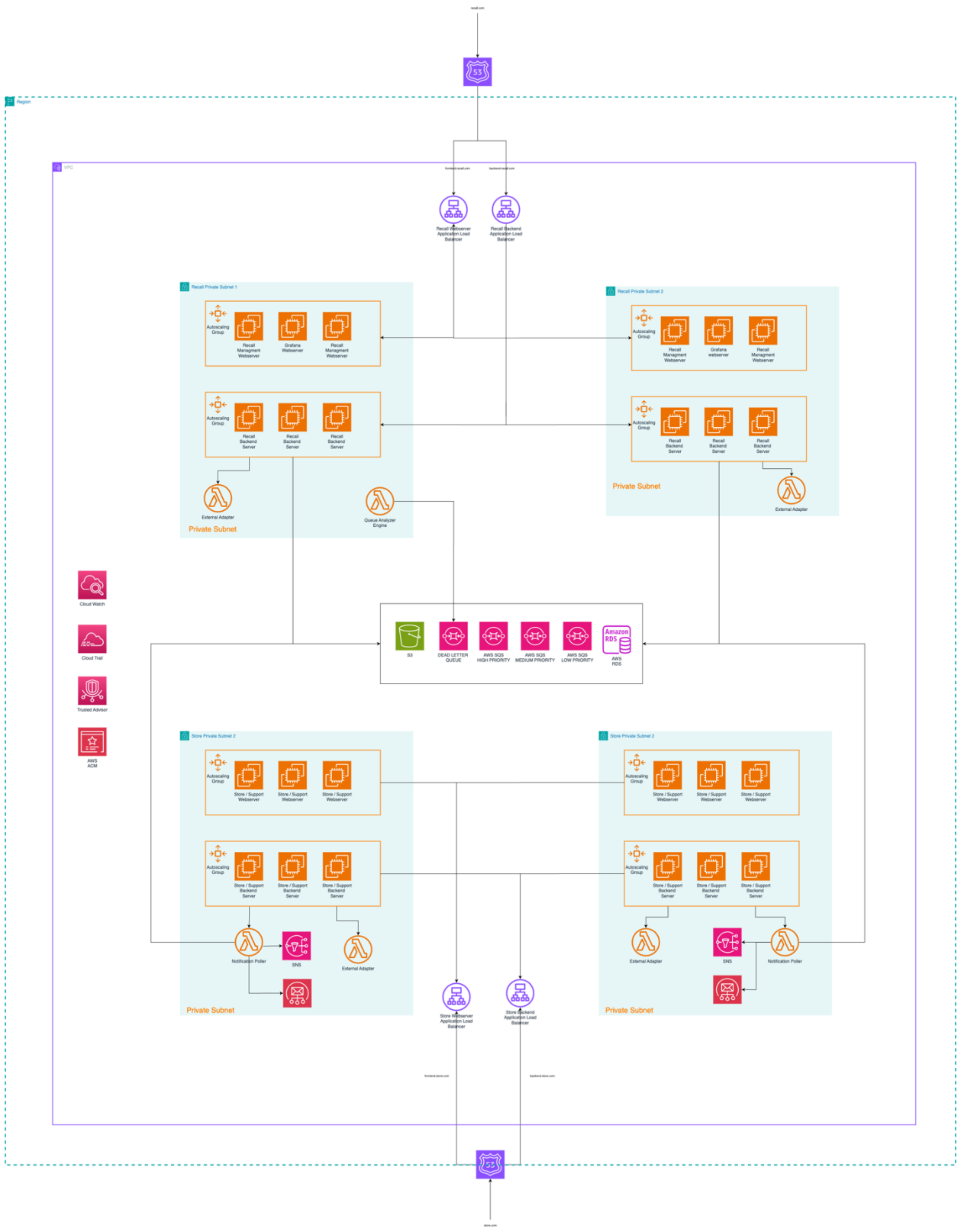
DEPLOYMENT ARCHITECTURE

Overview

The AWS deployment architecture for the recall notification system is meticulously designed to ensure high availability, scalability, and security. By deploying the application across multiple subnets within a VPC and employing various AWS services, the architecture supports a robust recall management process.



VPC and Subnet Configuration

The system is contained within an AWS VPC, enhancing security and network isolation. Within the VPC, there are four private subnets:

1. Recall Private Subnet 1
2. Recall Private Subnet 2
3. Store Private Subnet 1
4. Store Private Subnet 2

These subnets house the system's components, ensuring that resources are distributed across multiple AZs for fault tolerance.

Auto Scaling Groups

Each subnet contains Auto Scaling Groups (ASGs) configured for different tiers of the application:

- The front-end ASG contains the Management Server and Grafana for monitoring.
- The back-end ASG hosts the Recall Backend Server.
- The front-end ASG contains the store webserver for monitoring.
- The back-end ASG hosts the store Backend Server.

Auto Scaling ensures that resources match the demand at all times, spinning up or down instances as required.

Application Load Balancers

These ALBs are connected to AWS Route 53, which routes traffic to domain names such as frontend.recall.com and backend.recall.com, ensuring efficient load handling and domain-level traffic management.

The deployment architecture incorporates a suite of Application Load Balancers (ALBs) and leverages AWS Route 53 to manage traffic across various components of the recall notification system. Each ALB is designed to cater to different facets of the application, ensuring that the system is resilient, scalable, and responsive to user requests. Here's an overview of the role each ALB plays and how they interact with Route 53:

1. **Recall Web Server ALB**: This ALB is configured to manage incoming traffic for the front-end of the recall management application. It routes client requests to the Management Server, where corporate users interact with the recall system. The Grafana server, which falls under this ALB's jurisdiction, provides monitoring capabilities by displaying dashboards related to the system's performance and usage metrics.
2. **Recall Backend Server ALB**: This ALB is responsible for distributing load across instances of the Recall Backend Server, which handles the application logic and database interactions for the recall system. It ensures that backend processing such as queue management, recall data processing, and external integrations is efficiently managed and scales with demand.
3. **Store Webserver ALB**: Dedicated to the store component of the application, this ALB handles the traffic directed to the store webserver(s). It facilitates store users' access to the system for managing and responding to recall notifications.
4. **Store Backend Server ALB**: This ALB balances the load for the store backend server(s) responsible for processing recall-related actions taken by the store, such as notification polling and response tracking. The backend also deals with inventory management and communication with other services within the recall system architecture.

AWS Route 53 Configuration

Route 53 integrates with each of the ALBs to provide a robust and flexible DNS web service that routes user requests to the appropriate load balancer:

- By mapping domain names such as frontend.recall.com to the Recall Web Server ALB and backend.recall.com to the Recall Backend Server ALB, Route 53 simplifies the access points for different users interacting with the recall system.
- It employs health checks to ensure that traffic is only directed to healthy instances. If an ALB endpoint is deemed unhealthy, Route 53 can reroute the traffic to other operational endpoints, minimizing potential downtime.
- Route 53's traffic policies can also be configured to implement more advanced routing techniques, such as geolocation routing, latency-based routing, and weighted round-robin routing, all of which ensure that end-user requests are handled efficiently and with reduced latency.

- For the store components, similar domain name management ensures that store users can reliably access the store's web interface and backend services, maintaining consistent performance across the retail network.

By utilizing ALBs and integrating them with Route 53, the architecture ensures that different aspects of the recall notification system are both isolated for security and performance reasons, and integrated for seamless user experience and system reliability. This approach underscores the system's readiness to manage large-scale, distributed recall events with the agility required for modern retail operations.

External Adapter and AWS Lambda

An external adapter is implemented using AWS Lambda to interface with third-party services like Microsoft Teams via webhooks. This serverless approach is cost-effective since Lambda functions run in response to events and only incur costs per execution.

Amazon RDS vs. AWS Aurora

Amazon RDS is chosen for its cost-efficiency over AWS Aurora. RDS supports the MySQL database, managing user and inventory data. While RDS provides a cost-effective solution, Aurora could be an alternative for more scalability and performance. Aurora delivers higher throughput and resilience, suitable for more demanding workloads.

Notification System

The notification system within the recall notification architecture is designed to ensure that stores receive timely alerts about product recalls. A crucial component of this system is the Notification Poller, a Lambda function, that interacts with Amazon Simple Queue Service (SQS) to manage and process recall notifications based on their assigned priority levels.

Notification Poller and Priority Queue Processing

Here's how the Notification Poller handles different priority queues in SQS:

1. **High Priority Queue**: This queue contains recall notifications that require immediate attention. The Notification Poller is configured to prioritize this queue over others. It will first poll messages from the high priority queue, ensuring that the most critical recall events are addressed promptly.
2. **Medium Priority Queue**: Once all messages from the high priority queue have been processed, or if there are no messages in the high priority queue, the Notification

Poller moves on to the medium priority queue. These notifications are important but do not require the immediate action that high priority messages do.

3. **Low Priority Queue**: Lastly, the Notification Poller checks for messages in the low priority queue. These recalls are less urgent and can be scheduled for action after the more critical recalls have been addressed.

Integration with SES and SNS for Notification Delivery

Upon retrieval of messages from the appropriate SQS queue, the Notification Poller triggers the notification process through two AWS services:

- **Amazon Simple Email Service (SES)**: This service is used to send email notifications to store managers or other designated personnel. The emails provide details of the recall event, including the priority level and the specific actions required.
- **Amazon Simple Notification Service (SNS)**: In addition to emails, SNS is employed to send text messages or push notifications directly to users' devices. This ensures that the recall information reaches the relevant store personnel quickly, allowing for rapid response.

Design Rationale for Priority-Based Queue Processing

The decision to implement a priority-based queue system with SQS and process them sequentially from high to low priority is driven by the need to:

- Respond swiftly to the most critical recall events to mitigate potential risks to consumers and comply with regulatory requirements.
- Manage resources effectively by addressing the most impactful recalls first, which can help in reducing potential financial and reputational damage.
- Provide a structured response plan for store personnel, enabling them to organize their workflows according to the urgency of each recall notification.

This system is designed to be both responsive and resilient, ensuring that the notification process is not only fast but also reliable. By employing AWS Lambda for the Notification Poller, the system benefits from the scalability and cost-effectiveness of serverless computing, paying only for the compute time used, with no need to provision or maintain servers. The integration of SQS, SES, and SNS reflects the system's commitment to leveraging cloud-native services to create a seamless, efficient, and highly available notification system that scales dynamically with the demand.

Monitoring and Compliance

For SSL/TLS certificate management, AWS Certificate Manager (ACM) is used, ensuring encrypted connections. AWS Trusted Advisor, AWS CloudTrail, and AWS CloudWatch provide monitoring, logging, and compliance. These services are integral for operational oversight, security auditing, and real-time alerting.

Design Decisions and Preferences

The architecture demonstrates a preference for managed services and serverless computing, focusing on scalability, cost management, and operational simplicity. ASGs show a commitment to handling variable load, while ALBs ensure high availability and fault tolerance. AWS Lambda for external integrations emphasizes a serverless-first approach, minimizing infrastructure management and operational costs. The choice of RDS over Aurora is a cost-based decision, with the option to scale vertically with Aurora if necessary.

Alternative Services and Comparison

- **ECS/EKS**: Amazon ECS or EKS could be used for container management instead of EC2 instances in ASGs, providing advanced orchestration features.
- **Amazon DynamoDB**: DynamoDB could replace RDS for non-relational data needs, offering fast performance and seamless scalability.
- **Amazon API Gateway**: This could provide managed API endpoints for Lambda functions instead of direct invocation, offering additional security and monitoring features.
- **Direct Connect**: For consistent, high-performance connectivity, AWS Direct Connect could be considered, especially if hybrid cloud architecture is desired.
- **AWS Fargate**: For serverless container execution, AWS Fargate could replace EC2 instances, removing the need to manage servers or clusters.

Amazon SQS vs. AWS MQ (with RabbitMQ as a broker):

- **Management Overhead**: SQS is a fully managed service with no infrastructure to manage, making it easier to set up and scale. AWS MQ requires some management for the broker, and RabbitMQ, especially self-hosted, requires substantial setup and maintenance.

- **Scalability**: SQS scales automatically and can handle high throughput without user intervention. AWS MQ can scale but not as seamlessly as SQS, and it is limited by the broker instance type. RabbitMQ can be scaled manually, which requires deep knowledge and can be complex.
- **Features**: SQS provides simple queuing services with fewer messaging features. AWS MQ offers full broker features, including topic-based pub/sub and message ordering. RabbitMQ has a wide feature set and plugins but requires manual management.
- **Use Cases**: SQS is ideal for decoupled microservices in a cloud-native environment. AWS MQ is suitable when you need traditional messaging features like JMS, or if migrating from an on-premise message broker. RabbitMQ is often chosen for legacy applications that rely on its specific features and where migration to a managed service isn't feasible.

Grafana vs. Amazon QuickSight:

- **Complexity and Use Cases**: Grafana is an open-source platform known for its advanced monitoring capabilities and complex visualizations, suitable for technical users. QuickSight is a managed service that is more user-friendly, designed for business intelligence use cases, and integrates easily with other AWS services.
- **Scalability**: Grafana can be scaled but may require additional infrastructure and setup, while QuickSight scales automatically without any infrastructure management.
- **Cost**: Grafana can be cost-effective as it's open-source, but there might be costs associated with running servers or managed instances. QuickSight has a pay-per-session pricing model which can be cost-effective for light usage with few clients but might become expensive at scale.
- **Integration**: Grafana has a wide range of integrations with various data sources, while QuickSight provides deep integration with AWS data services and easy setup.

The current architecture is robust, yet flexible enough to incorporate these alternative services should the system requirements evolve. The decision to use AWS Lambda for external adapters and notifications, as well as the split between management and operational concerns within the VPC subnets, shows a thoughtful approach to balancing cost, performance, and security.