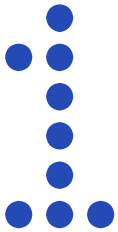
# Where commerce happens on AWS.

*How Punch helped assure global merchants were open for business.*

instore

**punch** case study

# *Contents*



# *About Instore*

**INSTORE IS A LEADING 360-DEGREE VIEW POINT-OF-SALE SOLUTION FOR RETAILERS AND RESTAURANTS, PROVIDING BEST-IN-CLASS CHECKOUT, LOYALTY, AND REWARDS FUNCTIONALITY.**

## WHERE COMMERCE HAPPENS

Instore's leading design and ui/ux combined with its comprehensive feature set have made Instore a favorite among restaurants and retailers for its comprehensive, real-time business management capabilities.

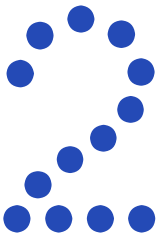
When it came time for Instore to scale and meet the ever growing demands of its clientele, Punch helped to create multiple layers of fault tolerance and redundancy on Amazon Web Services (where Instore's servers are hosted) to potentially eliminate power, internet, machine, and data loss outages.

The result is an exciting partnership as Instore writes the next chapter of its commerce revolution.

**PUNCH SERVICES PROVIDED**

Punch Provided Expertise in platform engineering, developer operations, site reliability, and staffing to help Instore meets deadlines and goals for rapid development.

# Platform Architecture, Site Reliability, Database Engineering, Developer Operations, Staffing



# *An Achille's Heel*

**INSTORE'S CLIENTS - GLOBAL SMALL-TO-MID-SIZED RETAILERS AND RESTAURANTS – EXPECTED UP-TO-THE-SECOND DATA ANALYTICS TO FULFILL INSTORE'S BRAND PROMISE.**

The problem: how to architect a setup on AWS that served multiple layers of redundancy to fulfill on this brand promise, while keeping costs in line?

1. **Connection/I.P.Rotation.** The datastore/server and the web app serving were hosted in different infrastructure platforms. If an I.P. address were to rotate, connection between the web application and the server would sever.

2. **Downed Machines.** If a machine serving the web application (or database) were to crash, a platform developer would have to manually fix the issue, wasting valuable time.

3. **Software Stack Fatal Errors.** The back-end node.js application, in edge cases, was throwing fatal errors and crashing. While the developer team would quickly try to fix these issues, these fatal errors would bring all connected clients for that regime offline.

4. **Database Crashes.** If the database servers CPU utilization or available disc space would exceed reasonable thresholds the machines would slow to a crawl, leading to slower and slower load times and ultimately a crash.

# 380,000

**NUMBER OF YEARLY TRANSACTIONS PER STORE**

Instore's software powers hundreds of thousands of transactions per store per year.

# *The fix for Instore*

**PUNCH WORKED WITH INSTORE TO SOLVE THESE FOUR KEY ISSUES. WE IMPLEMENTED A SERVICE-ORIENTED-ARCHITECTURE ON AWS.**

We worked closely with Instore and their clients to research, architect, script, and execute solutions using AWS management console and SDK. We identified five key areas that formed the pillars of our solution.

**AWS SOLUTION CORE FEATURES**

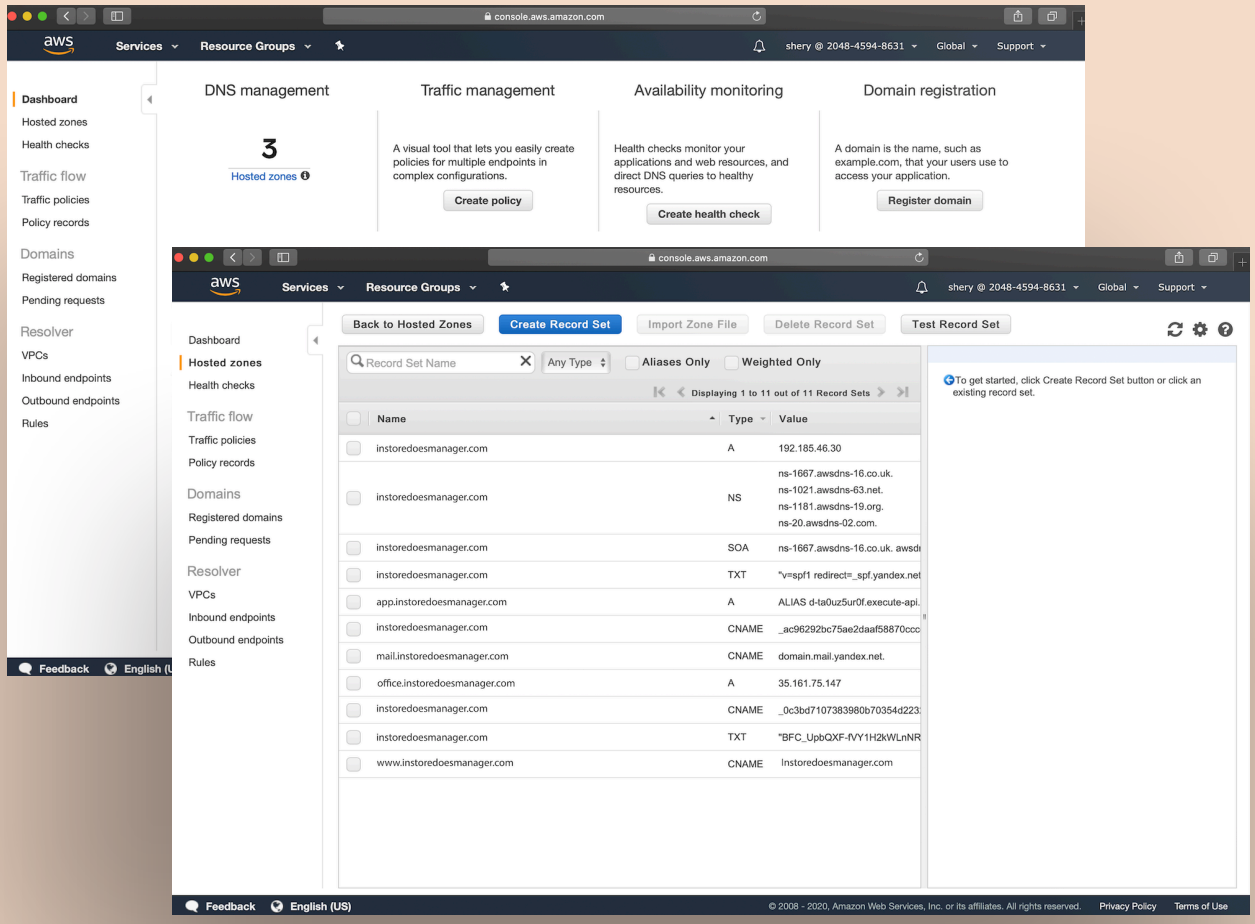
| | |
|---|---|
| **Route** | Use an intelligent router with static public IPs connected to static internal IPs to execute traffic. |
| **Balance** | Implement Elastic Load Balancers (ELB) to collect traffic. Have load balancers ping each virtual machine to determine real-time availability and status. |
| **Access** | Access policies and security groups determined per-user level access. Separate database, deployment, and provisioning groups enabled fine-grain user access controls. |
| **Backup** | Enable regular database backup schedules to Amazon Simple Storage Service (S3). Replicate node servers using Forever. Create a master-server shared database cluster using volume and ephemeral storage. |
| **Monitor** | Implement individual server monitoring with Amazon CloudWatch. Use Amazon Simple Messaging Service (SNS) to notify the Punch and Instore teams of any potential issues. |

1

## ROUTE

### USE AN INTELLIGENT ROUTER WITH STATIC PUBLIC IPS CONNECTED TO STATIC INTERNAL IPS TRAFFIC.

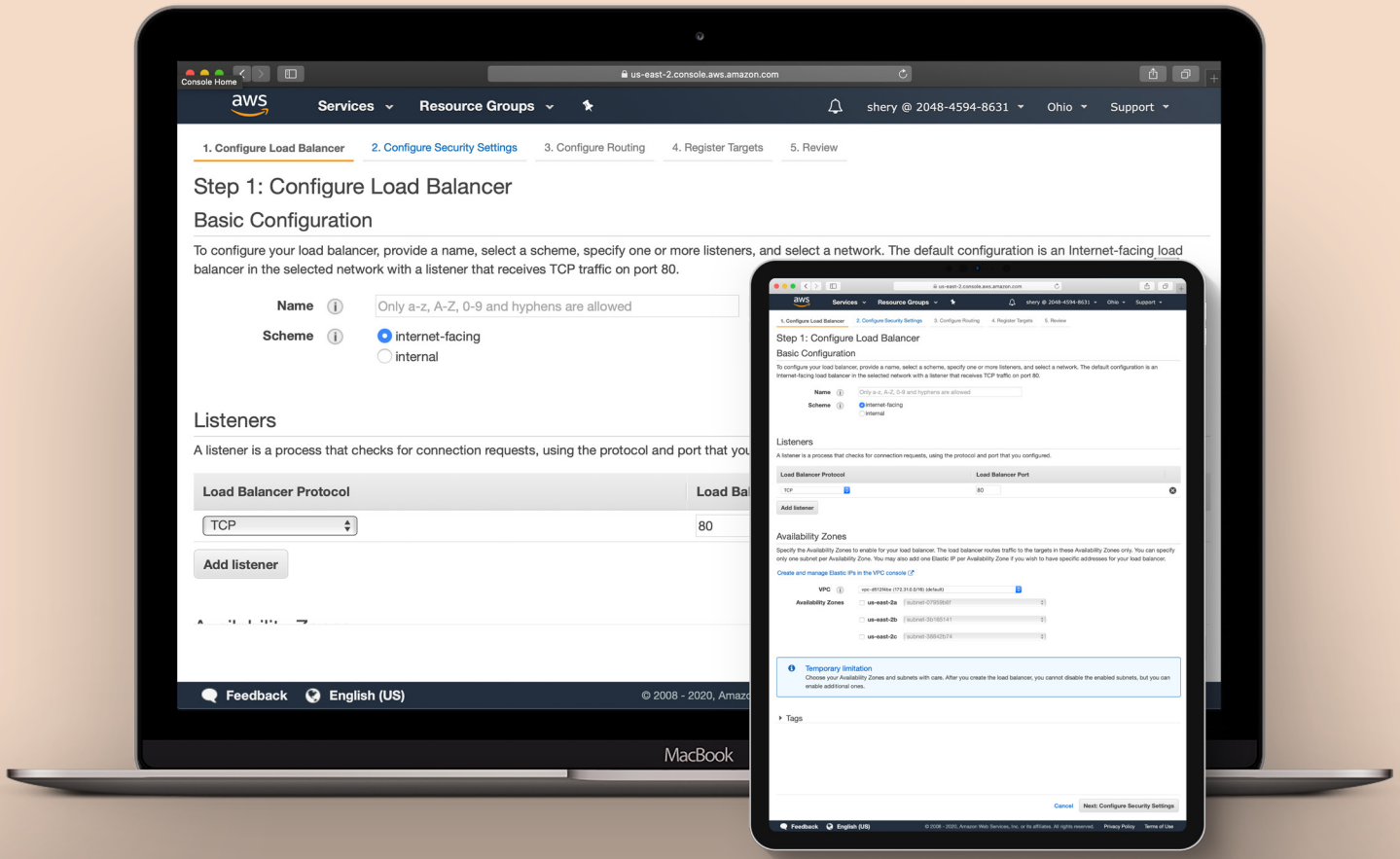
Route 53 enabled us to have static public facing IPs while routing to internal static IPs. The flexibility of Route 53 enabled us to route traffic to multiple regions to mitigate against platform level datacenter risk.



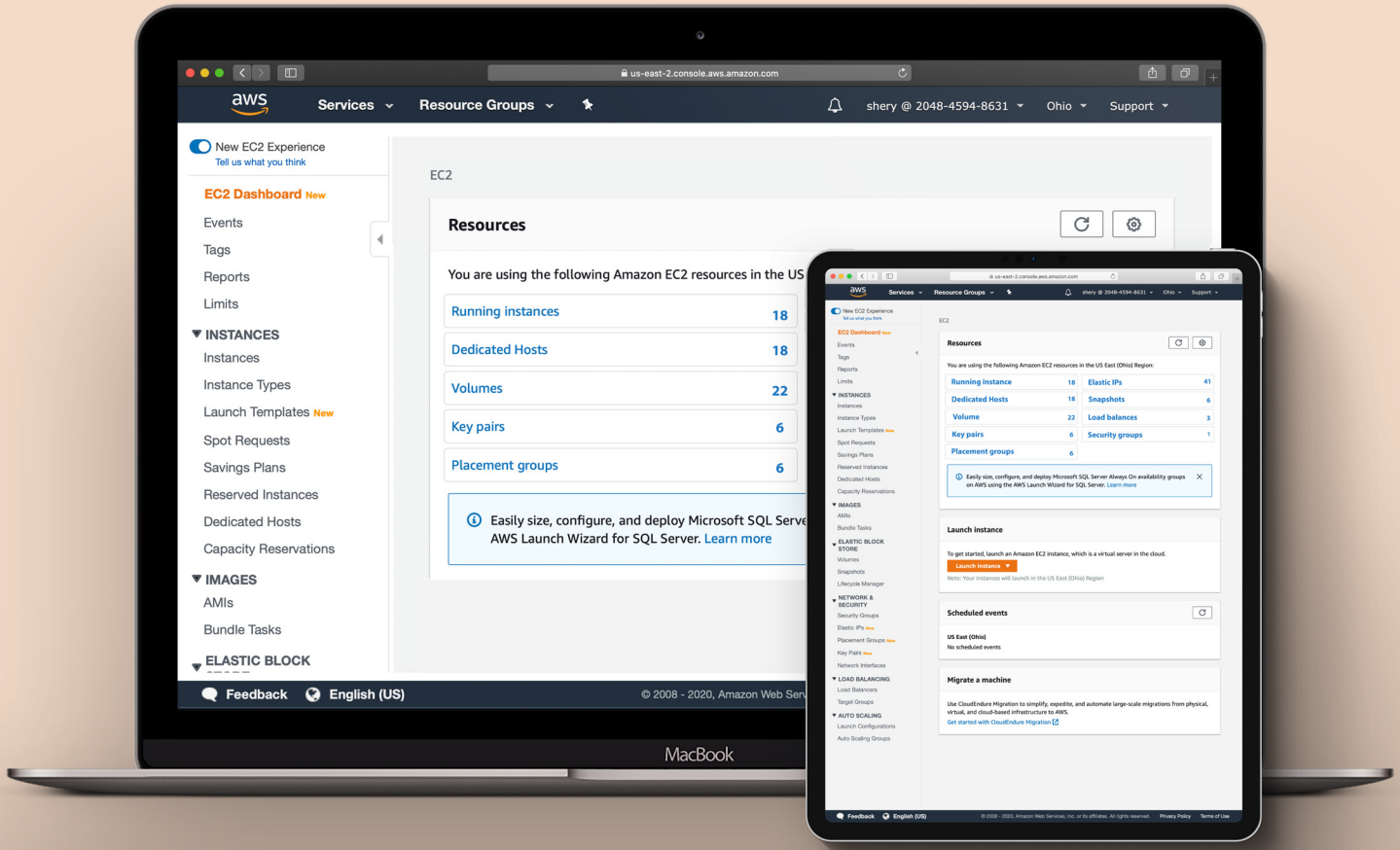
# 2

## BALANCE

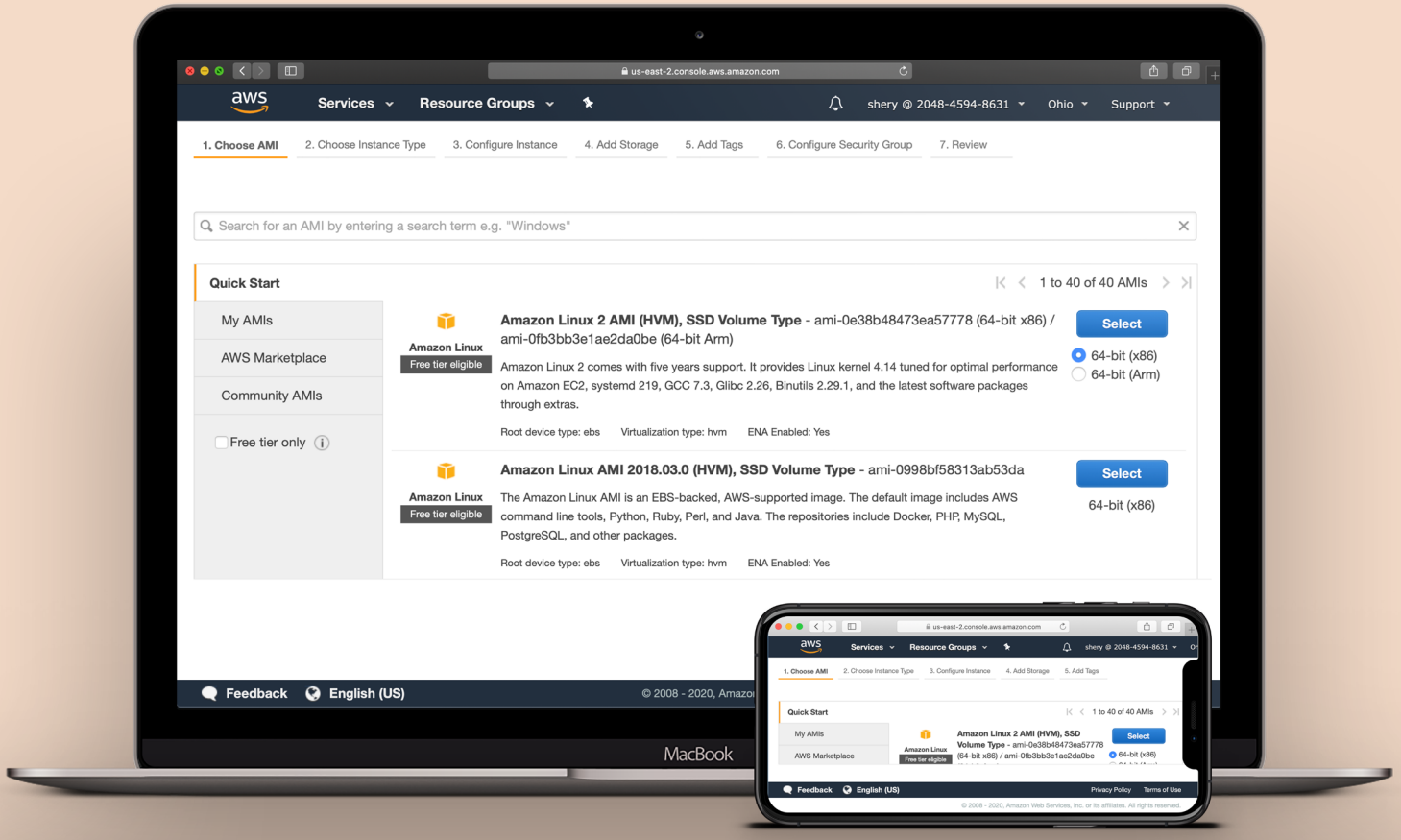
We implemented Load Balancers (ELB) to collect traffic and enabled real-time pinging of each virtual machine within each load balanced cluster to determine real-time availability and status.



Running instances were automatically queued and dequeued from AMIs to fit network traffic. If an ELB pinged an EC2 instance and it was unresponsive a new instance would be auto-generated in its place and attached programmatically to the least allocated ELB.



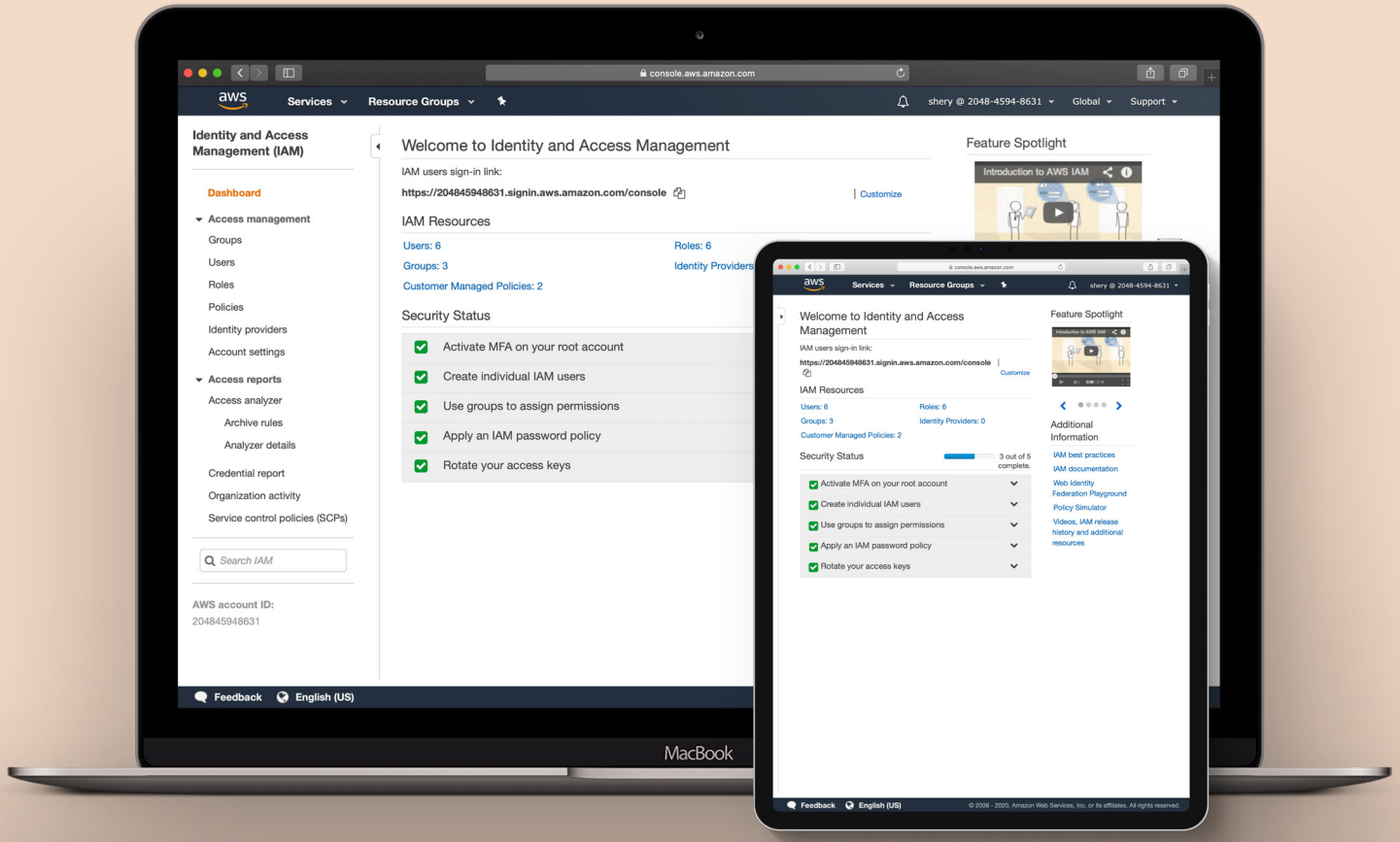
New instances were auto populated from pre-set AMI custom configurations. This ease-of-use enabled our developer operations team to automate the deployment and scaling / descaling process, while serving best-in-class uptime during peak load.



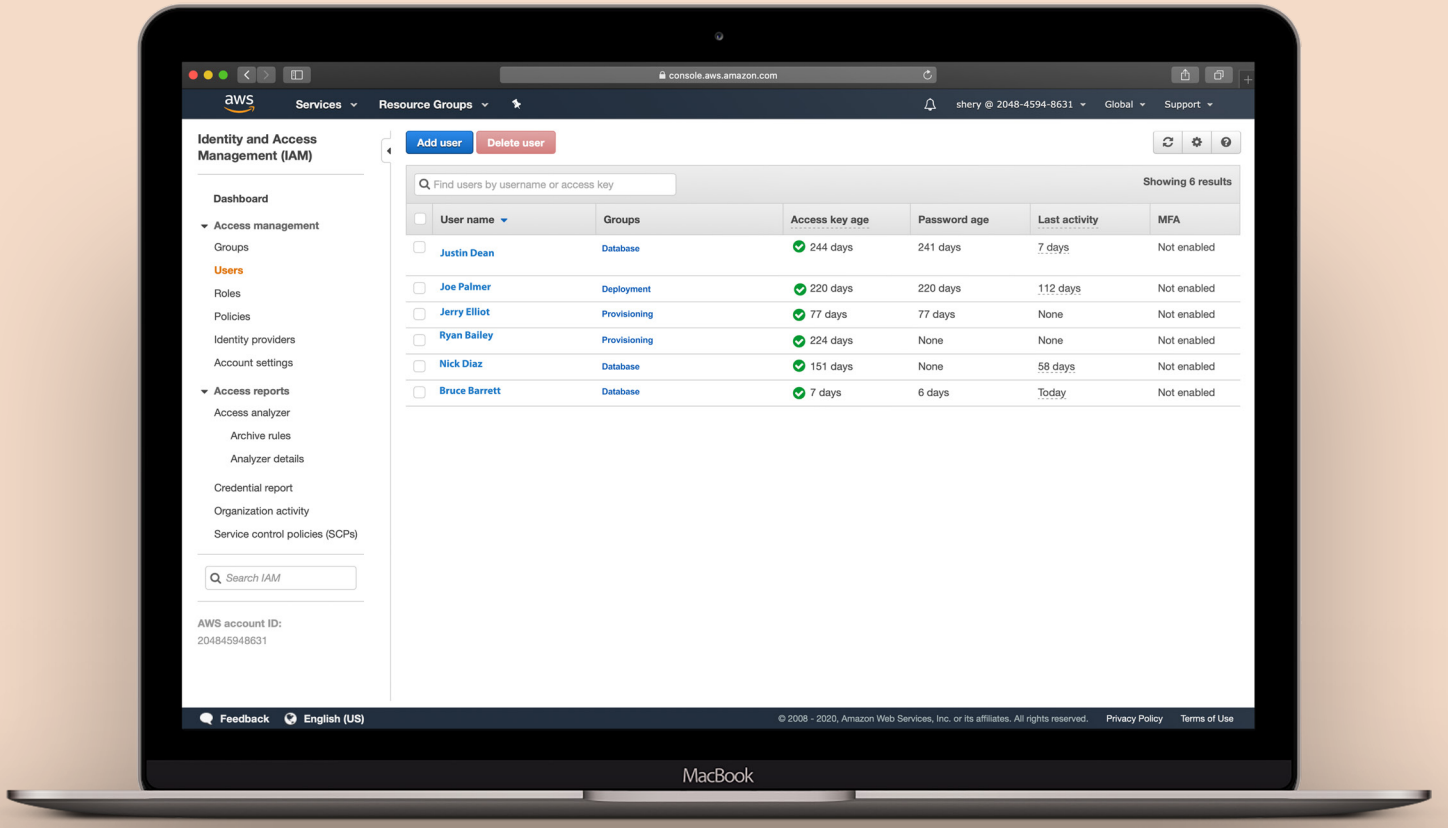
# 3

## ACCESS

Access policies and security groups determined per-user level access. Separate database, deployment, and provisioning groups enabled fine-grain user access controls.



Users security keys were rotated and access keys updated yearly to help mitigate security breaches. Least access privilege helped prevent a "God-user" from leaking sensitive information publicly and compromising user data.



*“Punch has moved **swiftly** and took control of all our problems. Great team and **effective** methods.”*

---

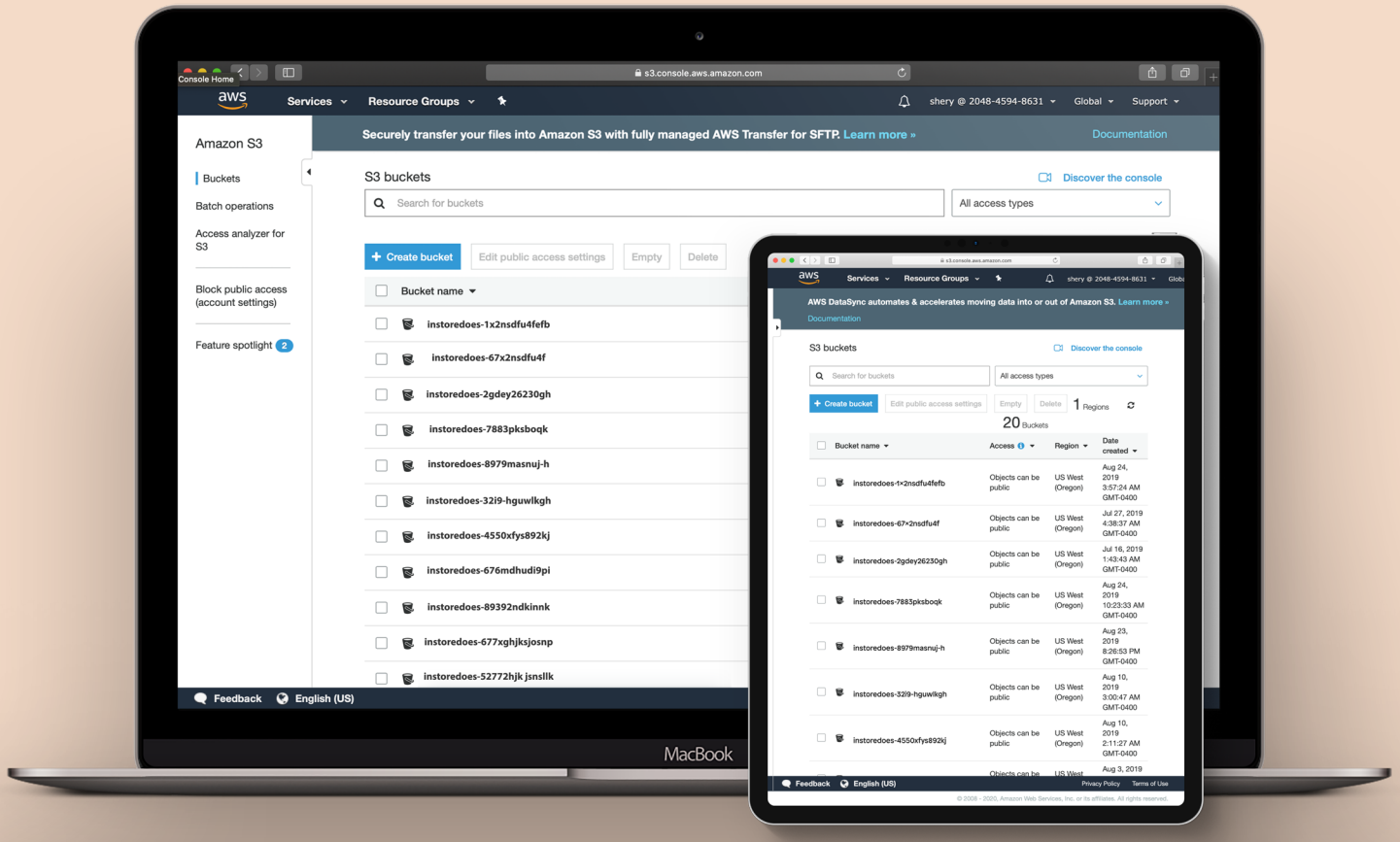
MATT

*CEO*
*Instore*

# 4

## BACKUP

Instore's front-of-the-house point-of-sale system generated tens of thousands of daily transactions, with many more line-items. All this information needed to have redundant back-ups in AWS, and Amazon Simple Storage Service (S3) was used to enable database backups with date and timestamping. S3 backups were programmatically rotated to keep S3 total utilization to a minimum.



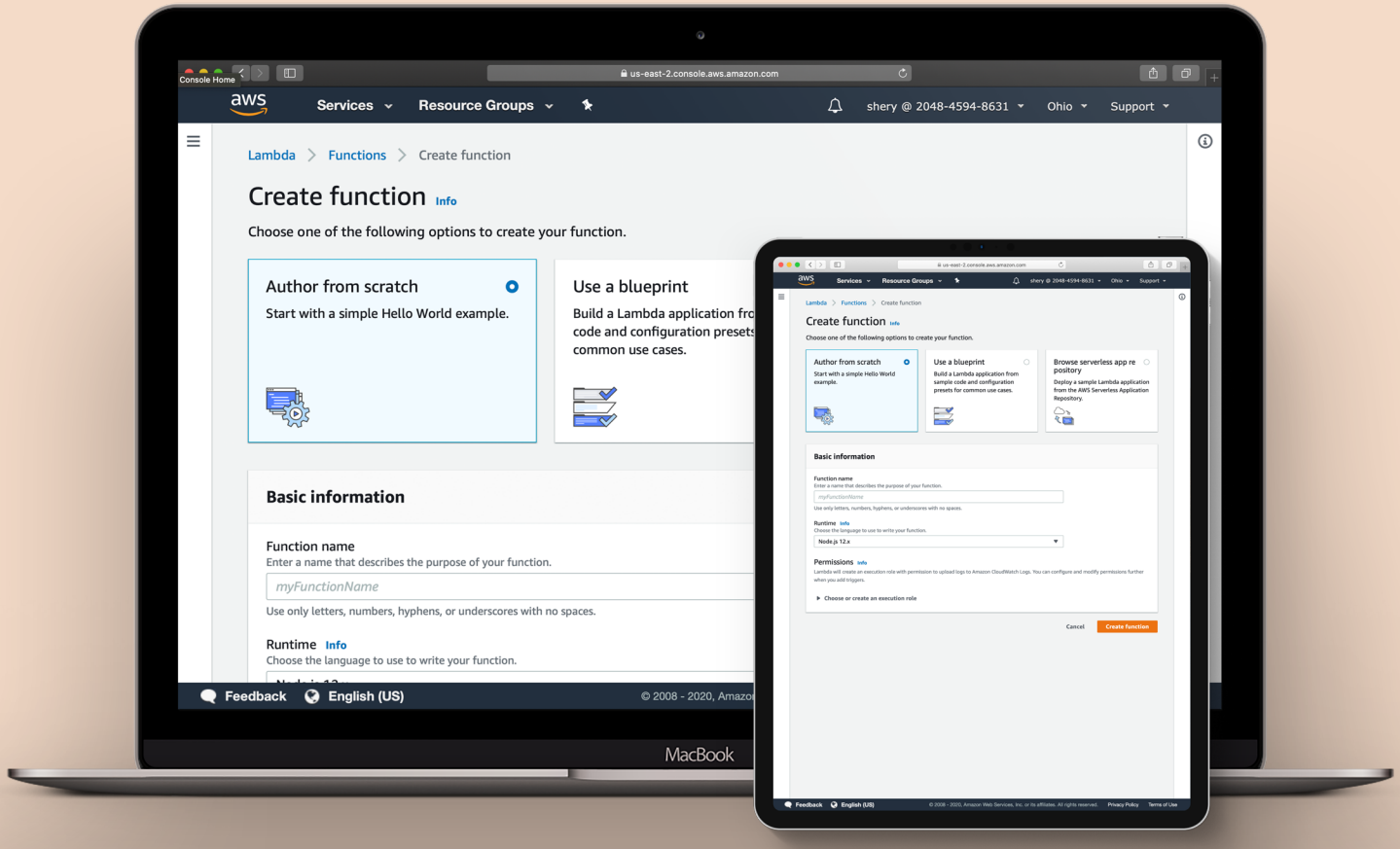
Instore's back-of-the-house dashboard reported on the very data that required backing up – with any lapse in uptime operators wouldn't have access to their analytics. With AWS platform development and testing, we achieved 4-and-a-half-9's Uptime (99.995%) – less than 27 seconds of downtime per month.

# 5

## MONITOR

AMAZON CLOUDWATCH WITH AMAZON SIMPLE NOTIFICATION SERVICE (SNS) MONITORED EC2 AVAILABILITY. IF ISSUES AROSE, AMAZON LAMBDA (CLOUD FUNCTIONS) WAS USED TO EXECUTE A SERIES OF RISK MITIGATION STEPS PROGRAMMATICALLY.

AWS lambda resolved disk capacity, I/O, CPU utilization, and hardware failure issues on each EC2 instance. These script executions were then logged and browsable by the developer operations team.



# *Results*

**PUNCH PRODUCED AN ENTERPRISE APP SOLUTION FOR INSTORE TO HELP GLOBAL MERCHANTS.**

**APP STATISTICS**

| | Items |
|---|---|
| Total Virtualized CPUs Utilized | **128** |
| Length of Project | **4 months** |
| Tech stack | Amazon EC2, S3, CloudWatch, ELB, Route53 |
| Teams involved | San Francisco |

**Legal**

Prepared on March 3, 2020. This document is confidential. It contains material intended solely for the original recipient. Ideas presented here are the property of Punch and are copyrighted.



punch