Private Outputs

Web Game Development

- AirPoker: A game from a certain manga. Next.js + WebRTC + MongoDB
- To Court the King: A famous board game. Next.js + WebRTC
- Slack CodeGolf: Node.js + WebSocket on Heroku

OSS Contributions

- Node.js Core: Contributions to timers and hidden class optimizations, documentation, etc.
- Initial release contributions to AI Speech Synthesis TTS Style-Bert-VITS2
- AI Singing Synthesis NeuCoSVC
- kotoba-whisper

Side Jobs

- kakekomu (ST Booking): Assisted for about a month. Worked on question notification features, JWT encryption, test expansion, etc.
- meetsmore: Assisted for about a month. Migrated application data from MongoDB to BigQuery (table/partition design, Digdag batch processing, containerization)
- Smoothy: A side business partnered with a Google sales. A reservation service for restaurants (React Native, service ended)
- Kotoba Technologies: Assisting now about AI product

Others

- Presentation History: https://github.com/darai0512/talks/blob/master/README.md
- Sales and Distribution:
  - OBS screen reflection animation for Twitch stream comments (js + WebSocket + css)
  - VR accessories created with Blender/RVC models/VITS models/Image LoRA: Sold on Booth
- Hackathons:
  - Domestic: Filed a patent for an alarm game (iOS)
  - Taiwan Hackathon: Participated twice in Taiwan. Developed prototypes and gave presentations in English. Still in contact with local engineers
- LeetCode: See gist
- Qualifications: Certified Scrum Developer (CSD)

Work History: Online Payment System (API & SDK)

Overview

Worked at a B2B service company providing online credit card payment functions (PCI-DSS compliant APIs and SDKs). Achieving IPO during my tenure was an invaluable experience.

Duration

Approximately one year in 2018 as a contract employee (utilizing the side job system of my previous company while still employed there), and from 2019 to the present as a full-time employee.

Scale

Always operated with a small team of about five engineers (+ one PCI-DSS security specialist), all mid-career professionals.

Approach

For maintenance and operation of the main payment service platform (BE: Python Gunicorn + Pyramid, DB: Aurora PostgreSQL, FE: Vue.js + Jinja) and back office (same stack), we adopted a Scrum format using Notion (previously Asana) for task management (fully remote since COVID-19).

All team members, including engineers, business staff, customer support, and managers, participated in the formulation and refinement of the product backlog, prioritizing and assigning points to tasks, and working through them in order.

For each separate project, we gathered and proceeded with similar management.

Ad-hoc tasks (calculating KPIs with Redash or BigQuery, responding to technical inquiries from client companies) were handled on a duty roster basis.

Achievements (6 Highlights)

Platform Service Renewal

- **Effort**: About 4 months with 3 engineers and 2 PMs
- **Role**: As a player, involved from requirements definition to DB design, BE design, and overall FE development

- **Challenges**:
  i. The Platform, a B2B2B function (our company to platformer to tenant), had issues due to the creation of a new account system, which meant we couldn't reuse code or data from the main service (cf. Managed as a monorepo, service logic based on accounts was shared across APIs, their GUI management screens, and back office in the main service).
  ii. There was a demand to use it in a B2B2C setting (our company to platformer to individual business owners), but the existing setup and data structure were not suitable.
- **Solution**:
  - By **redesigning the data structure from scratch and integrating it into the account system of the main service**, we were able to share code and data, and modified the main service so that new features could also be used on the Platform.
    - Compared to the main service (B2B), the Platform (B2B2B) had nested account structures. After weighing the pros and cons, we decided not to use recursion but to integrate the root accounts into existing tables and separate nested parts into different tables, linking them to each resource with nullable foreign key constraints.
    - Unified the previously separate management screens. While differences arose in login screens and API structures after logging in (increased use of if-branches in UI components and services), we covered this with DI and flattening of nested structures, and the benefits of unification outweighed the drawbacks.

Automation of Chargebacks (Mechanism to Forfeit Sales in Case of Fraud)

- **Effort**: About 6 months with 2 engineers and 1 PM
- **Role**: As the development leader, involved from planning and requirements definition to state transition diagram creation, DB design, BE design, and development
- **Challenges**:
  - The chargeback process involved repeated email exchanges with upstream parties (like card companies) and merchants, and sending invoices—a complex manual workflow. The task was also becoming dependent on specific individuals.
- **Solution**:
  - **Participated from the stage of formulating exchange formats with upstream parties**, aiming to standardize formats and workflows as much as possible. For remaining differences, we built batch processing that could flexibly read them.
  - Mapped the complex state transitions into a state machine.
  - To prevent users from preferring Excel input over the management screen's input form (UI of Excel format), we improved the design and enhanced features like auto-fill and batch import.
  - Created logic to offset amounts from sales proceeds, reducing the need to send invoices, which are prone to default and difficult to automate.
- **Development**:
  - Led two contract engineers (rotating every three months) as the only full-time engineer. To bridge the gap in domain knowledge, I facilitated frequent video calls and casual interactions, and used concrete code changes on GitHub to minimize misunderstandings instead of relying on specialized terminology.

Multi-Account Support + PCI-DSS + Vue.js 2 EOL Support + Design Overhaul

- **Effort**: About 5 months with myself as the sole engineer, one designer, and one PM
- **Role**: As a player, involved from planning and requirements definition to DB, BE, and FE design and development
- **Challenges**:
  i. Revamp the management screen to allow multiple users with roles and permissions within a single account (similar to IAM), moving away from the one-company-one-account assignment.
  ii. Implement operation logs on the management screen as required by PCI-DSS.
  iii. **Migrate from Vue.js 2 to 3 due to EOL**.
  iv. Overhaul the management screen design. **The designer could interact via Figma but couldn't handle Vue.js or set up a local environment**.
- **Solution**:
  - Created a role table logic that allows permissions management through settings in the Pyramid interface, with simple ERDs and interface expansions.
  - Achieved permissions management and operation recording without modifying code for over 100 endpoints, using decorator-based settings additions.
  - To facilitate communication with the PM and designer, ensured that staging deploys ran automatically, allowing for screen-based discussions. This reassured the PM and helped with quick bug reports and spec reviews.
  - For the Vue.js 2 to 3 migration, aimed for minimal changes by retaining the Options API from the 2.x assets.
  - The bottleneck was library compatibility: We used the VeeValidate library's 2.x version in over hundreds of places, but Vue 3 compatibility starts from VeeValidate 4.x, with many breaking changes between versions 2 to 3 to 4.
  - **Searched for a workaround to apply Vue 3 while still using VeeValidate 2.x without modifying the library internally**, by exploring code-level solutions.
  - By extending non-public methods and properties and using mixins, managed to apply Vue 3 while keeping the existing code and VeeValidate 2.x.
  - Needed to address issues where input/change events and nextTick timings didn't align, causing real-time validation to fail. Leveraged knowledge of event loops to ensure appropriate nextTick usage.
  - While this migration method isn't fundamental (if vulnerabilities are found in VeeValidate, it's insufficient even if Vue vulnerabilities are addressed), we covered this by fixing the VeeValidate version (with no known vulnerabilities at that time), conducting QA from scratch, performing penetration tests during PCI-DSS audits, and establishing an exit strategy guideline (confirming that VeeValidate 2.x and 4.x can coexist within one package and using VeeValidate 4.x for new developments).
- **Remaining Challenges**:
  - Difficulty in collaborating with the designer.
    - The cost of environment setup and learning Vue.js was a barrier, so we managed HTML, CSS, and JS in a separate repository, and I decomposed and integrated them into Vue.js. However, when designs were revised, it was easy to lose track of where to reflect those differences.

Addressing Advanced Technical Issues (e.g., Moving Away from Custom Extended Cryptography/PyOpenSSL)

- **Challenges**:
  - At that time, PyOpenSSL didn't support certificate verification for Apple Pay (retrieving oid of X509Extension, ECDH derivation, etc.), so a predecessor had forked it for Apple Pay payment support. Later, due to OS updates and PCI-DSS compliance, we needed to move away from the old version of Cryptography/PyOpenSSL forked earlier, but there was no one with the surrounding knowledge after the predecessor left.
- **Solution**:
  - First, I read the code added in the fork, then compared it with the latest versions.
  - For example, support for X509Extension was included in the latest versions with certain libssl versions and conditional compilation (if-def macros), allowing us to discard some of the code added in the fork if conditions were met.
  - Wrote detailed tests and shared certificate verification and encryption logic with reviewers in an accessible manner. Managed to discard the fork and catch up with the main branch.
- **Others**:
  - Addressed issues like the exception handling in urllib3 when TLS handshake timeouts occur, where exceptions were broadly grouped under the same class, etc.

Containerization and Infrastructure Overhaul

- **Effort**: Initial technical investigation and basic setup by myself over six months, followed by three months with three engineers, one dedicated security specialist, and one PM.
- **Challenges**:
  - To reduce the workload of PCI-DSS compliance tasks (e.g., constant OS updates), we moved away from self-managed EC2 instances and implemented:
    - Containerization using Fargate
    - Infrastructure code management using Terraform
    - Replacing self-managed Nginx with a structure of CloudFront -> ALB -> ECS
    - Strict outbound policies (all external communication goes through a proxy)
- **Solution**:
  - Structured Terraform directories using workspaces (multi-provider and module separation) to ensure identical configurations for staging and production.
  - Migrated Nginx functionalities like path rewrites, header additions, and rate limiting implemented in Lua to CloudFront Functions + Terraform.
  - Switched from Ansible deployment to deployment using Ecspresso described in Jsonnet.
  - Application modifications due to containerization: Moving away from filesystem dependencies, logging settings, environment variable-based configurations (reading settings from tfstate/secret manager).
  - Configured proxies for requests/urllib3/various SDKs (Sentry, BigQuery, OneLogin, etc.). While most could be handled with the HTTP_PROXY environment variable, care was needed as Boto3 for internal communication, which doesn't require a proxy, could be affected.
- **Note**:

    - Initially spent a few months planning for EKS (Kubernetes) with Helm + DevSpace. However, after warnings from the AWS support team about the challenges of long-term EKS operations and discovering features difficult to implement on EKS on Fargate, we shifted to ECS.

SDK Maintenance and New Development

JS Card Information Input Library

Developed a latest PCI-DSS compliant iframe-type (does not go through merchant domain) e-commerce JS library in 1.5 months as a single engineer.

- Referenced globally recognized libraries through reverse engineering.
- Expanded documentation such as samples aligned with Material Design to support migration from previous JS versions that didn't meet requirements. Successfully achieved complete migration.

Faced challenges in ensuring operation across various environments for end-users, unlike management screens where browsers can be restricted.

- Automated tests with Playwright for Chrome/Safari/Firefox.
- Used BrowserStack for iOS, Android, and Windows.
- Dealt with numerous edge cases, such as window.open not working only within the Line app's internal browser, or country code emojis not displaying correctly in select.options on Windows in the 3DSecure UI.
- Verified that built artifacts stayed within scope (no global variables) using abstract syntax trees for testing.

Mobile Card Information Input Libraries

Prepared libraries for Swift / Kotlin / React Native + Expo / Flutter. Contributed by supporting and reviewing as a mobile-exclusive engineer.

Server SDKs

Maintained and operated SDKs for Go / Node.js TypeScript / PHP / Perl / Java / Ruby / Python as the only person capable of handling all languages.

- Focused on rich features beyond automatic generation from OpenAPI.
    - Auto-retry on rate limits, automatic publishing via actions, realizing object structures avoiding circular references, etc.
- Paid special attention to Go.
    - Addressed zero-value issues.
    - Faced challenges in representing cases where list structures contain resource structs, without using generics due to version support constraints.
- Published a non-official Rust SDK and OpenAPI v3.1.

Work History: Launching a Media Trading Web Service

Overview

Was approached to assist in launching a service for buying and selling secondary creations (any format such as images, videos, novels, VRM, etc.) that had been in development for two years with a total of 10 people but had not been released. Worked as a part-time contractor alongside my main job to launch the service.

Duration

From October 2019 for 1.5 months

Role and Contributions

As the development leader, I was involved in planning, design consolidation, requirements definition, task creation, and led two other engineers to launch the service in one and a half months.

Details

When I joined, there was only the founder and two university student developers.

After being briefed on the infrastructure and finding no screen transition diagrams, I listened to the requirements and read through the partially developed code on GitLab.

It was a microservices architecture with multiple servers communicating via gRPC, handling media processing (resizing images and videos, storing them in storage, etc.), APIs, dispatchers, and more, all written in Go language. The API and UI communicated via GraphQL.

The BE was being developed by a talented student, so I helped him complete the gRPC writing to get it running, then left the API server (GraphQL) and other parts to him.

Since the FE was less experienced, we shared tasks focusing on design, and I gradually helped him learn Vue and GraphQL.

I focused on creating pages: multimedia upload forms, PayPal payment and account registration forms, search functions, login, etc.

We were gathering at Yahoo! Japan's open Lodge at the time, so I joined after my main job, and after closing, we developed at a family restaurant. We released at 3 AM a month later at Joyfull Akasaka, and I still remember the joy of toasting.

Work History: Cross-Departmental Language (Node.js) Support Activities

Affiliation and Duration

Worked as a full-time employee at a major IT company, concurrently holding this position from 2018 until my resignation in 2019.

Overview

With Node.js (version 10 at the time) adopted as a recommended language within the company and the arrival of a Node.js committer, a team was formed to support Node.js development across departments. As a concurrent role, I engaged in activities such as user training sessions, individual consultations for department-specific issues, external PR, and code reading sessions to enhance the team's capabilities.

Activity Achievements

User Training Sessions and Consultations

Instead of covering topics like building an API server with Express, we focused on the four core foundational technologies of Node.js: Modules, Buffer, Events, and Stream. The program was designed to help participants become self-sufficient in problem-solving by deeply understanding these areas (3 hours × 4 sessions, about 20 participants per session). The original materials and problem sets were well-received in post-session surveys, allowing us to run the program multiple times.

In consultations, we addressed issues like performance and scaling in a single-threaded environment (carefully explaining that these could be resolved through design), packet loss and memory leaks after switching from PHP (investigated with tcpdump to explain that Node.js itself wasn't the cause), and more.

External PR:

- Presented at NodeFest 2018: https://nodefest.jp/2018/schedule.html#conference-2-3
- Yahoo! Frontend Advent Calendar 2018: https://techblog.yahoo.co.jp/advent-calendar-2018/yahoo-frontend/
- Interview on en-ambi: https://en-ambi.com/itcontents/entry/2019/08/08/103000/

Code Reading Sessions

We read through Node.js code to create the above materials.

- Particularly focused on understanding the event loop by studying libuv and V8 under the guidance of a committer, which was immensely valuable.
- Later, to cover communication and encryption for the second training session, we studied foundational technologies like TLS communication, HTTP2, and encryption based on Node.js implementations. This knowledge has been widely applicable in my work, making it the period when I grew the most as an engineer.

Reference: https://techblog.yahoo.co.jp/entry/2020072830014370/

Work History: Designing and Developing a DMP (Data Management Platform)

Affiliation and Duration

Worked as a full-time employee at a major ad distribution company (also with its own media) from 2016 until my resignation in 2019.

Challenges

In the sale of targeted advertising, as competitors increasingly utilized data, there was a desire to create a BI tool that would empower our sales team and agencies to make stronger proposals to advertisers.

We had a vast amount of data (over 1 TB compressed in HDFS for one day's worth of media access logs, over 2 billion PVs), managed via Hadoop, but when asked questions like "We want to raise awareness of our product among users of competitor X. How much reach will we have, and what are the demographics and usual behavior patterns of those users?", running MapReduce repeatedly in real-time was too time-consuming.

Additionally, there was a demand to analyze data combined with big data from advertisers.

Solution

Developed a tool that could interface with third-party data, return results in seconds, and visualize them richly, using Hive + Solr and React + D3.

Example:

An advertiser wants to deliver ads to a user list obtained by combining their "own user list A" with "user group B interested in competitors (searches, BT)" from our data, specifically targeting B and not A.

Before delivery, they can instantly visualize reach numbers and demographic information, and can adjust reach by extending the data period or adding other segments. Conversely, they can narrow down to female users in Tokyo.

Role in Development

In a project with 20 engineers, I developed six components (microservices).

Responsible for requirements definition and system design (initially mainly external design of some components, but later took on the grand design of the entire system), and handled

development, testing, and maintenance of two components (a service that performs ETL and a UI server for visualization). Acted as PM for the ETL component.

Achievements

Design and Operations

- Collaborated extensively to ensure the overall component diagram was loosely coupled without single points of failure. Despite being our first attempt at microservices and API orchestration, we completed it successfully.
- As an engineer overseeing the entire system, I identified design issues in each component and facilitated adjustments.
  - Anticipated potential issues by studying the technologies used across the system in advance, leading discussions during overall design, and gaining trust to manage progress and provide assistance during component-specific development. When deadlines were tight, participated in discussions on feature selection with a focus on overall optimization.
- In the backend, we partitioned data into long-term unchanging attribute data (residence, gender, etc.), short-term variable attribute data (devices, interests, etc.), and individual action data (searches, site visits, etc.), depending on the aggregation cycle. In the UI, we focused on abstracting these different data types so users could select them uniformly without being aware of the differences.
- Recognized that third-party data structures could change easily. Changes in column types in Hive tables due to data restructuring could impact how data is selected in the UI. Conversely, limitations on the number of segments or aggregation period constraints in the UI required knowledge of Hive or Solr capacity and performance to set appropriately. As responsibilities were divided by microservices, coordination between data ingestion and UI teams was necessary. As someone who understood the entire system, I facilitated communication and acted as a bridge.

Development

From PHP to Node.js

Encouraged the team, which consisted of engineers with only PHP experience, to develop in Node.js due to the following advantages:

- **No Limit on Session Count**:
  - Reach calculations involve fetching data via APIs to the DWH (Solr). While results generally return in seconds, heavy conditions (e.g., long periods) without caching could take tens of seconds. With a PHP + Apache setup, Apache's PS would occupy requests during this time, limiting the scale of UI connections (MAX_CLIENTS < 32767).
- **Seamless Integration with UI**:
  - Shared validation code between UI and FE.
  - Engineers were responsible for UI operation, which was developed by designers using React/Redux. There was a longstanding issue of engineers not being proficient in JS, posing challenges in UI operation. Learning Node.js and getting accustomed to JS lowered psychological barriers, even if not completely.

- **Support Through Company-Wide Node.js Assistance**:
  - As I was concurrently part of the company-wide Node.js support team, I could respond quickly to issues with libraries or the language itself.
- **Communicating Between UI and Hive/Solr via Custom Expressions**:
  - Created custom expressions (like "B and not A" from the earlier example) using PEG.js (parser generator).
    - Using JS allowed us to reuse parsing and expression string generation between FE and BE.
    - Ensured expressions weren't overly customized and stayed close to Solr queries.

Communication Based on API Specifications Between Services

Introduced API Blueprint for creating API specifications and shared the method across the project.

- Used Markdown to generate specifications and mock servers with Aglio/Drakov.
- Chose API Blueprint over OpenAPI (then Swagger) because PMs who were non-engineers could write and access specifications in Markdown format.
- In hindsight in 2024, API Blueprint has become obsolete, and using Swagger (OpenAPI) specifications would have allowed for longer-term utilization.
- However, at the time, the crucial needs were:
  - Base communication between FE and BE on API specifications.
  - Allow anyone to write and access specifications.
  - Ensure documentation and mock servers were synchronized.
  - We achieved sufficient results with API Blueprint under these requirements.
- There were no plans for external API exposure, and the automatically generated Node.js code from Swagger at the time wasn't very useful.

Challenges Faced

- Handling dates and periods in data was complex, causing difficulties in coordinating between components.
  - Partitions were based on daily dates (generally log_date).
  - In Hive pre-processing during ETL, the date within the log (when the event occurred) and the date when data shaping was completed always differed by at least one day. Complex conditions could take more than two days to generate. Third-party data often contained logs from multiple dates.
  - If the meaning of dates differed, correct results couldn't be obtained. Since the meaning of dates displayed in the UI could vary per data, careful handling was necessary.
  - Recognized the need for a component to centrally manage such metadata and developed an additional component.
- For machine learning purposes, at least a year's worth of data was needed to account for seasonal variations. However, the DWH (used in Solr's in-memory operations) couldn't retain data for such long periods, requiring trial and error.
- **Hive is Not SQL**:

- There were differences in behavior compared to intuitive SQL understanding when parsing queries, significantly impacting performance if one couldn't imagine how queries translated into MapReduce jobs.
- While SQL injection could be prevented with libraries, when converting custom expressions to HQL, one had to be cautious about injections. After release, unescaped '%' in strings passed from the UI led to looser conditions and processing delays (MapReduce didn't finish even after dozens of hours).

Work History: Overhauling a Metadata Generation System for Machine Learning

Affiliation and Duration

Assigned as my first job upon joining a major ad distribution company (also with its own media) as a full-time employee from the end of 2013 to 2015. Continued maintenance and operation until resignation.

Background

While generating user lists for behavioral targeting (BT) ads using machine learning logic based on gradient boosting, we enhanced accuracy by attaching metadata to the logs used for training.

This metadata was being created manually by another department using legacy tools, which needed to be overhauled.

Role and Developed Products

In a project with a PM and one developer (myself), I was responsible for development and operation of the following:

A. UI/FE/BE for creating metadata (linking behavioral content with interests) (e.g., associating the search term "Hawaii travel" with the overseas travel tag, or linking the Nintendo DS ad creative with the games tag)

B. A system that uses MapReduce to lookup action logs with the dictionary from A, performs rule-based modeling, and creates user lists by interest.

Actions Taken and Achievements

Regarding A

- **Migration from Oracle 9i to 12c**:
  - The legacy system was extremely slow, retrieving data via Oracle's Database Link (Materialized View) and performing shaping and processing with PL/SQL.
  - Replaced PL/SQL processing with more readable ANSI SQL or moved it to the language side to enable unit testing.
  - Simultaneously reviewed the DB design to improve speed.
  - Undertook careful redesign and necessary data migration, collaborating closely with the PM after thoroughly understanding the code due to the significant overhaul.
- **Data Migration from Isilon**:

    - Distribution of dictionary data to B was done via Isilon, but mounting was only possible on physical machines, limiting scalability and posing risks if the machine failed. Switched to an in-house S3-compatible storage.
- **Rewrote Perl Batches Lacking Unit Tests**, aligning processing with the new DB design, and prepared unit tests and smoke tests.
- **Rewrote New Functional Aspects Using Python's Django**:
    - Chose Python because many new graduates could use it.
    - Selected Django because django-nose automatically created test tables for each test, allowing SQL-included tests and clean-up after completion.
- **UI Improvements**:
    - Since the UI was a proprietary PHP framework without documentation, created E2E tests with CasperJS before the overhaul.
        - However, it was too time-consuming. Initially, developed test code on Mac using NightmareJS 2.x (Electron), which had intuitive APIs and fast performance, and operated it on an independent CI. When a company-wide CI was introduced and enforced, the only available environments were CentOS 6 or 7, and the then-current Electron couldn't be built, leading to abandoning the E2E code.
        - Rewrote using CasperJS (running on PhantomJS), but soon after, PhantomJS announced end-of-life. At that time, Puppeteer and Playwright didn't exist, and choosing Selenium might have been the next best option.
- **Collaborated with Regional Teams Responsible for Metadata Assignment**, frequently communicating via business trips and video calls to improve the UI. Also conducted data analysis to implement operational improvements.
    - Discovered through interviews that due to historical reasons, they were assigning metadata in order of ad submission. As a result, many new ads lacked metadata.
    - Since ad creatives and video views had the most weight in BT learning impact, I conducted data analysis alongside development. Found that including ads with high click counts significantly improved final effectiveness (predicted increase in reach). Proposed and implemented a feature to prioritize tasks based on click counts, which was adopted.

Regarding B

- **Added New Ad Data to Increase Training Data**, coordinating with data providers for new data ingestion, parsing new logs in MapReduce, adding rule-based modeling in machine learning, and conducting numerical verification.
    - Migrated from Hadoop processing to Hive (OnTez) for speed improvements and resource reduction.
    - Implemented Hive commands with Oozie to leverage Oozie coordinator's rerun functionality during operations.
    - Implemented HDFS upload processing and HDFS to S3 transfer processing using HttpFS commands (REST API).

(end)