Boston Office:
1 Harris Street
Unit 7
Newburyport, MA 01950

Little Rock Office:
401 Main Street, Suite 203
North Little Rock, AR 72114

Objectives and Challenges

Our client, a multi-state online and physical cannabis retailer, had several technical and business process challenges. The cannabis market mandates systems which, by law, must be used for specific portions of the seed to sale process. There is no consistency between all states; states require using specific seed-to-sale vendors and states also mandate different stages of the process be reported to their systems. This presented a challenge for operating their business consistently across all states and having a business-wide view of the state of their cultivation, manufacturing, production, orders, customers, wholesale, delivery, and marketing efforts.

Products

Following state regulations, cannabis products can only be cultivated, manufactured, and sold (or destroyed) within the same state. State systems must be updated regularly with product information (name, product, THC/CBD levels, lab results, etc.) and inventory levels. Our client cultivates and produces the same and/or similar products in different states

Customers

Requirements around customer information vary from state to state. Some states have mandated systems for storing customer info for use within the state. The business must be able to identify a customer who shops in multiple states for loyalty, customer service, and marketing purposes. Some states mandate anonymity for recreational customers such that there is no storage of individually identifying information. Since Medical customers are required to register with the state, and maintain an active recommendation from their doctor, there is more information available for these individuals. States still require different storage and transportation requirements for customer specific data.

Orders

When a reservation is placed, to meet all the different state and business needs, a chain of business actions needs to happen:

- Check available inventory levels
- update inventory levels
- report purchase details to the state
- trigger loyalty achievements
- update demand planning systems
- update corporate finance systems
- update executive business dashboards

Due to the state-to-state requirements and variances, there are multiple inbound order systems that need to be accounted for.

Boston Office:
1 Harris Street
Unit 7
Newburyport, MA 01950

Little Rock Office:
401 Main Street, Suite 203
North Little Rock, AR 72114

Client Implementation

To operate their business given the demanding technical logistics, they had variety of different shims in place to connect systems together. They were using an ERP system, a data transformation service, google sheets with JavaScript, MS Excel, and a host of other tools to operate day by day. In some cases, there were vendor to vendor implementations to exchange data, and in other cases, data came through our client's systems. Their systems required a large amount of human capital to run and manage and had an enormous amount of risk for errors.

Goal State

The Elyxor response to this challenge was to design and implement a Service Layer built on a message bus to handle and manage inbound and outbound data. Systems of records for different business objects could be identified, and this data would then be distributed to other systems which needed it. Disparate 3rd party systems used by the business, which could not be consolidated, would communicate with the Service Layer, to send or receive information.

Inbound data would be cleansed and validated and stored in a unified internal format, marked by the system of record where it was sourced or generated. Adapters could be implemented to transform that data into appropriate formats for consuming systems. Data would be made available in a Data Lake for analytics and reporting.

The system would be event-driven using reactive development patterns so that it would be efficient, performant, with near-real-time data updates. Data would be pushed to consumers as soon as it was validated and became available. APIs would be created for consuming systems with no push capabilities, so that they could consume data on a configurable interval.

Elyxor Approach

The Service Layer is a net-new system using a microservice architecture deployed as containers within Azure. The summary of the project approach is shown in the following sections.

SDLC

The project used agile practices and 3-week sprints. The team consisted of several developers, a Scrum Master/Tech Lead, a process analyst, and two DevOps engineers. The project was organized around first getting the SDLC working, the infrastructure stood up, and finally tackling the individual domain models (Locations, Customers, Inventory, Orders, etc).
The DevOps engineers set up GitHub pipelines for provisioning Azure resources and infrastructure using Terraform. They also setup pipelines to build and deploy the Kotlin/Java microservices.

elyxor

Boston Office:
1 Harris Street
Unit 7
Newburyport, MA 01950

Little Rock Office:
401 Main Street, Suite 203
North Little Rock, AR 72114

Domain Model Analysis

Work was completed by the process analyst to work with the business and client's existing technical team to understand what the sources of data were and how this data could be grouped into domain objects. This included a deep analysis of the current processes, some of which were very poorly understood and documented. The output of this provided a view of the data sources and consumers. For each domain model, it was required to understand how many systems of record produced new data, what the data format was and to map the data to the client's unified format for that domain model, and what systems needed to consume new, updated, or deleted data when the systems of record provided updates. This analysis yielded the prioritization of the microservice development.

Infrastructure

Using the standard "infrastructure as code" pattern, the team configured Terraform Cloud, connected to GitHub and linked to Azure to provide pipelines for provisioning networks, subnets, application gateways, storage, and the other infrastructure needs of the Service Layer. The initial rollout had 4 environments, dev (CICD), qa, stage, and production, which were reflected in Terraform Cloud as a workspace per environment.

The dev environment was setup to use CICD, so that when build pipelines pushed new or updated containers to DockerHub, the dev environment would be refreshed with the new images.

Azure Key Vault was used to store secrets.

Microservices

Each domain model had a set of microservices built as Verticles using the Vertx framework. Communication between the microservices was message based and could be implemented over RabbitMQ or using API calls. The Service Layer microservices all used RabbitMQ messaging for communication, with API verticles wrapping the message calls to provide access to client system which couldn't use RabbitMQ.

Low-volume microservcies all ran as verticles in a single JVM in a single container instance. For the services that had higher load or performance requirements, verticles could be cloned and moved into any number of container configurations to meet scale requirements. For example, inbound order information, especially at peak times of the day, were implemented as a cluster of order validator verticles to guarantee they were processed quickly.

The microservice design is flexible and allows the team to quickly adjust the footprint of the system to meet the changing characteristics of the data generate by the business and its customers.

elyxor

Boston Office:
1 Harris Street
Unit 7
Newburyport, MA 01950

Little Rock Office:
401 Main Street, Suite 203
North Little Rock, AR 72114

Monitoring

The microservices used the Micrometer library to produce metrics and publish them to an InfluxDB Cloud instance. Dashboards were created for each microservice to monitor its performance. Higher-level monitoring dashboards were also published to provide system and business insights. These did not replace the business reporting used by the business, rather, it augmented these reporting tools to provide valuable lower-level detail for performance monitoring and for proactive troubleshooting.

Tech Stack

- Kotlin / Java
- Vertx
- Maven
- InfluxDB Cloud
- Micrometer
- Azure
    - MsSql Server
    - Container Groups
    - Kubernetes Service (AKS)
    - App Gateway
    - Key Vault
    - Storage
    - Networking
    - ActiveDirectory
    - DataFactory
- Terraform
- GitHub
- Locust.io
- Docker
    - DockerHub
    - Docker-Compose
- OpenAPI 3 (OAS)
- RabbitMQ
- SalesForce API
- Microsoft Dynamics 365 (NAV)