APPENDIX B
Taco Bell's Serverless Journey

With contributions from Vadim Parizher, Vice President of Technology at Taco Bell, and Robbie Kohler, Sr. Director of Software Engineering at Taco Bell

Taco Bell is a quick service restaurant (QSR) brand selling craveable Mexican-inspired food to its loyal fans around the world. As of November 2023, Taco Bell has over 7,200 restaurants that serve more than 40 million customers each week in the US. The company's technical challenges are unique, as the order point can be either in or outside the restaurant. In the restaurant, orders may be taken by employees at a point of sale (POS) or entered by customers through self-service kiosks. Outside of the restaurant, orders originate from sources such as Taco Bell's own website and mobile applications, as well as third-party delivery marketplaces.

Additionally, each Taco Bell restaurant operates in a highly dynamic environment where continuous updates are necessary for maintaining timeliness and accuracy for customers. These updates encompass menu variations, restaurant operating hours, product availability, pricing, order updates, promotions, and loyalty rewards. On top of that, order volumes can spike considerably because of a marketing campaign or a social media post, which is great for the brand but can be a challenge for technology teams. The customers may not realize it, but it takes a lot of computing resources to serve each taco or chalupa exactly as ordered!

Taco Bell's digital transformation began in early 2018. This case study describes the serverless development of the company's Menu Middleware and Order Middleware and the successful migration of its legacy ecommerce system from a monolith to a modern microservices architecture.

Before Serverless

The following are three business domains that have been transformed by serverless technology at Taco Bell:

Multichannel menu integration (Menu Middleware)

Near the beginning of Taco Bell's digital transformation, there was an immediate need to build a system to store and integrate menu data with internal and external partners. This data consists of restaurant information such as store hours and locations and product information such as categories, customization information, pricing, and taxes. Maintaining accurate pricing data was particularly challenging, with 7,000+ US restaurants and thousands of products and modifiers, requiring hundreds of millions of unique pricing rules that needed to be accurately synchronized across internal and external systems. On top of that, the menus were constantly being updated as restaurant-specific details changed. To add to these challenges, the engineering team was presented with a time constraint: the system needed to be built as fast as possible, with the goal of fully integrating the entire system of menus with a major delivery provider by mid-2018.

Meanwhile, requirements gathering had already started for the second major integration channel, which would go live later in 2018, providing menus for the kiosk program. Vadim Parizher, who joined Taco Bell in late 2017 to lead the company's technology transformation, remembers: "It was clear that building the Menu Middleware would be the first building block on our digital journey. It was going to be key to unlocking a successful digital transformation."

Multichannel order and POS integration (Order Middleware)

In March 2020, the emergence of COVID-19 sent shockwaves across the globe, significantly impacting various industries—including the QSR business. With the onset of the pandemic, QSRs saw an immediate shift in consumer preferences from in-store or drive-thru pickup to the convenience and safety of delivery. To meet this sudden demand, many of Taco Bell's restaurants started participating in delivery by using aggregator-provided smart tablets. Unfortunately, the restaurants soon saw major drawbacks to using the tablets. Customer orders being transmitted were often missed by restaurant staff during busy times, and the manual re-entry of orders into the POS from a tablet led to a higher level of order inaccuracy.

The engineering team now faced another time-sensitive challenge: to help the company and its franchisees make it through the pandemic, they needed to build a system that would integrate orders from the major delivery aggregators into their POS systems as quickly as possible. Parizher recalls, "We had to act quickly to help our franchisees and serve our customers. The tablet solution was only

a temporary stopgap. We had to build a fully integrated solution direct to the POS to keep the business running." With the challenge before them, they set out to build a custom multichannel order and POS integration platform, the Order Middleware.

Ecommerce (Tacobell.com and mobile apps)
: In October 2014, Taco Bell became the first QSR brand to launch a mobile app that enabled customers to customize and order their favorite items and pick them up in-store or at the drive-thru. Parizher states, "It was a very strategic move to demonstrate that Taco Bell was investing in technology, and we wanted to show the world that we were the first to have a functional mobile app. It was a statement."

  In early 2018, the next major version of the ecommerce system was released, which was a complete rewrite of the commerce backend and the client frontends (web, iOS, and Android). In 2020, the mobile app added Taco Bell Rewards, allowing customers to earn and redeem points for free items. Today, US customers can also participate in Taco Bell Delivery directly from the mobile app, save their favorite items and locations, and even enjoy unique digital experiences such as loyalty voting (customers can vote for their favorite products to come back on the menu) and Coin Drop (a digitization of a countertop skill game in which proceeds go to the Taco Bell Foundation).

The Reasons for the Change

This section describes the motivation for migrating Taco Bell's ecommerce system (the website and mobile apps) to serverless. Both the Menu Middleware and Order Middleware projects were greenfield, so the serverless journey for these domains will be discussed in the next section.

According to Robbie Kohler, who became Director of Ecommerce Engineering at Taco Bell in late 2020, the legacy ecommerce system was a Java-based monolith with a single database, which presented a multitude of challenges as the company's digital business started growing. "We tried to tune the monolith as much as we could by scaling horizontally with read replicas and adding more and more application nodes, but we were never fully comfortable when we had a large promotion coming up," he stated.

With the monolith, Taco Bell was not immune to the usual list of problems that a system like that imposed, including:

Scaling woes
: High-traffic events, such as loyalty promotions or free taco giveaways, caused system slowdowns and downtime. Scaling in preparation for events was a manual and error-prone task.

Operational costs
: Maintaining and scaling the monolithic application was costly. Servers were run 24/7, even during periods of slow traffic. Oversized databases were required to handle the unpredictable traffic patterns.

Slow innovation
: Despite efforts to maintain agility, the excessively slow build processes resulted in a development lifecycle resembling a waterfall approach within sprint cycles. There were over a million lines of complex and brittle code, with many complicated layers of abstraction and dependencies. Only a few specialists really understood the code.

Single points of failure
: If the database experienced significant latency or if a job server went offline, it could trigger a complete system-wide outage. The team became hesitant to make changes and often refrained from doing so to prevent potential service disruptions.

Onboarding/development
: It would take weeks, or sometimes months, to set up a new developer on the monolithic system. Setting up a new environment took just as long, with many infrastructure changes and much DevOps work required before it was usable.

With Taco Bell's parent company, Yum! Brands, building a headless commerce API that would handle core services such as user authentication, payment processing, and cart and tax calculations, Taco Bell decided it was the perfect time to migrate the entire monolith to a modern microservices platform, starting in the middle of 2022. As Kohler described, "We knew we wanted to move toward a MACH architecture (microservices, API-first, cloud native, and headless). We already had a few new microservices that were built outside of the monolith, specifically loyalty, delivery, and favorites, but it was time to go 100% and disconnect the monolith once and for all. We needed the flexibility, the scalability, and the other numerous benefits that would come if we made the full switch."

The Serverless Journey

As a traditionally nontechnical company undergoing a digital transformation, Taco Bell had its fair share of initial constraints. First, the company did not have the luxury of having a large and experienced team of software engineers ready to build new systems. The team was small, and the company intended to grow it in a careful and planned way. Instead of throwing more resources at the problem, they wanted to make sure they were enabling the engineers they had to be as productive as possible. Kohler recalls:

> We knew we had to do a lot with a little, and we had to be scrappy, much like a start-up would be. We wanted to stand on the shoulders of giants, take ideas from others where it made sense, and leverage strategic vendors like AWS as much as possible. The goal was always to solve business problems and not get too caught up with the technical details. Simple was better than complex. Less code was better than more code. We wanted to make it as easy as possible to get real value out the door and into the hands of our customers.

However, as the small engineering team started researching how to build the Menu Middleware, they started with what they knew: using servers. Kohler states, "I was a .NET developer for over a decade at my previous job. OO programming and servers were what I was comfortable with." However, one thing became clear very quickly: deploying software to servers was very complex in a serverful environment, even in AWS. DevOps engineers were required to set up VPCs, security groups, EC2, complex Jenkins pipelines, and numerous other components, all before any business value could be delivered. And more engineers would be needed to ensure the system stayed up and running once the code was deployed. The team was small, and watching server health metrics was not what they wanted to spend their time doing. Kohler recalls, "I remember feeling frustrated trying to tweak VPC and security group settings for over two days to get my connection to an app server working, and we were falling behind with the Menu Middleware project. We had to start coding, and I started looking to see if there was an alternative to the serverful approach."

Enter serverless.

It was around 2018, and Lambda had been out for a few years. Kohler states, "I had a friend who had done some Lambda at this company, but I must admit I didn't understand the concept at first. There had to be servers somewhere, right?" The more Taco Bell's engineering team learned about it, the more they realized the potential. The main motivation for serverless emerged: they wanted to spend the least number of resources possible and still deliver meaningful business value.

The idea of serverless and what it could mean to the business was extremely exciting to the team, and they felt a surge of energy. Kohler remembers, "We were just reading and learning as much as we could about serverless. It was still a little bit new in early 2018, but it really felt like this could be the future of software development." Vadim Parizher was glad to move away from managing servers. He states, "I had managed data centers for years and was happy to get out of that business. Operating servers was not going to differentiate us in any way."

The strategy was set: Taco Bell decided to dip its toes into serverless for a few specific workloads, gain confidence, and then see where it would take them.

Initial Preparations

Because serverless was a very new concept and represented quite a paradigm shift, Taco Bell decided to opt for a conservative approach at first. "Contrary to popular belief, we didn't jump 100% to serverless out the gate," states Kohler.

Proving it out early

Because the Menu Middleware system was the company's first major digital project in early 2018, a lot was at stake to ensure it worked and was delivered on time. Therefore, the middleware system employed some traditional non-serverless components that were familiar to Kohler and Parizher. Kohler states, "We went with a more traditional RDS MySQL database for our backend that would store all the integrated data components. We also first started out using enterprise data integration tools to perform ETL, and we even used stored procedures and a lot of raw SQL to get the data in and out. We weren't comfortable going with the NoSQL approach just yet."

Early on, Kohler saw an element of the architecture that could benefit from serverless: the menu ingestion functionality, which they called the menu exporter. "We wrote the first iteration of the menu exporter as a .NET console app. It was very heavy and included all the functionality required to map from source to target, and ingest the menu with a delivery integrator," Kohler recalls. They knew that they would be adding more channels over time and that this piece of functionality would have to scale. They decided to port their code as a monolithic .NET Lambda function and see how it would work.

Frameworks like SAM and the Serverless Framework were not used for the first few .NET functions (the team admits it was a bit "wild west" at the time). The code was simply zipped and uploaded to the AWS console. Obviously, this was not ideal, but in true start-up fashion, they decided to go to production and see how it would play out. Initial observability and alarms were based on simple text-based CloudWatch logs from the Lambda functions, as well as CloudWatch metrics. The results were promising, as the serverless components (SNS, SQS, S3, and Lambda) scaled with no manual efforts required. The components that gave the team trouble were the RDS databases and the traditional EC2 compute instances that were used for the data integration. This was the first time they saw the benefits of leveraging the managed services of AWS to move faster and worry less about non-business-differentiating components.

Iterating from there

The initial success of the Menu Middleware gave a burst of energy to the engineering team. In true Agile fashion, they decided to iterate and improve on how they were doing serverless. Kohler states, "The first thing we noticed is that not a lot of people

were using .NET with serverless, so the community support was lagging. We decided to start using JavaScript or Python from then on. We did keep the original .NET code, and they are still running it today." They started using the Serverless Framework with their JavaScript and Python projects, and saw immediate benefits from using a proper IaC framework. The team started adopting best practices and patterns from AWS Serverless Heroes, such as Yan Cui and even one of the authors of this book, Sheen Brisals.

Scaling serverless

A grassroots movement began at Taco Bell, and all the technology teams started to build their own serverless services to solve their respective business problems. A serverless community of practice was formed, and Taco Bell officially became a "serverless-first" tech organization. Parizher recalls, "Robbie and I debated using DocumentDB or DynamoDB for a new application that had to be built. I simply asked if it would make life easier if we were all serverless? The answer was a resounding yes, and we haven't looked back since. Being serverless-first was the strategy from that point on."

Architecture and Patterns

This section provides an overview of the architecture and patterns used in the three main components.

Menu Middleware

The purpose of the Menu Middleware, as shown in Figure B-1, is to accurately and quickly integrate complex menu data with different channels (POS, kiosks, delivery aggregators, AI vendors, digital menu boards).

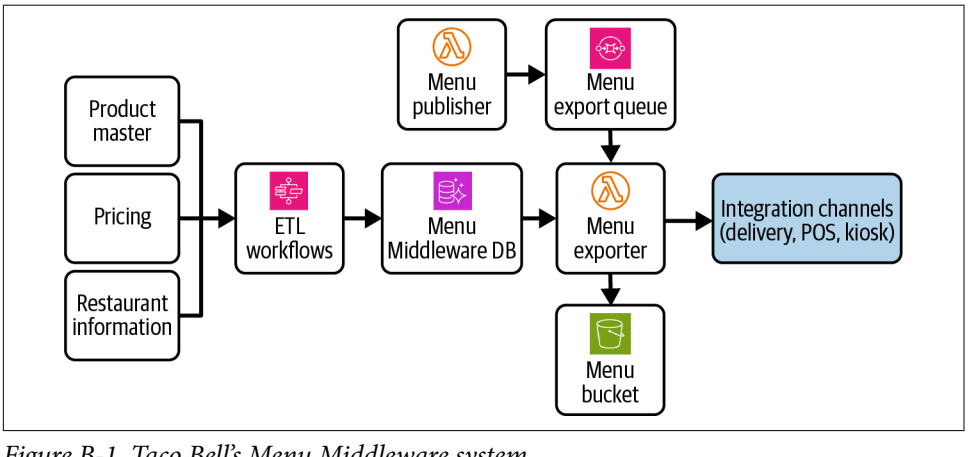


Figure B-1. Taco Bell's Menu Middleware system

It has the following features:

- Fully serverless compute (the last EC2s were decommissioned in 2022)
- Serverless Framework for IaC
- Lumigo and CloudWatch for observability
- Parameter Store for configuration and secrets management
- An RDS MySQL database containing:
  - — Pricing rules
  - — Restaurant information, such as store hours and addresses
  - — Product master information (images, names, types, product attributes)
  - — Product availability rules
- A custom Python-based data integration framework called "Diablo" built on AWS Step Functions, utilized for ETL into the Menu Middleware DB
- .NET, Python, and TypeScript Lambda functions for compute
- S3 for storing menu data

Order Middleware

The purpose of the Order Middleware, as shown in Figure B-2, is to integrate digital off-premise ordering sources, such as delivery integrators, into the company's POS systems.

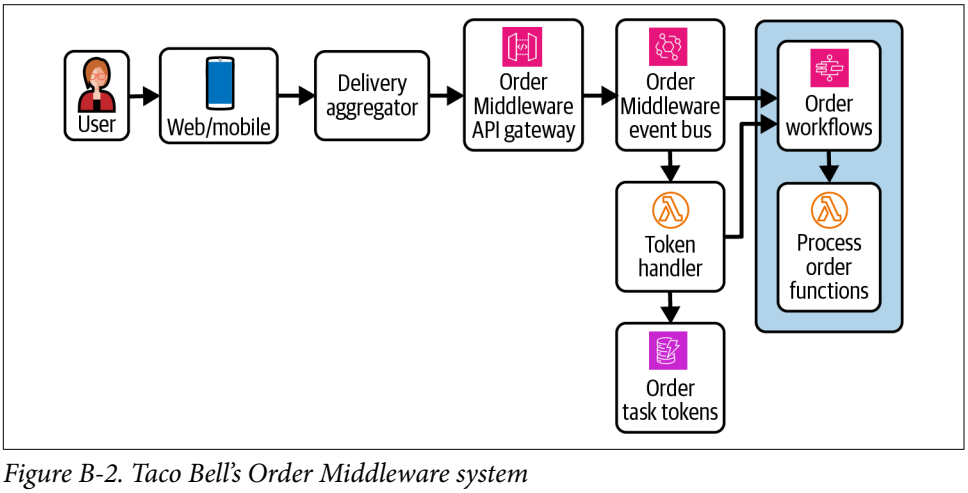


Figure B-2. Taco Bell's Order Middleware system

It has the following features:

- 100% serverless, VPC-less
- Serverless Framework for IaC
- Secrets Manager for secrets
- Middy Middleware and Lambda Powertools for TypeScript for cross-cutting concerns such as secrets/parameter caching, JSON structured logging
- Lumigo and CloudWatch for observability
- Step Functions for orchestration and EventBridge for choreographing events:
  - Express Step Functions workflows are nested in standard workflows for processing short, synchronous flows.
  - When a driver is close to the store, an event is sent to EventBridge so that the order can be released (using the callback task token pattern in Step Functions, discussed in "Distributed Orchestration" in Chapter 5) and the food can be made.
- DynamoDB for order info and task token persistence
- Caching with content delivery network (CDN) and Lambda function memory
- Lambda functions written in TypeScript for compute, using a simplified hexagonal pattern (see Chapter 3):
  - Core domain logic is encapsulated for easy unit testing.
  - Port/adapter logic is built into the Lambda functions.
  - Integration testing is done in live AWS environments.

Ecommerce system

The high-level architecture of Taco Bell's ecommerce system is shown in Figure B-3.

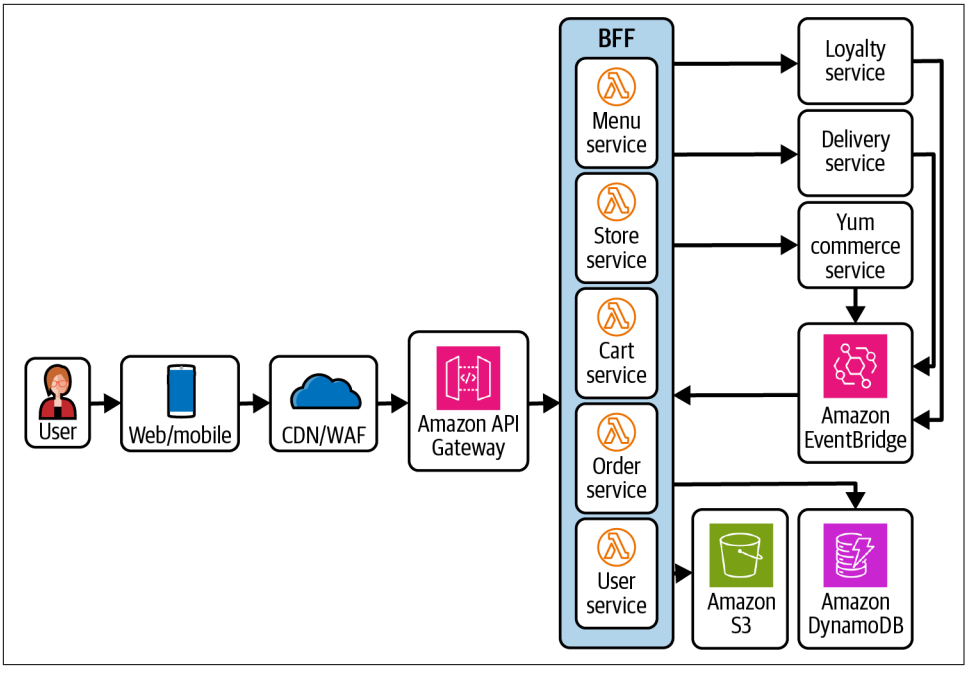


Figure B-3. Taco Bell's high-level ecommerce architecture

It has the following features:

- 100% serverless
- Serverless Framework for IaC
- MACH architecture:
  - — Serverless BFF provides flexible abstraction between frontends and APIs.
  - — Yum! Brands commerce engine provides core commerce functionality.
  - — Other best-of-breed products can be plugged in to provide the optimal customer experience.
- Event-driven—new business functionally can be extended by adding rules to EventBridge commerce bus
- TypeScript Lambda functions for compute, using a simplified hexagonal pattern:
  - — Core domain logic encapsulated for easy unit testing.
  - — Port/adapter logic is built into the Lambda functions.
  - — Integration testing is done on live AWS environments.
- Lumigo and CloudWatch for observability
- Caching with CDN and Lambda function memory

- AWS services used: S3, SQS, EventBridge, API Gateway, Lambda, DynamoDB, Route53, Secrets Manager, Step Functions
- Transition from monolith to serverless followed a strangler fig pattern with vertical slices to reduce the risk:
  - First moved over store microservice (store search, store details)
  - Then, menu microservice (menu and PDP API calls)
  - Then, cart/order microservice (checkout and payment, order submission and tracking)
  - Finally, user microservice (order details and login: passwordless, profile info, saved payments)



Here are some additional resources you can consult for further information on this case study:

- 2021 Re-invent: "Building Next-Gen Applications with Event-Driven Architectures"
- "This Is My Architecture": Order Middleware
- "This Is My Architecture": Menu Middleware

Development Process and Team Structure

The following development processes and structures apply to most of the technology teams at Taco Bell, with some levels of variation to be expected:

Team setup

- Two-week sprints
- Agile/scrum
- Two-pizza teams, cross-functional and multiplatform (e.g., the ecommerce ordering team includes serverless, web, iOS, Android, and QA engineers, and a product manager)
- Cloud platform engineering team:
  - Works on standardizing cross-cutting concerns for developers
  - Builds shared libraries (logging, caching)
  - Ensures security best practices are being addressed and are built into new solutions
  - Audits existing serverless services and provides suggestions and feedback

Guardrails and principles

- Prefer TypeScript/JavaScript and Python for data-specific work.
- Less code is better: for example, direct integrations are preferred unless more custom observability is required.
- Architecture review with a senior serverless architect is required for critical functionality.
- Security review is required before new services are released.
- Testing requirements include:
  - Unit tests for business-critical functionality
  - Integration tests against live AWS services, mock third-party services
  - Performance/load tests for high-scale workloads
- Deployment checklists:
  - Secrets are deployed.
  - If multi-Region, make sure deployment occurred in both Regions and health checking is working.
  - Deployment frequency is up to individual teams, but some teams deploy multiple times per week.
  - All non-production environments are in a single non-prod AWS cloud account per domain/team. Production is in a separate AWS cloud account per domain/team.

The Outcome of Serverless Adoption

Here's a recap of the outcome of the serverless adoption project for each service:

Menu Middleware

- Able to meet the deadline in mid-2018 to integrate with the major delivery provider due to serverless adoption. The team consisted of only two engineers.
- As of November 2023, the system had scaled to support over 12 different integration channels, integrating tens of thousands of menus a day in real time.
- The team started out with a hybrid model of serverful and serverless components. Once they saw success with serverless, the system was completely migrated to serverless over the course of several years.

Order Middleware

- Within two months of starting the project it was live in production, integrating customer orders directly from the delivery marketplace to the POS. In the

coming months, three more delivery providers were integrated into the Order Middleware platform.

- The system is capable of processing millions of orders per day without scaling issues.
- The system is cost-effective and predictable in cost forecasting, as it's easy to calculate the total cost per order (something that would have been near-impossible to do accurately using a serverful approach).

Ecommerce migration

- In less than a year, a small team of five engineers migrated the entire backend to the serverless architecture without impacting existing operations.
- The team saw a 95% decrease in lines of code, from roughly one million lines of source and configuration code in the monolith to less than 50K lines in serverless.
- The team saw a 90%+ reduction in AWS costs going from serverful to serverless.
- Post-migration, releases to production occur multiple times per week; before, they averaged around once a month.
- The team can get a new developer productive on a serverless service in less than one hour. With the previous system, this would have taken multiple weeks or even months.
- The new event-driven approach allows new services to be added seamlessly without impacting existing code.
- High-traffic promotional campaigns are handled effortlessly with the serverless architecture, with no preparation needed.

What Were the Pain Points and Learnings?

Here are some of the challenges that the team at Taco Bell faced along the way, and some of the key lessons learned:

Monolithic functions

One of the earlier mistakes that Taco Bell made on its serverless journey was creating large, monolithic Lambda functions that did too much. According to Robbie Kohler:

> My background was in .NET shops writing complex OO software, so naturally, I brought many of these same ideas into my first Lambda functions. Soon after, we realized that the best way to write a Lambda function was to apply the single-responsibility principle and to create smaller, more focused functions that handle a single responsibility or task.

The team also started to strategically use Step Functions for orchestrating complex business logic instead of relying on Lambda functions to do this. "There are obviously trade-offs to using Step Functions, but we started to see the benefits and flexibility of this design with our Order Middleware platform. In addition, many of the Lambda functions could be reused across different integrations, which reduced the need for duplicative code," according to Kohler.

Understanding of concurrency

Once Taco Bell's engineers started doing more serverless, they saw a few things appear in production that they weren't prepared for. The importance of an understanding of Lambda concurrency and how to configure reserved concurrency was a lesson they had to learn the hard way. As they scaled, they saw instances of throttling or hitting account limits for the number of concurrent Lambda functions running simultaneously. From there, they realized that things like setting timeouts appropriately were very important. Kohler states, "The default setting is 1,000 Lambda functions running simultaneously in an account, which seems like a lot. However, once teams started adding more functions, some of which could be synchronous and last more than a few seconds, we started seeing throttling and problems. We had to raise the concurrency limit a few times."

Orchestration versus choreography

When Taco Bell's engineers were first designing the Order Middleware in 2020, they were very excited about EventBridge, which had been released a few months earlier (in late 2019). Kohler remembers, "We were probably a little too excited to use it. We thought the Order Middleware, being naturally event-driven, would be something that should be completely powered by EventBridge." The first iteration of the design had complex delivery workflows powered by EventBridge rules. When a Lambda function would finish, it would send another event back into EventBridge to execute the next step. Very quickly, the team realized that this approach was getting complicated and becoming difficult to debug and troubleshoot.

Kohler and his team met with Serverless Solution Architects from AWS, who recommended that they consider Step Functions to orchestrate the business processes within their order domain and use EventBridge to choreograph events from outside of the domain. The second iteration of the architecture used a combination of EventBridge and Step Functions (including express workflows nested in the longer-running standard workflows), and the system immediately became much easier to understand. The visual representation of the business process was very handy as well, especially when explaining the system to nontechnical team members. In addition, the team found that patterns such as the callback task were a natural and intuitive way to handle pauses in the workflow as they waited for external delivery events from delivery aggregators.

They found Yan Cui's article "Choreography vs Orchestration in the Land of Serverless" to be very helpful in understanding the differences between the two.

Five Pieces of Advice to New Serverless Adopters

Asked what advice they would offer to enterprises starting out in their serverless adoption journeys, Robbie Kohler and Vadim Parizher offered the following tips.

Join the Serverless Community

One of Taco Bell's earlier strategies that paid off quickly was to reach out to the tight-knit serverless community and work directly with those who had done it before. Kohler states, "I really think the best way to start off right is to seek guidance and inspiration from experienced serverless practitioners. Learning from experts like Yan Cui and Alex Debrie can provide insights into best practices, common pitfalls, and real-world solutions. Learning from their experiences can help you avoid mistakes and accelerate your serverless journey. There is no shame in standing on the shoulders of giants!"

For Taco Bell, the serverless community has been incredibly friendly and helpful, and interacting with it has been one of their favorite parts about being a serverless-first dev shop. And as you're reading this book, you are following this advice already!

Embrace an Iterative Mindset of "Good Enough"

Taco Bell aims to limit FUD (fear, uncertainty, doubt) as much as possible, so each engineer feels confident and enabled to be their best self. One way it does that is by embracing an iterative approach of "good enough." The team favors releasing features when they are good enough, as trying to be 100% perfect is impossible. The bar is always moving, so you can spend an infinite amount of time chasing an unattainable goal. It's better to adopt a mindset up front to move quickly, adapt, and either succeed or fail as fast as possible. Taco Bell feels that being a serverless-first dev shop fits with this rule perfectly, as it is quite easy to get to the point where something is in a good enough state to push it to production and start seeing results—and, most importantly, gain confidence and momentum. Then, as the team learns more while the feature is live, they can iterate and work on the next goal. According to Kohler:

> With serverless, you've already removed so many operational and infrastructure concerns from the picture that you can truly focus on attacking the business problem. So, we always try to get to "good" as fast as possible and let the momentum carry us! Of course, as part of any release, we make sure we meet the minimum quality standards of our serverless services. Our Cloud Platform Engineering team is devoted to ensuring best practices for security, cost optimization, and scalability are baked into templates and starter repos, which allows our developers to just worry about solving business problems. Bonus tip: start a Cloud Platform Engineering team!

Avoid Big Bang Migrations

Taco Bell's engineers feel that a major reason they were successful in migrating their massively complex ecommerce monolith to serverless was that they avoided a "big bang" migration and instead adopted a more iterative migration approach to de-risk the project as much as possible. "The ecommerce app is a central piece to the digital experience for our customers, so ensuring a safe and seamless launch was paramount," states Parizher.

They broke up the migration into four major vertical slices: the store, menu, cart/order, and user microservices. While the entire monolith was still running, they tackled the first slice, which was the store service. They kept the existing data contracts for each API to limit any changes to the frontend systems, using feature flags as an emergency switch if needed to route back to the existing system. They also created a DynamoDB table to control whether the new serverless API was used based on the store number. This allowed them to test with a small number of stores first before expanding as they gained more confidence. At any point in time, they were able to roll back, greatly minimizing their risk.

Next, they moved on to the menu service. They repeated the same steps, starting with one store, testing and verifying, and then moving on to more. At this point in the migration, users were still logging in using the legacy system, but they were accessing store and menu data from the new serverless backend.

They then moved on to the cart and order services, which had the most risk in terms of impacting revenue. They had to ensure their customers could still place orders, without knowing the backend ecommerce system was changing. This meant they needed to make sure prices, taxes, delivery fees, loyalty promos and discounts, and anything else related to financials was being calculated correctly. They moved forward carefully, building confidence. With serverless, they were able to swiftly address and deploy bug fixes due to the liberation from the constraints of the monolithic architecture.

Finally, the user service, including favorites, saved payments, and authentication information, was migrated over, with minimal disruption to the customers.

Another key aspect of the successful migration was the use of Amazon EventBridge and the move toward event-driven architecture. The team was able to handle user notifications by listening to specific events from the headless commerce system and calling a decoupled user notifier Lambda function. This was a massive improvement over the way they had handled events and user notifications in the monolith, where they were tied together with many other jobs and processes that would fail together at the slightest sign of trouble.

Embrace Simple Over Complex

As mentioned previously, many of the Taco Bell engineers who first started with serverless had backgrounds in building "complex" enterprise object-oriented systems. Kohler states, "Thinking back, I now realize that the countless layers of abstractions and clever programming tricks did nothing but confuse other engineers and were probably done only to serve our egos. Ultimately, what always matters is the business value that is produced, not the lines of code." Taco Bell now adopts the slogan, "Less code is better than more code." For the most part, if they can avoid a Lambda function, they will tend to do so. Employing direct integrations from API Gateway to EventBridge and Step Functions to basically every AWS service has been a time-saver for them—remember, though, that you may lose some flexibility with observability, error handling, and logging.

They also stress simplicity in their patterns. They tend to use a basic design pattern in their Lambda functions where they separate core business logic into easily testable modules, but most of the mapping and adapting is done directly in the functions themselves. The "You Aren't Gonna Need It" (YAGNI) principle is especially true when designing serverless applications; they knew they would most likely never be moving away from DynamoDB back to RDS, so they kept their abstractions as light as possible.

In terms of code, they tend to favor explicit and straightforward over clever. In almost all cases, a simple `for` loop will be easily debuggable and understandable by every developer, and that will be preferable to a clever one-liner that will take a code reviewer several Google searches to unpack.

Remember, there are always trade-offs in software architecture, and serverless is no exception. Choose wisely, but favor simplicity first. That has worked well for Taco Bell.

Go Serverless-First!

The Taco Bell team feels one of the best moves you can make as a company looking for some success in serverless is to try to become serverless-first as soon as possible. According to Kohler, having one foot in and one foot out is OK at first, but once you are serious about it, you should fully commit. "It may seem scary at first, and maybe even like you are locking yourself in. However, we were pleasantly surprised that the opposite was true for us. It felt like it liberated the team."

Parizher states, "It was never our intention to be multicloud. We looked at it in the beginning and realized it would never be feasible. Moving to serverless-first on AWS was somewhat of a calculated risk, but we had enough successes already to move forward with confidence." The decision to focus purely on AWS and serverless immediately removed a lot of the cognitive load of having to consider different

technology stacks and vendors. The whole team felt relieved that they could dig deep and focus on the AWS serverless stack, which has plenty of complexity already. Good things started happening very quickly for them once the decision was made to be serverless-first, and they haven't looked back since.

The last thing to keep in mind is that being serverless-first does not necessarily mean you are serverless-only. Kohler describes what being serverless-first is like for Taco Bell:

> When there is some type of new business functionality or problem to solve, our team will first look at a possible solution with a serverless-first lens. We are biased toward using serverless for good reasons because we know it will allow us to move much faster, produce higher-quality output, easily handle the pivots and requirement changes that are common in Agile development, and be cost-effective to boot. So far, we have seen very few use cases where serverless wasn't the right fit. However, we know that there may be specific workloads where serverless may not make the most sense from a practical standpoint. Perhaps it's not economically viable due to high volume, or the use case won't work quite right using stateless functions, such as long-running jobs. In those rare cases, we would set up a VPC and use serverful components.

That being said, Taco Bell hasn't had to be serverful very much, and they expect the same will be true for most of the companies out there who are considering serverless. So why not be bold? Have fun and go serverless-first!

About the Contributors

Vadim Parizher is an information technology architect and executive who developed his career at several Fortune 500 companies after starting as a software engineer at NASA in the mid-'90s, where he helped develop image processing software used on the Mars rover mission. He is currently VP of technology at Taco Bell, leading the engineering functions.

Robbie Kohler is a serverless advocate who started his technology journey in 2004 and has worked for several Fortune 500 companies and start-ups. He has extensive experience as a hands-on developer and architect, as well as an engineering leader. He is currently senior director of software engineering at Taco Bell, leading ecommerce engineering, cloud platform engineering, and production support engineering.