Creating an integrated plug-and-play supply chain with serverless computing

Microsoft Core Services Engineering (CSE, formerly Microsoft IT) teamed with the Microsoft Devices Supply Chain team (MDSC) to enable digital transformation across the supply chain process. Using Azure PaaS, microservices, and serverless infrastructure, we have created the basis for a scalable, agile solution that gives us end-to-end control over the supply chain. The new solution reduces onboarding for new vendors and partners from weeks to minutes. This has helped to shift the MDSC paradigm to a “plug-and-play” model that results in a dynamic and demand-driven supply chain model.

Supply chain infrastructure at Microsoft

CSE is responsible for end-to-end services that support Microsoft product manufacturing, distribution, sales, and returns. It builds services that are used by component and third-party suppliers, contract manufacturers, logistics partners, retailers, and consumers. The Microsoft supply chain engages with hundreds of vendors who manufacture and distribute Microsoft products around the world. Our partners need to be integrated into our network and communication stream to create a properly functioning supply chain.

Our old solution for integrating partners into MDSC was built on an infrastructure that included approximately 200 on-premises servers handling over 3 million transactions per day. The solution had 15 hubs deployed across several Microsoft datacenters that partner organizations could use to connect with the supply chain. We wanted to fix several issues with the old solution so that MDSC can continue to improve and evolve the supply chain.

- **Onboarding new partners was too slow and expensive.** Adding a new partner to the supply chain was a unique, time- and labor-intensive process. New code needed to be developed for each partner integration and, depending on the partner system, partner users needed to be trained. A team of program managers and developers was typically assigned to each new integration, which involved coordination, connectivity testing, and end-to-end testing, all of which could take months to complete.

- **Volume was pushing the solution beyond its limits.** Our old solution did not scale gracefully. Each increase in capacity required significant development and deployment effort, which meant that increased volume or heavily fluctuating volume wasn’t always handled effectively or promptly.

- **We didn’t have reliable end-to-end visibility.** Our visibility into the supply chain process was limited by our ability to collect telemetry data from partner systems that were either inadequate or unreliable. As a result, our end-to-end supply chain picture was incomplete.

- **Operation costs were high.** The costs to run infrastructure, upgrades, and maintenance were too high. Overall, there was a lack of ability to scale the infrastructure to meet the demand and variable growth.

Moving to serverless Azure infrastructure with microservices

The high-level goal of our new solution was to enable digital transformation within MDSC. We needed our supply chain platform to move at the speed of our business and enable cost savings and improvements, rather than hold us back. We developed a short list of goals for the new supply chain solution:

- **Accelerate development.** We wanted a shorter development cycle so we could integrate new features and fixes faster.

- **Build microservices-based architecture.** We identified microservices as the most appropriate architectural model for achieving a modular, yet fully integrated, solution. The more we individualized components and created them to interact cohesively with the whole, the easier it would be to onboard new partners.

- **Achieve highly elastic scalability.** We needed rapid elasticity. The nature of the online, retail business means that high-volume buying periods—like holidays—can escalate quickly in unpredictable patterns. We needed a solution that can scale automatically and effortlessly to meet demand.
• **Reduce infrastructure and maintenance costs.** Provisioning and maintenance of our old infrastructure was too time-intensive. Disaster recovery planning, operating system maintenance, and infrastructure repairs all took time away from improving the supply chain. We wanted to reduce that overhead with the new solution.

• **Improve telemetry and business visibility.** Our new solution had to give us an end-to-end picture of our business, so we could accurately and effectively find ways to improve and grow the supply chain.

• **Move toward DevOps.** We wanted an agile, business-driven development process. DevOps would give us the opportunity not only to optimize and improve development efforts, but also integrate a customer-first mindset into the development processes.

• **Increased availability and disaster recovery.** We needed high availability for global supply chain integration and we wanted to reduce administrative overhead and improve the speed and automation of disaster recovery.

**Design principles for integration strategy**

Keeping our solution goals in mind, we set about creating the core design principles for an integrated supply chain platform. We knew it needed to be built on world-class foundation architecture using a service-based approach. We expected that Azure PaaS would be the platform technical solution, but we also knew that a comprehensive solution would incorporate the right mix of serverless, PaaS, and IaaS technologies. Our core technical design principles were:

• **Simplify and scale.** Simplify and scale partner integrations by building architecture using a modular and scalable microservices-based model.

• **Build for the future, accommodate the present.** Use the right mix of hybrid solutions until we can move all our workloads to the cloud.

• **Accommodate a diverse partner ecosystem.** Design the solution to do business with partners who have varying degrees of technical capability—from limited to advanced.

• **Build for automated self-provisioning.** Enable self-provisioning and discovery for all services, which will result in a simple, fast onboarding experience.

• **Design for network changes.** Enable consistency, repeatability, and automation to ensure seamless network changes.

• **Use REST-based edge services.** Build externally callable REST-based supply chain services for our partners to call, and require our partners to build REST-based services for Microsoft to call (for partners with technical capability).

• **Separate contract from class.** Keep processing and transport layers mutually exclusive.

• **LiveSite first.** Design for operations, with telemetry and monitoring catered to in every phase. Create business services with high availability and automatic disaster recovery built in.

• **Abstract complexity.** Use the right mix of underlying technologies to build out our business services in the cloud, and provide REST endpoints that abstract the underlying technology for partners to call.

• **Unify authentication and authorization.** Adhere to existing certificate-based authentication for existing EDI/AS2 partners and Azure Active Directory-based authentication for cloud services.

• **Design for options.** Provide mechanisms for Post, Get, and Notify/Pull (or Poke/Get) so that we can enable multiple business processes. Our preferred integration method with partners will be based on process latency considerations, with our preference being Pull or Notify/Pull on the outbound (we notify partners and they get data from us), and the Post option on the inbound (we ask partners to publish data to an endpoint that we set up).

• **Accountability model across partner ecosystem.** Each party within the supply chain is accountable for the data contract and ensures the accuracy, timeliness, and availability of data for the end-to-end business service.

**Business and solution architecture**

One important aspect of offering plug-and-play partner integration and creating a manageable end-to-end solution was to change the way our partners fit into the supply chain. Rather than viewing them as exterior components of our internal solution, we looked at each piece of the supply chain, including partners, as a node—an equally contributing part of a managed whole ecosystem.

By treating each partner as a node, we designed the standards for onboarding and integrating partners into the supply chain. Plug-and-play strategy was a significant driver of our architecture design. We wanted our partners to have a consistent and trusted mechanism to integrate with the supply chain and design their processes around.
Establishing integration layers
We set up five layers for supply chain components:

- **Partners.** This layer holds all the partner systems that Microsoft integrates with and onboards into the supply chain.
- **Marketplaces.** This layer is an interface between partners and the supply chain to trigger workflows and actions.
- **Services edge.** The services edge offers over-arching integration between the marketplaces, partners, and core supply chain services. Third-party integrators are a part of this layer, specifically for retail relationships that involve a large volume of message translation.
- **ERP and processing layer.** Transactional systems, all business processes, and supply chain capabilities are enabled here to process supply chain transactions. This layer has the core supply chain processing components that MDSC manages internally.
- **Data warehouse and analytics:** This layer is responsible for enabling the data processing and analytics to derive business metrics and key performance indicators (KPIs).

Figure 1 shows the overall integration of the supply chain.

![Figure 1. Supply chain integration](image)
Overview of fulfillment and logistics supply chain events
The fulfillment and logistics core processes for this scenario start with:

1. A delivery order issued to a node from Microsoft, along with a corresponding load plan from the freight management vendor (FMV).
2. Upon receipt of the delivery order, the distribution node is responsible for picking, packing, and shipping the goods, and for sending the required delivery status notifications to Microsoft.
3. When the goods are shipped out of the node to the receiving location/entity, a shipment notification is sent to Microsoft.
4. For every stage in the shipment process, the carriers send their milestones, which are received by our FMV and sent to Microsoft.
5. Finally, the goods are received by the intended receiver, and Microsoft receives confirmation of delivery from the carrier.

To enable this scenario, following a service-oriented architecture, the core business services will include: delivery order service, load plan service, shipment notification service, and carrier milestones service. Figure 2 shows the fulfillment and logistics processes.

![Diagram of fulfillment and logistics processes](image)

**Figure 2. Overview of fulfillment and logistics processes**

Enabling plug and play integration with REST
All integrations use REST API standards that help Microsoft and its partners connect to endpoints and exchange data as effortlessly and securely as possible. The standards help us achieve the following goals:

- Implement consistent practices and patterns for all REST endpoints across Microsoft.
- Adhere as closely as possible to accepted REST/HTTP best practices in the industry.
- Make accessing Microsoft services via REST interfaces easy for all application developers.
- Allow service developers to use the past work of other services to implement, test, and document REST endpoints that are defined consistently.
- Allow for partners to use these guidelines for their own REST endpoint design.
Implementing API standards

To achieve our API goals, we used the following design standards in our APIs and gave the same standards to partners who were establishing APIs to connect to the supply chain:

- **Versioning.** All the services in our supply chain will support versioning, and different versions will be created based on the request URL (for example, https://supplychainservices.microsoft.com/v1/orders).
- **JSON Standardization.** Services expose JSON for all the REST calls and follow API JSON standards.
- **OData support and handling collections.** Services expose OData format for getting data and support standard OData filters using hypertext. They will also support filtering, sorting, and pagination wherever applicable.
- **Delta queries.** Services support delta queries and pagination wherever applicable.
- **Long running operations and push notifications.** Clients typically ask for a resource using the basic HTTP GET. If clients want to avoid polling, push notifications are a backup option. Services will implement the webhook model where a notification is sent to a client, and the client requests the current state or a record of changes since their last notification.
- **Exception/error handling.** Exception and error handling is done consistently across different services. The error response will follow OData v4 JSON spec as mentioned in the One API guidelines.
- **Authentication using Azure AD.** Services use an Azure AD token-based authentication mechanism. Onboarded clients will need to generate a valid token and append it to the request header. Tokens expire within a configurable time and need to be regenerated.

Understanding API patterns in the context of supply chain

The proposed architecture focuses on reducing the dependency on partner service availability by using POST API for any inbound requests, while using GET APIs for outbound data or data retrieval by partners. This drastically reduces the complexity of partner configuration and maintenance and minimizes the retry mechanisms needed for partner service downtimes. The service is made up of the following layers:

- **Experience layer.** This layer consists of two components that work together to make the integration process for developers as simple as possible.
  - The API management gateway gives a rich developer experience by exposing the service catalog, the API contracts, and an easy programming language. The gateway can also be used for authentication.
  - The second component is the REST API which is responsible for authentication, authorization, data validation, and importing data into the repository layer.
- **Repository layer.** This layer enables the persistence of the payload, any persistent or transient metadata, and the cache. This high performing layer is enabled via system APIs native to the respective Azure resources.
- **Process layer.** This layer is responsible for processing the data, transforming, and orchestrating between the internal APIs of the microservice and managing processes across external APIs and applications.
All these APIs are implemented with Azure API Apps, Azure functions, and Logic Apps based on capability needed, latency, and throughput requirements. Figure 3 shows our design using microservices and serverless computing.

**Figure 3. API pattern for supply chain processes**

**Design for operations**
To create a truly modular, microservice-based architecture, we used several practical design patterns that helped us keep integration both efficient and secure.

**Security**
Security is built into supply chain processes from the very beginning because the integration spans corporate boundaries

- **Authentication.** All cloud endpoints are authenticated by Microsoft-approved authentication methods and cannot be hosted anonymously. All calls should be audited for security.
- **Authorization.** Only authorized partners can conduct business transactions with Microsoft, and all authorized partners should be limited to their respective service contracts and API operations.
- **Message encryption.** All messages support encryption of data at-rest and in-transit to avoid data compromise.
- **GDPR compliance.** All messages and APIs support meet GDPR compliance standards.

**Monitoring**
Monitoring is categorized for all operations, enabling detailed, individual component monitoring and insight into broader, end-to-end scenarios.

- **Telemetry.** All components of the microservices components and APIs can be monitored using Azure App Insights and Azure monitoring capabilities. Business transactions can be examined end-to-end using unique identifiers, giving us the ability to ingest telemetry from external partners and vendors to capture the end-to-end supply chain business workflow.
- **Self-healing.** All transactions support auto-healing and message reprocessing capability so messages can be received and replayed in a consistent pattern.
- **Service health.** The design supports synthetic transaction monitoring to simulate end-to-end flow, highlight any service failures, and provide KPIs for healthy, unhealthy, and degraded states. Service rollup monitors end-to-end business process service health.
- **Dashboards.** All monitors are delivered with real-time dashboards to monitor all cloud services and provide actionable, end-to-end health states.
Creating an integrated plug-and-play supply chain with serverless computing

Benefits

Having a plug-and-play provisioning platform benefits Microsoft and our partners. We give our partners a more modular way of engaging with Microsoft, which helps derive more mutual value. Our partner onboarding process will be streamlined, consistent, trustworthy, and repeatable based on a unified, coherent Microsoft value proposition. Our PaaS-based supply chain platform provides several significant benefits:

- **Low management overhead.** With serverless PaaS infrastructure, there are no servers to patch and manage. Using the integrated mechanisms for managing Azure PaaS architecture gives us an automated, streamlined approach to deployment, provisioning, and maintenance.

- **Standard integration architecture.** Using PaaS architecture and microservices also makes it easy to expand our solution and onboard partners. We can use infrastructure as code to quickly provision and manage new resources. Because we are using the same codebase to deploy our PaaS components, our deployments are standardized and consistent.

- **Scalable architecture.** We’re using the native scaling capability of Azure PaaS to handle volumes that our previous solution simply couldn’t handle. And when volume decreases, so do the resources deployed to the supply chain solution—and the cost of those resources.

- **Global platform for bi-directional data sharing.** By migrating the supply chain services out of the datacenter and into the cloud, our solution has become instantly accessible to our partners through a resilient, global presence, which makes onboarding new partners and exchanging data quicker and simpler.

- **An agile and flexible platform.** Azure PaaS and microservices have enabled an unheard-of level of agility for MDSC. We can change small parts of the solution without negatively impacting the whole, and we delegate ownership and responsibility to smaller business units and dev teams. Integrating DevOps has put our business goals at the forefront of the development process. Our development cycles can incorporate what is immediately necessary to our business and our engineering teams are empowered to make any changes that are needed.

- **Low cost.** We no longer need to maintain a monolithic solution based on hundreds of servers. Our core operations happen in Azure, which reduces capital and maintenance costs.

Conclusion

We used microservices and serverless computing in Azure to enable digital transformation for MDSC. Our new solution has created a simpler, faster, and more dynamic supply chain management process. It’s easier for Microsoft partners to onboard, interact, and conduct business with us and we have end-to-end control and visibility of our processes. The solution has allowed us to focus not only on maximizing our revenue, but on doing so optimally and sustainably by building on a strong services-
based foundation across our entire supply chain, transforming the way we do business. We are taking this transformation further by exploring technologies like Blockchain, Internet of Things, and cognitive bots. These investments give Microsoft a competitive edge in the devices supply chain and bring maximum value to our partners and customers.

For more information

Microsoft IT Showcase
microsoft.com/itsshowcase

© 2017 Microsoft Corporation. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.