Alaska Airlines improves services in the cloud using containers and Azure Service Fabric

April 24, 2018

Authored by Brent Funk, Microsoft CSA for Alaska Airlines

This article is part of a series (https://blogs.msdn.microsoft.com/azureservicefabric/tag/case-study/) about customers who've worked closely with Microsoft on Azure Service Fabric (https://azure.microsoft.com/en-us/services/service-fabric/) . We look at why they chose Service Fabric, and dive deeper into the design of their application, particularly from a microservices perspective.

In this post, we profile Alaska Airlines. For their web applications, the journey from on-premises hosting to a cloud architecture involved one hackathon, minimal recoding, and an evolving set of Azure services.

Since 1932, Alaska Airlines (https://www.alaskaair.com/) has grown from a small regional airline to an international carrier with more than 40 million customers a year and 118 destinations. Outstanding customer service has always been the heart of their business--including all IT products and services.

To bring new services to their customers and optimize their operations, Alaska Airlines adopted cloud computing as a key element in their long-term strategy. For most of their customers, the front door to their services is alaskaair.com. The systems supporting the website enable people to make reservations, look up travel information, manage their mileage plans, get customer support, and more. Alaska Airlines' engineers looked for ways to take advantage of Azure services in their system to continue to drive outstanding customer service.

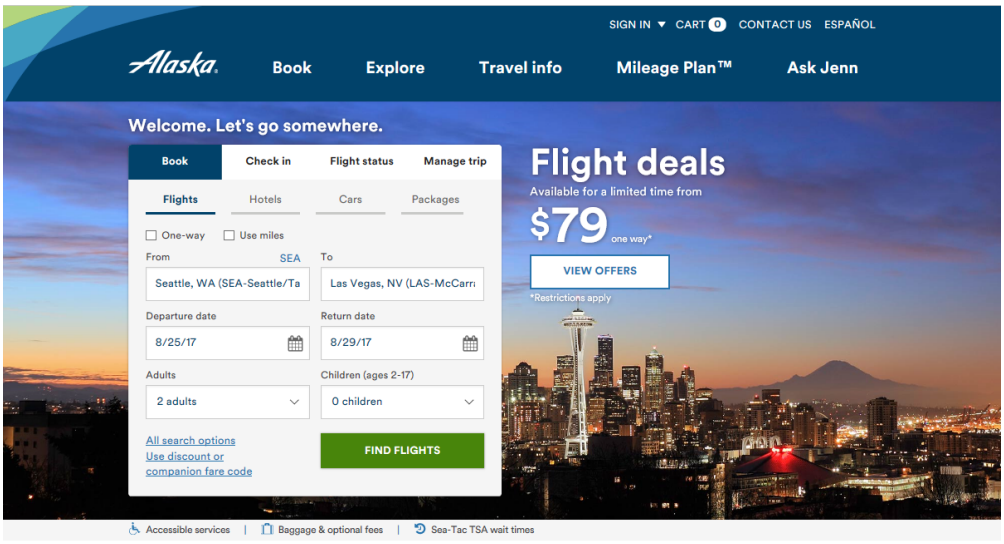

Figure 1: The alaskaair.com website

Customers provide personal information and make payments through the website, so the team at Alaska Airlines designed their system to be fault tolerant and extremely secure. As they incorporated mobile solutions, Alaska Airlines made an architectural decision to use an API-driven strategy. APIs support reuse and enable them to scale features and services independently. APIs also provide a single point of business logic regardless of the device customers use to access services.

To improve and modernize their implementation, the team began porting discreet services into Azure, such as the e-commerce engine (https://customers.microsoft.com/en-us/story/alaska-airlines-travel-transportation-azure) . The process got a lot easier when they adopted Service Fabric, a microservices platform.

Evolution of their service delivery

Alaska Airline's development teams have been creating monolithic systems and hosting them on premises for years. To ease reuse and increase agility of specific business functions, the team has been gradually splitting many of these systems apart by porting aspects into discreet services. Since development teams are well versed in Microsoft development technologies and platforms, they wanted an architectural approach that would be consistent in technology and make use of existing skillsets.

The most straightforward approach, they decided, was to pull out common aspects of their back-end systems and turn them into ASP.NET Web APIs. As early adopters of Microsoft Team Foundation Server (TFS) and then Azure DevOps (formerly Visual Studio Team Services), the team could easily modify existing build pipelines to support the new APIs. Release pipelines were relatively unchanged, since they just deployed the distributions to standardized application servers in the on-premises environment.

Over time, however, they outgrew their datacenter, which couldn't support the larger number of customers and emerging mobile-based delivery models. For example, they began to see loss of services caused by networking, hardware, or virtual machine issues, and performance impacts due to governed usage to maintain uptime requirements. To address these scale and capacity issues, they decided to extend their hosting model by adopting several Azure services--notably, Azure App Service (https://azure.microsoft.com/en-us/services/app-service/) for hosting their ported APIs, Azure API Management (https://azure.microsoft.com/en-us/services/api-management/) for secured access to those hosted APIs, and Azure Cache (https://azure.microsoft.com/en-us/services/cache/) for session management. Best of all, little to no recoding of their services was required and integration with Azure DevOps was simple.

The next challenge

Azure App Service makes it easy to extend web apps to support mobile clients, so the development team used it as the platform for migration to Azure. However, the team soon faced a new challenge as customer demand outpaced their system's ability to scale the more demanding back-end services. The problem was a design flaw that prevented the system from recovering gracefully from failures on top of the inconsistent performance of third-party APIs and downstream dependencies. Many issues resulted from too many threads of activities and sessions. The immediate resolution was to increase the number of servers--that is, scale horizontally.

For the Shopping Cart API, an absolutely critical business component, service disruptions aren't acceptable. Any downtime means lost bookings, lost revenue, and unhappy customers. During peak usage times such as Cyber Mondays, at least 100 virtual machines, or nodes, are needed to host this service to ensure zero interruptions.

With this in mind, the team went back to the drawing board to think about a different strategy for service delivery. They needed a solution that would:

- Uphold their use of Azure DevOps for CI/CD.
- Work with limited to no refactoring--there wasn't time.
- Run locally, on-premises, and in the cloud.

- Support a variety of libraries and technology versions.

Bring on the hackathon

The strong partnership between Alaska Airlines and Microsoft provided a unique opportunity--a hackathon with the Service Fabric product group. In one day, the Microsoft engineers worked with the Alaska Airline development team to build a proof-of-concept, microservices-based solution. They used a sample service hosted in a Service Fabric cluster on a local machine and in Azure. During the hackathon, the Alaska Airlines team saw first-hand that a microservices architecture using Service Fabric can be datacenter-agnostic and provide a way to move forward for hosting their ported APIs. This architecture also offered the capabilities and flexibility they needed for their new service delivery platform.

When they looked closely at their Shopping Cart API, however, its implementation appeared to be incompatible with Service Fabric. The Shopping Cart API wasn't built on the .NET Core, so it wasn't host-independent. The API required Internet Information Services (IIS), which isn't supported as an application host, since Service Fabric expects workloads to be self-hosted.

The solution to this challenge involved yet another technology introduced to the team at the hackathon--Windows Server Containers. The combination of containers and Service Fabric provided a solution for the Shopping Cart API that didn't require refactoring or rewriting the service. Containers combined with Service Fabric's upgrade and management capabilities resolved another issue as well--how Alaska Airlines could keep their images up to date in production.

Alaska Airlines found the shift to a container-based delivery model much easier than expected. They didn't need to change the core software or the platform (IIS) hosting their distribution. They merely added a few steps during their Azure DevOps build pipeline, incorporating a Docker image build activity (that incorporated their software package and IIS distribution) and a push of that image to their *private* Azure Container Registry (https://azure.microsoft.com/en-us/services/container-registry/) . The release pipeline, however, took some effort due to Service Fabric hosting. But the integration with the Azure services via APIs or PowerShell made the tasks fairly straightforward.

The following figure illustrates this new build and deployment workflow:

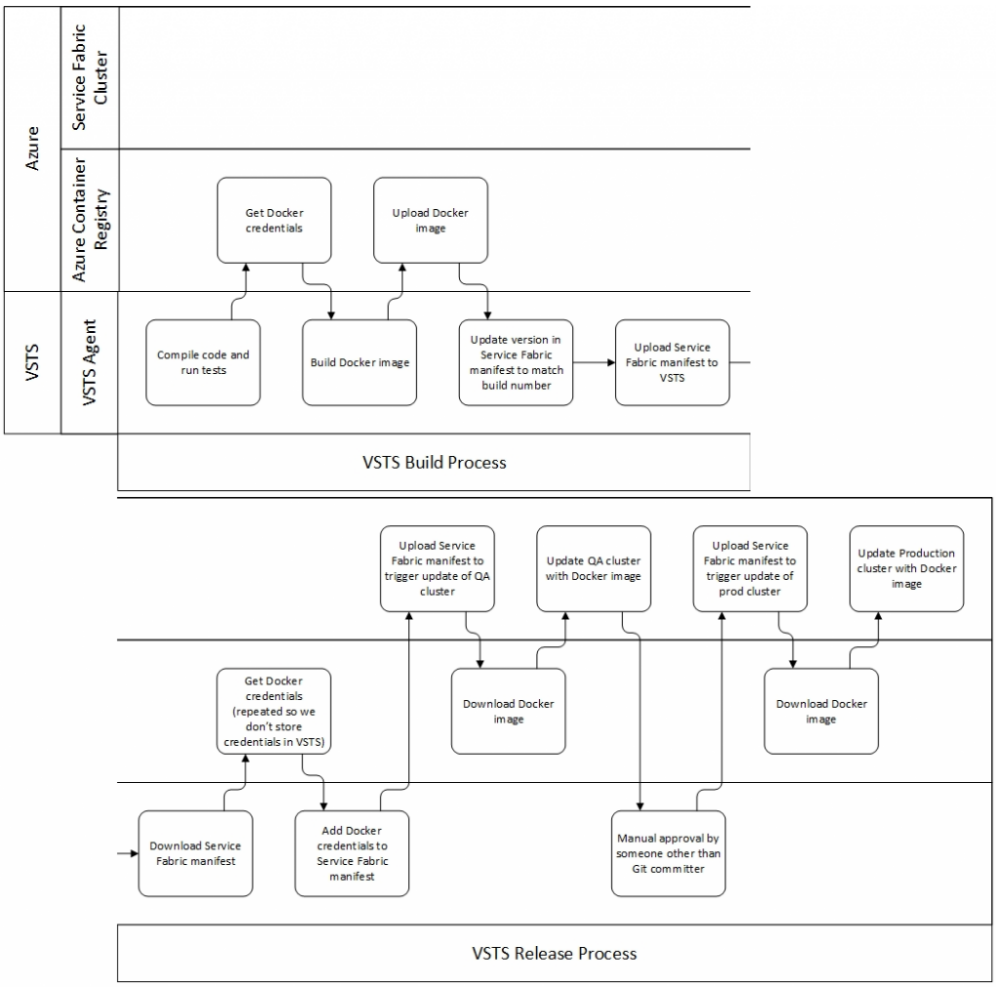

Figure 2. The new release pipeline for the Shopping Cart API.

Logical architecture of Azure Services used

In addition to Service Fabric, many other Azure services are part of Alaska Airline's Shopping Cart API architecture, each performing an indispensable function for the delivery of this service.

> “ Service Fabric provides a robust microservices platform without locking us into a single cloud provider.”
>
> —Carol Richardson: Senior Software Development Engineer
> Alaska Airlines

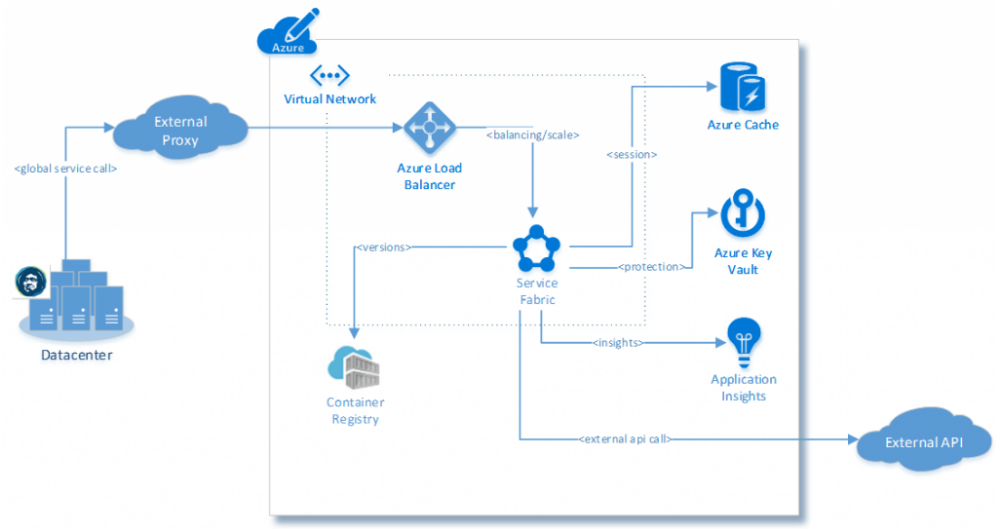

Figure 3. Overview of Azure services in the Shopping Cart API architecture.

The architecture includes the following services:

- Azure Load Balancer (https://azure.microsoft.com/en-us/services/load-balancer/) distributes incoming traffic so there's no need to manage the routing and health of each node hosting the Shopping Cart API.
- Azure Key Vault (https://azure.microsoft.com/en-us/services/key-vault/) helps safeguard credentials and encryption keys during runtime and CI/CD processes.
- **Azure Cache** shares state across all nodes in the cluster hosting the API. Although the Shopping Cart API is a stateless web service, it maintains a level of user state for the objects within a cart to allow automated scaling.
- Azure Application Insights (https://azure.microsoft.com/en-us/services/application-insights/) provides business intelligence for the API to augment the node-level health events that Service Fabric provides. The team uses Application Insights to verify performance within their service and also with chained API calls made by the Shopping Cart service.
- **Azure Container Registry** provides a private, secured environment that works with their CI/CD pipelines to maintain the latest Windows Server Container image, packaged with their Shopping Cart API

distribution. The Service Fabric cluster is configured to use the appropriate version as part of ongoing updates and management.
- **Azure Service Fabric** is the cloud-portable solution for hosting microservices.

Advantages of using Windows Server Containers with Service Fabric

Alaska Airlines was already well underway in their adoption of Azure PaaS cloud services for their APIs when they switched from Azure App Services to Service Fabric with Windows Server Containers for their alaskaair.com Shopping Cart service. This move gave them the scale and density they needed for their crucial service. Service Fabric and containers also serve as a proven hosting model for other high-demand, back-end services that need to maintain their current hosting platform, such as IIS, by using a container-based packaging model.

Improved performance and reliability

As an essential service, the Shopping Cart API for alaskaair.com must perform reliably at all times. Correcting errors manually is too slow to be acceptable. Service Fabric provides a highly available application uptime environment by monitoring each node and self-correcting when a node or the services on the node fails.

By using Service Fabric, the team also saw a significant drop in the time needed to deploy the API and make it ready for consumption (time-to-use)--from more than 20 minutes in Azure App Services to under 10 seconds with Service Fabric. This huge time savings really pays off when they need to troubleshoot or scale out.

With Alaska Airline's zero-downtime policy, the team appreciated the in-place upgrades with automatic rollback if necessary. Once the process is initiated, Service Fabric handles it seamlessly without user intervention.

Service Fabric also frees the development team to focus on business functionality and capabilities rather than local disaster recovery or scalability concerns. Alaska Airline's Service Fabric cluster relies on Virtual Machine Scale Sets to handle availability and auto-scaling based on custom metrics. For example, virtual machine instances can be restored within a cluster during a failure event. Service Fabric also rebalances

every node in the cluster and ensures it's operational before that node and its service are allowed to accept incoming requests.

Savings at scale

The Shopping Cart API isn't necessarily CPU-intensive, so vertical scaling is less important than bandwidth for the many incoming connections and requests. Service Fabric guarantees that container services get deployed on all nodes in the cluster. Using Service Fabric telemetry and service metrics, the engineering team can monitor alaskaair.com and use only as much compute power as needed.

Because scale sets are set up for the different types of nodes in their Service Fabric cluster, each type of node can be scaled out independently and easily managed either by specific performance characteristics or by predicted peak load times. The team can easily scale horizontally by adding and removing nodes as demand changes, paying only for what they use.

Greater control

A Windows environment for microservices and container management was the perfect fit for alaskaair.com. Services can run on premises or in the cloud with no impact to the development team, who appreciate the maturity of Service Fabric's internal governance and security controls.

Containers gave the team control over their service package--right down to the specific libraries and versioning. Alaska Airlines got the straightforward migration path they wanted for APIs so their services could benefit from the cloud.

The team gained control over management and testing as well. Service Fabric Explorer (SFx) shows the live cluster status so the team can ensure proper operation of each node. With Windows Server Containers, they can easily test functionality locally before deploying it into a Service Fabric cluster. If necessary, they can deploy Service Fabric locally to test a configuration change to a service or cluster on the spot, even using Service Fabric's chaos scenarios to test production-like random failures.

Summary

Alaska Airlines is now successfully running and scaling the Shopping Cart API on a 40-node Service Fabric cluster with their custom Windows Server Container, meeting the current and future needs of their global customer base.

The deployment was such a success that many other development teams at Alaska Airlines now intend to migrate their components to Azure. With more than 100 additional APIs supporting alaskaair.com, their long-term cloud strategy continues to evolve.

> " We love the pristine and predictable environments provided by Windows Server Containers. Service Fabric brings us a safe and consistent deployment strategy that maximizes uptime. The marriage of the two in Microsoft Azure gives us unlimited potential."
>
> —Ralph Feltis: Software Engineer
> Alaska Airlines