Honeywell builds microservices-based thermostats on Azure

April 20, 2018

Contributors: Greg Feiges, Tomas Hrebicek, Richard Sirny, and Jiri Kopecky of Honeywell

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 Honeywell and their Internet of Things (IoT) solution, which was originally designed as a microservices architecture. The engineers at Honeywell migrated the solution to Service Fabric to gain the scalability they needed.

More than 125 years ago, Honeywell (https://www.honeywell.com/who-we-are/our-history) started as innovators in heating technology with a patented furnace regulator. Today they are a Fortune 100 software-industrial company with operations in 70 countries and more than 129,000 employees worldwide. They deliver industry-specific solutions that include aerospace and automotive products and services; performance materials; and control technologies for buildings, industry, and homes, such as the T5 and T6 thermostats.

Several years ago, Honeywell Homes and Buildings Technologies (HBT) entered the IoT era with Total Connect Comfort, its first offering for connected thermostats. While wildly successful, the solution was a child of its time technologically: a monolith architecture hosted on premises. With the advent of the cloud, HBT made a strategic decision to create a new, born-in-the-cloud system that would rely on a scalable microservice architecture.

Using Microsoft and open source technologies, they developed Lyric, a solution to support a connected home offering, including thermostats that enabled home owners to control their home environment remotely and save on energy bills. The platform enables extended services as well. For example, by using real-time data streaming from a connected thermostat, the company can provide maintenance alerts to HVAC professionals and allow them to proactively address problems and maintain service levels.

For this early innovation, Honeywell relied on Project Orleans, a next-generation programming model for the cloud developed by Microsoft Research and released as open-source software in 2014. Over time, as new cloud services became available, Honeywell wanted their solution to evolve as well.

IoT smart home systems

The Lyric family of products from Honeywell provide home comforts. Home owners can remotely control cameras, thermostats, and home security services using the Lyric app on their smartphones and tablets. The T5 and T6 WiFi thermostats are two of the smart home devices that work with the Lyric app. From a smartphone or tablet, users can monitor and control their heating. The Lyric app uses geofencing technology to track the location of home owners and update their thermostats, which are designed to be compatible with home automation ecosystems such as Amazon Alexa, Apple Homekit, SmartThings, and IFTTT.

The Lyric cloud solution comprises several logical blocks (Figure 1). The Connected Home Integration Layer acts as a gateway into the system for Lyric mobile application and various third parties. It provides several user-centric services such as user management, location management, and alerting. This layer is also responsible for interacting with specific subsystems for command and control of devices, including cameras and thermostats.

> " I just love the convenience of adding a new service into Service Fabric applications. It gives you so much flexibility with your architecture and enables it to be truly microservice-oriented "
>
> —Jiri Kopecky: LCC Architect
> Honeywell

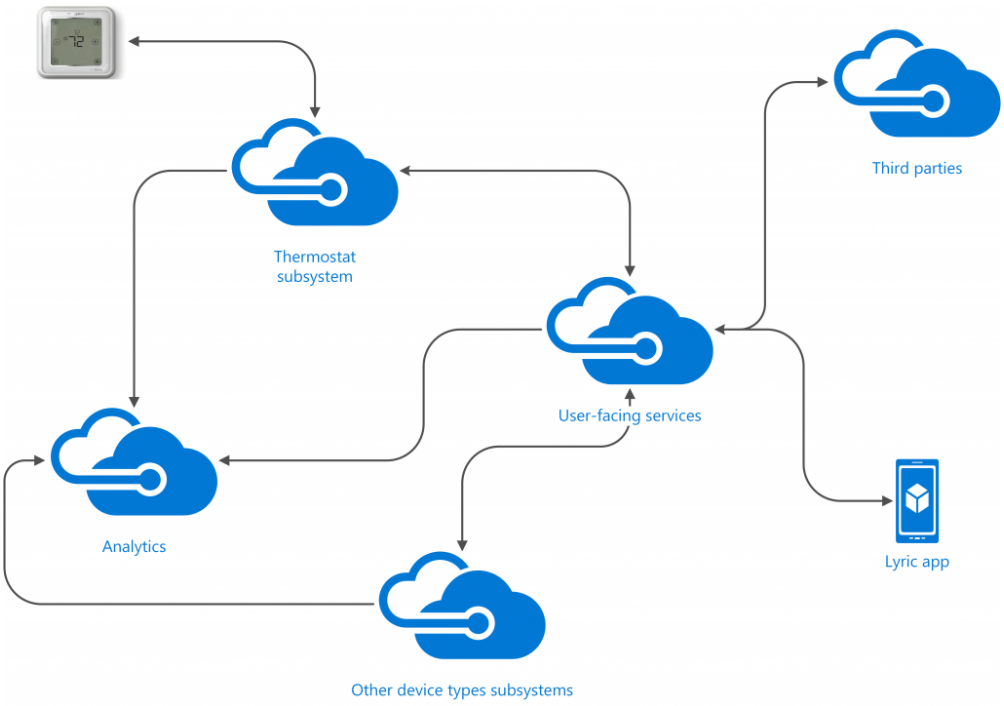

Figure 1. Lyric cloud solution

Migrating a microservices architecture

The first version of this IoT architecture was built using Project Orleans. With Orleans, the team at Honeywell had the framework they needed to build a distributed, highly-scalable cloud computing platform, without the need to apply complex concurrency or other scaling patterns.

"Thanks to our on-premises system, we were no strangers to the issues one has to tackle in distributed computing," said Richard Sirny, Senior Software Engineer. "As soon as we started using Orleans, we were

struck by how much simpler it had become to introduce a new feature and, overall, work with the system."

In the initial phases of the thermostat subsystem design, the Honeywell engineering team worked closely with Microsoft. They were recommended to consider using an actor-based framework to represent the connected thermostats. In addition, Microsoft let the team know that a new offering, Service Fabric, was coming soon. Since it wasn't ready yet, the safest design approach at that time was to start with the Orleans Virtual Actor Framework hosted on Azure Cloud Services (https://azure.microsoft.com/en-us/services/cloud-services/) and put in place a design that would ease the switch to the new actor framework when it became available.

Architecture on Orleans

The thermostat subsystem, Lyric Comfort Cloud (LCC), allows control of Lyric T5 and T6 thermostats and potentially other devices. Its architecture is based on microservices that handle different aspects of connecting and controlling IoT devices. From an implementation perspective, each microservice is a mix of Azure Cloud Services web and worker roles. A typical responsibility for a worker role might be accessing or caching data from various storage types, while web roles host ASP.NET Web API applications that act as authenticating and authorizing proxies for worker roles.

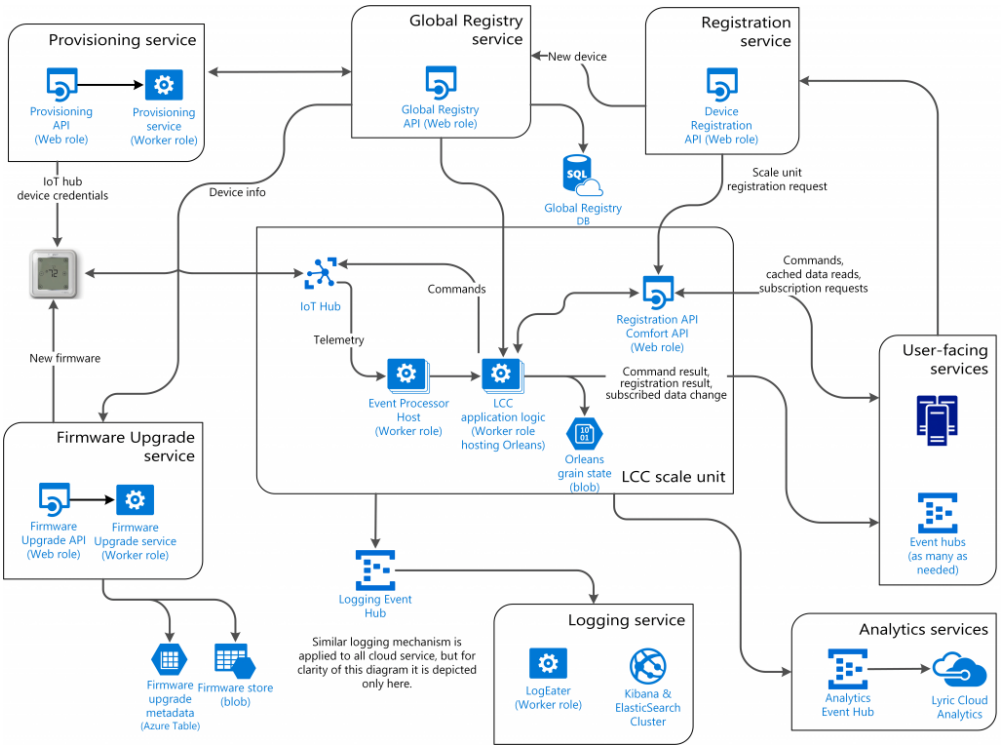

Figure 2. The early LCC subsystem architecture based on Orleans, microservices connect, and control IoT devices.

As Figure 2 shows, the architecture relies on Azure Cloud Services and Orleans. User-facing and analytic services are not part of the thermostat subsystem, but they are included to provide the context in which it operates.

The following microservices form LCC:

- **LCC Scale Unit**: This service implements the business logic that allows users to interact with their thermostats. Its clients can query cached runtime data (such as temperature) and connection status, send commands to a thermostat, and subscribe to changes in device data.

  This service supports horizontal scaling of the subsystem. While other services could in theory use this pattern as well, this service was identified during the initial design to be the one most used. Consequently, user-facing services are implemented so that they can interact with multiple scale units. As a result, all other services may be considered global in context of LCC.

  Internally, this service consists of worker role for receiving messages from Azure IoT Hub (https://azure.microsoft.com/en-us/services/iot-hub/) , and an Orleans silo host worker role with grains (actors). The grains represent thermostats. They also enable orchestration for thermostat registration and deregistration flows, implemented as a web role with a REST API exposing the thermostat and registration functionality.

- **Registration**: To introduce a new device into the system, the user-facing layer must send a request with new device details to this service. A scale unit is then chosen and the registration process is started. When registration is finished, the device can start communicating with LCC. This service is implemented as a simple web role with REST API, while the registration grain in the chosen scale unit is responsible for the registration process.
- **Firmware Upgrade**: Devices can query this service to determine if a newer version of firmware is available. It consists of a web role that provides a REST API and a worker role that caches firmware-related data and provides firmware updates.
- **Provisioning**: The main purpose of this service is to provide devices with tokens necessary for connecting to the Azure IoT Hub. It is implemented as a web role that hosts a REST API that acts as an authenticating proxy to the worker roles where token generation takes place.
- **Global Registry**: Details of all registered devices, including binding to the scale unit are accessible through this service. It is a web role REST API overlay over SQL storage.
- **Logging**: This service collects and displays logs from all LCC services

The expectation for the new system was to handle millions of thermostats. As a result, the team decided to use a design where a single, connected device would require only a single grain (actor). Two types of Orleans grains were used in the LCC Scale Unit service: job and thermostat grain.

The job grain was used in the process for registering and unregistering devices. Several components had to be interacted with and prepared before the scale unit service deemed a thermostat registered or unregistered. Since it operated in a distributed environment, any step of the process could incur transient errors and retries. Grain state was used to track progress while reminders and timers were used to run different steps.

The thermostat grain implemented all business logic related to thermostat operation, including caching real device state, messaging, connectivity tracking, and state changes subscriptions. To keep the design actor-framework agnostic and to have greater flexibility with storage options, Orleans grain state was not used. Device state and other data were persisted into Azure Storage (https://azure.microsoft.com/en-us/services/storage/) blobs. The solution also used other features such as timers for non-blocking message sending and reminders for connectivity tracking.

Learnings in the Orleans-based implementation

Overall, the Orleans-based implementation had very few issues and worked well. However, there were several consequences of the chosen compute model:

- To achieve the desired availability, the web as well as some worker roles had to be over-scaled even though the utilization was minimal most of the time.
- Although fully automated infrastructure and environment creation was not impossible with Azure Service Manager and Azure Cloud Services, it proved challenging enough for the team to compromise on partially automated environment creation.

With this architecture, scripts supported deployment to different environments. The team developed a suite of scripts that were meant to enable declarative configuration of applications and their dependencies. However, the scripts were not able to create or alter dependencies in some cases that required a check for complex conditions. For example, infrastructure creation was not fully automated. The situation could have

been solved by using Azure Resource Manager (https://azure.microsoft.com/en-us/features/resource-manager/) templates, but they were not available when the scripts were created.

In addition, the team wanted to be able to create a single package that would travel through their CI/CD pipeline without creating builds for different environments. This was rather difficult with Azure Cloud Services as the size of virtual machines was embedded in the package.

A new approach to services

To gain more density and resiliency, the team decided to move their Orleans-based microservices architecture to Service Fabric when it became available. The challenge was to rearchitect their solution and migrate multiple front-end and back-end microservices designed to represent the Lyric T5/T6 connected line of thermostats, then host them in Service Fabric.

To jump-start the migration and make best use of Service Fabric in the new architecture, the Microsoft Service Fabric team worked closely with the Honeywell engineering team. In a mere week, they built a Service Fabric application prototype based on the code of the existing scale unit service.

The experience showed the team exactly what they needed to do:

- Port all services to Service Fabric except the existing logging service, which worked well.
- Rewrite all REST APIs—a strategic, long-term investment—to use ASP.NET Core.
- Make full use of the Service Fabric Reliable Actor features for the scale unit service. Namely, the preferred data store should be the actor state. However, some critical data could benefit from a better disaster recovery story—for example, if the local drives that Service Fabric uses to store state were to fail or be destroyed by some errant management operation. While the actor state could be backed up and stored inside of Azure Storage, since the write performance of this data wasn't critical, Honeywell elected to store the data directly in storage, simplifying their operations..
- Automate infrastructure creation, and let the application deployment create all the resources needed by the application if possible.
- Run the registration process as part of the registration service, the not scale unit service.

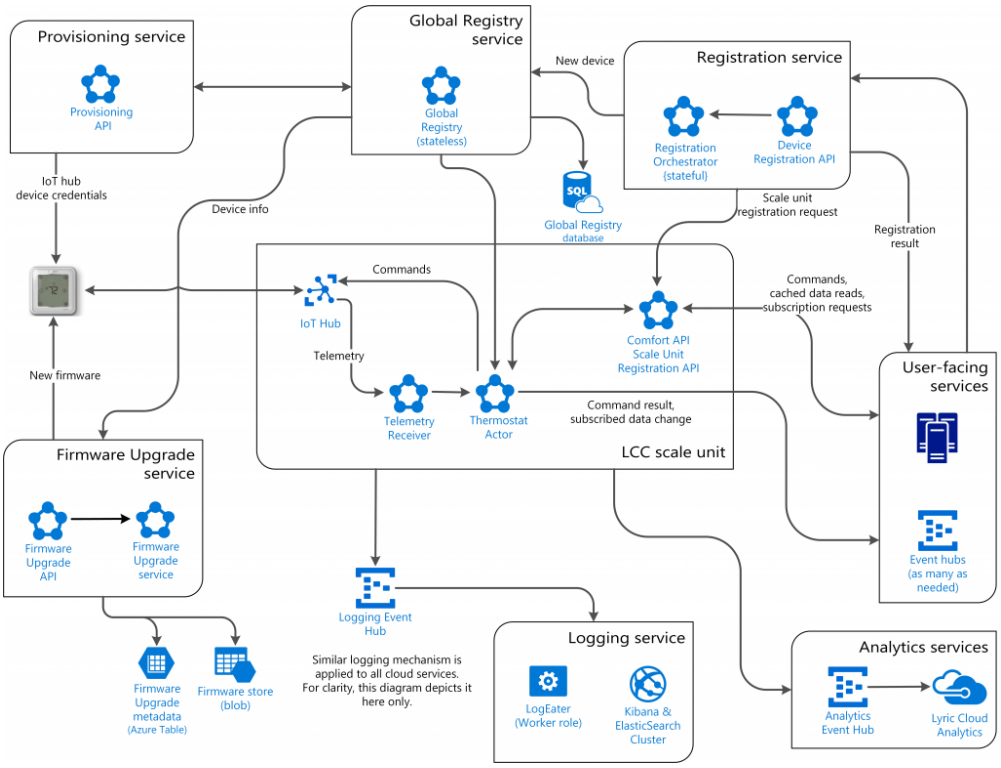

Figure 3. The new architecture for the thermostat subsystem, based on Service Fabric.

Figure 3 shows the new architecture. The team made many minor changes inside the microservices, but functionally, the architecture changed very little. Most of the refactoring took place in the scale unit and registration services. The former now caches its data as actor state to benefit from storage locality. The registration service was extended to include a stateful Service Fabric service that acts as an orchestrator of the process.

A bigger impact was in the infrastructure. The Service Fabric cluster is hosted on two Virtual Machine Scale Sets—one dedicated to APIs and the other to back-end and stateful services. Everything is described using Resource Manager templates. Creating a new environment takes just a few manual steps related to configuration. In addition, application deployment is automated, and dependent resources are created and updated with each deployment.

Migration strategies for a microservices architecture

Several migration strategies were considered for the registered devices. Originally, the team intended to migrate all devices in one go with short downtime for all users. However, after careful analysis and feedback from stakeholders, a less disruptive plan was conceived. The new architecture required device data in the global registry service to be migrated to a new SQL database with a different schema. After thorough analysis, it became clear that only one service—registration—would experience any downtime.

Moreover, the team realized that the original Orleans-based scale unit service could continue to run as is, and a new, empty scale unit service could be introduced gradually into the Lyric ecosystem. The last piece was to add support for migrating devices between scale units.

The whole plan can be summarized in the following steps:

1. Deploy Service Fabric cluster.
2. Deploy all the applications to the cluster.
3. Stop the registration service.
4. Migrate data to the new SQL database.
5. Stop all other global services: provisioning, firmware upgrade, and global registry.
6. Switch to the new services by altering appropriate DNS records.
7. Gradually migrate all devices from the old scale unit to the new one.

The plan was executed without a problem. Over the next few weeks, all devices were migrated from the old scale unit to the new one.

The similarities in the frameworks used by Orleans and Service Fabric made the migration relatively pain-free, but a few glitches became visible only after the solution had been running.

For example, although the Service Fabric Reliable Actor source code is open source now, initially it was not. During this time the team had difficulty validating some of their assumptions. One assumption was that both the Virtual Actor and Reliable Actor frameworks handled memory similarly, so when they encountered a rather subtle memory leak in the new architecture, they were puzzled.

It turned out that there were differences in how the two frameworks handled Timers. Honeywell had been manually disposing the Timers before, but that code caused a leak in Service Fabric since the *record* of the timer remained in Service Fabric. The fix was simply to explicitly call Service Fabric's API for removing the timers (UnregisterTimer) rather than disposing it directly.

Another migration challenge had to do with the use of preview tools. The team was keen to switch to ASP.NET Core, but at migration time, most of its tooling was in preview. As a result, projects builds that

used the new .csproj format (a Visual Studio .NET C# Project file extension) succeeded on some development computers and failed on others. Yet the situation improved steadily with each new version of the tooling as ASP.NET matured.

After the launch, the team also had to rethink some of their application monitoring. The original architecture used only a few virtual machines running a single application process. With fewer application instances, the performance counters make it simple to understand overall application health. On Service Fabric, some services with a high number of partitions need to collect performance counters per partition. The team didn't anticipate that collecting so many counters would cause performance issues for their monitoring tool, so they had to find another.

Advantages of Service Fabric

One of the immediate advantages of the new infrastructure was cost savings. By using Service Fabric, Honeywell dramatically reduced the number of virtual machines they needed from 48 to 15. With fewer moving parts, the platform became significantly less expensive to maintain overall.

Other key benefits included:

- **Speed**: Data locality provides great performance: 99 percent of scale unit calls typically finish within 40 ms.
- **Greater stability**: After implementing Service Fabric, there were fewer outages compared to the old cloud services architecture.
- **Smoother deployments**: The team created a modern CI/CD pipeline with relative ease using Azure Resource Manager templates.
- **Elastic scaling**: The front-end tier benefitted from elastic scaling. Although this functionality was not available at the time of migration, it is possible now when the appropriate durability level is chosen.

Summary

Any platform migration involves a learning curve, but the team at Honeywell enjoyed working with both the Orleans and Service Fabric Reliable Actors frameworks. In the end, though, the promise of Service Fabric was too great to ignore, and they have no regrets about moving the Lyric solution.

The team's success in the cloud has set an example for the company. A team located near the LCC developers saw their early experience with Service Fabric and drew upon their learnings to take a different

project from nothing to production within four months. Now many other development teams at Honeywell intend to use the LCC team's approach for their projects and take advantage of Azure.

> "The deployment automation and upgrade process with Azure Cloud Services was a huge improvement from previous on-premises deployments. Service Fabric and the Resource Manager deployment model pushed this even further and helped us greatly to have a fully mastered CI/CD pipeline."
>
> —Tomas Hrebicek: Senior Software Developer
> Honeywell