Service Fabric Customer Profile: Info Support and Fudura

Authored by Sander Molenkamp and Rob van Gils from Info Support. Reviewed by Matthew Snider from Microsoft.

This article is part of a series (http://aka.ms/TCP) about customers who've worked closely with Microsoft on Service Fabric over the last year. We look at why they chose Service Fabric, and we take a closer look at the design of their application.

In this installment, we profile Info Support and their Metering-as-a-Service (MaaS) application running on Azure that was built in collaboration with Fudura. The architecture uses Service Fabric, Azure IoT Hub, and Azure Stream Analytics.

Info Support is an IT services provider that specializes in developing, managing, and hosting tailor-made software, business intelligence, and integration solutions. Info Support also offers over 300 IT courses primarily focused on software development. Info Support's central office is in Veenendaal, in the center of The Netherlands, but their distributed workforce of more than 400 employees operates from offices throughout The Netherlands and Belgium. By combining their knowledge of IT and their markets, Info Support offers innovative, industry-oriented solutions.

When Fudura came to Info Support, they were providing metering services that help make energy flows visible and manageable. They measure and analyze energy consumption; give advice on energy infrastructure and savings; rent technical equipment; and design, manage, and maintain energy supply systems.

Info Support helped Fudura make a transition to a smart metering solution with a microservices-based architecture built on Service Fabric and deployed in the Azure cloud.

> *"Service Fabric is key in our mission to deliver scalable and affordable services. By leveraging the many features of Service Fabric, the software team can focus on creating business value."*
>
> *- Peter Meulmeester, Fudura Senior Business Developer*

Making energy flows visible and manageable

Fudura operates a heterogeneous pool of connected field devices, such as gas and electricity meters, each emitting valuable data. This telemetry was being sent to a legacy platform mainly consisting of large software packages deployed on premises. But that platform was no longer flexible or cost effective enough to meet new customer demands. Fudura needed to speed up time-to-market, provide competitive pricing, and quickly innovate in an ever-changing energy market.

A new architecture based on microservices provided the solution. MaaS allows Fudura to collect, validate, and distribute telemetry originating from various connected devices. The MaaS platform takes care of provisioning and monitoring these devices. Collected telemetry can be routed to standard customer portals. When customers want more options, the Fudura team can quickly build specialized business scenarios and deploy them flexibly as separate microservices. By providing an easy way to interact with a large set of supported devices, MaaS gave Fudura the flexibility they needed to empower customers to take control of their energy flows.

> *"By automating device connectivity, provisioning, and monitoring at a large scale, Fudura is able to provide customers insight into their energy flows at a very competitive price point."*
>
> *- Jaap Mintjes, Fudura Project Manager*

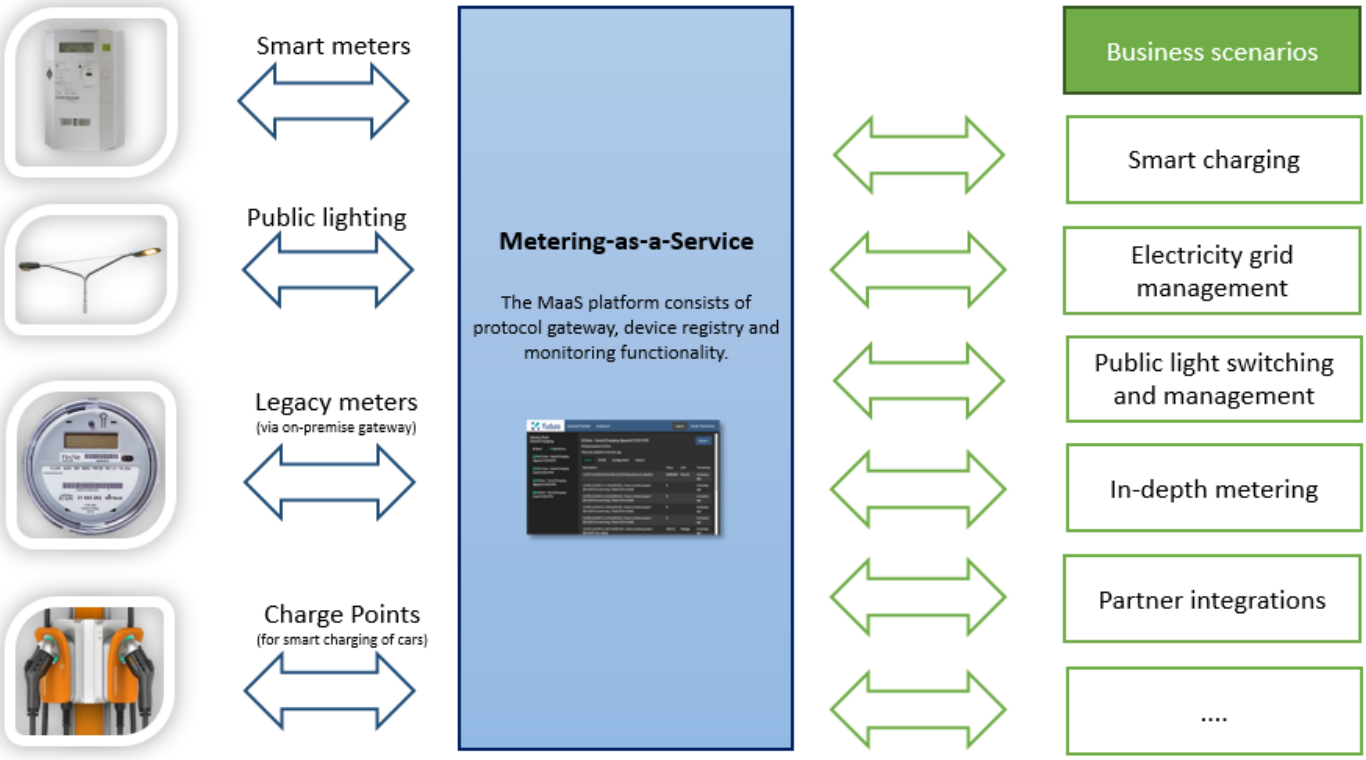

Figure 1. A high-level overview of the MaaS platform. Telemetry is collected from devices and used in various business scenarios.

Building a scalable MaaS platform

The MaaS platform consists of three main parts:

- **Protocol Gateway**: This Service Fabric application connects to devices using various metering protocols such as device language message specification (DLMS). For each physical device, a corresponding device agent starts and handles all communication with the device. Collected telemetry is sent to Azure IoT Hub for further processing. Multiple instances of the Protocol Gateway are deployed in different virtual networks to support connections with devices from various partners and customers, typically through a virtual private network (VPN).

- **Device Hub**: This Service Fabric application provides core services such as a device registry for storing device configuration, a device controller for starting and stopping device agents, and services for transforming incoming telemetry into materialized views. To manage and monitor devices, Fudura and its customers use Control Center, a business-critical component of Device Hub built using ASP.NET Core and Angular.
- **Tailor-made business scenarios**: If a customer requires software components not offered by the MaaS platform, Fudura can meet the need easily. For example, a microservice can be created to aggregate telemetry into a customer-specific monthly report and deliver it to the customer's FTP server.

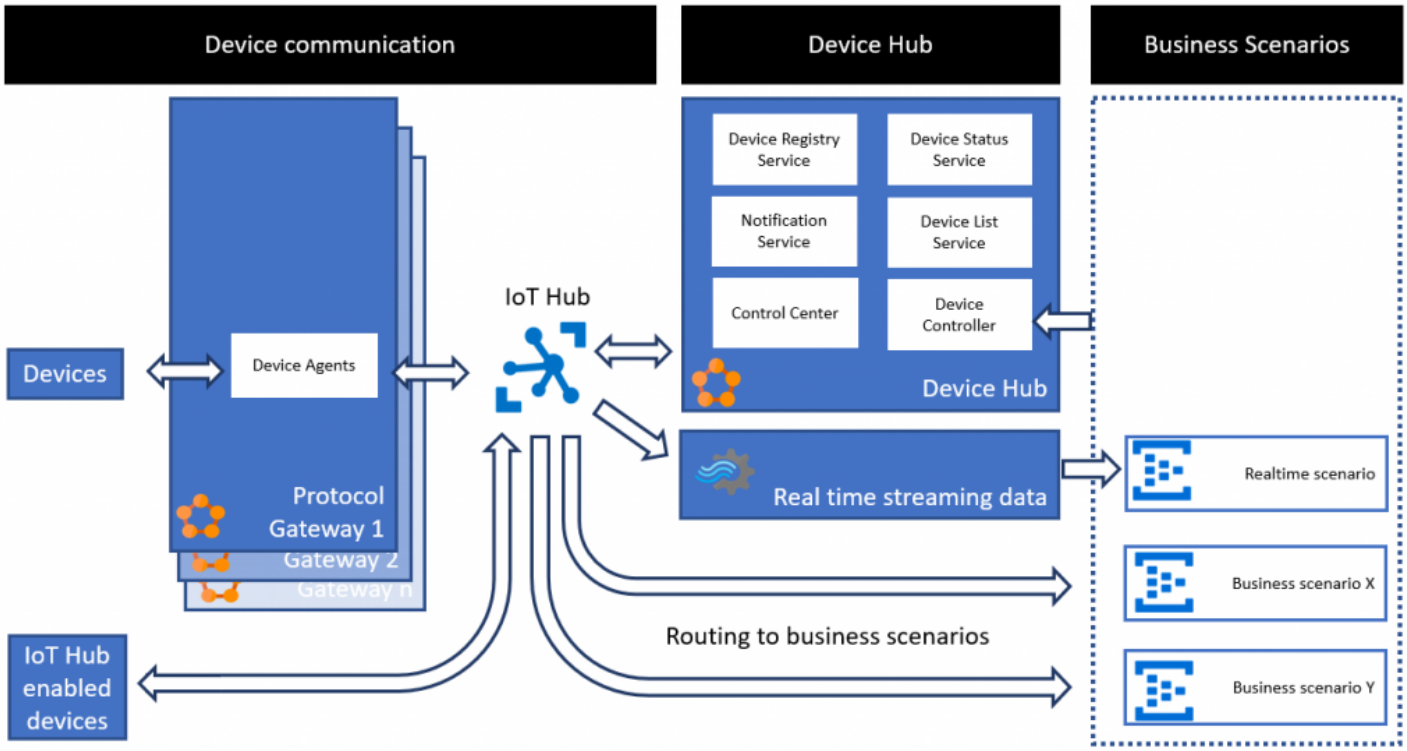

Figure 2. MaaS platform communications flow between Protocol Gateway, an IoT hub, and Device Hub to support various business scenarios.

All telemetry collected by the Protocol Gateways is published on an IoT hub, including device readings and metadata events such as device health information. From the IoT hub, an event can follow four major flows:

- All events are processed by the Device Hub application. Several of its services, including Device Status, generate views from the incoming events. Control Center can then display these views of the latest readings and health information to users.
- IoT Hub message routing dispatches events to the appropriate business scenarios. Events that match a filter are forwarded to a scenario-specific event hub, and then can be ingested by the business scenario.
- When further event processing is required for a business scenario, Azure Stream Analytics provides efficient windowing, geospatial operations, and filtering.
- Each event is archived in a deep storage store using Azure Blob storage. To support the growing archive, Azure Data Lake Store (https://azure.microsoft.com/en-us/services/data-lake-store/) will likely replace the Blob storage implementation in the near future.

> *"Service Fabric kickstarted the development of our platform by reducing the complexity of building a reliable and scalable application."*
>
> *- Rob van Gils, Info Support Senior Software Engineer*

Delivering a great user experience

To make sure the MaaS platform delivered great performance, the architecture was designed to keep frequently accessed (hot state) data close to the compute layer. Service Fabric stateful services offered very low-latency data access while enabling high reliability and availability. The Device Status stateful service stores its views in Service Fabric Reliable Collections for fast loading times when they are requested through the Control Center front end.

Initially the development team used the Reliable Actors programming model to process the events and store the views. However, this design did not consider that in the single-threaded model, read requests from Control Center would end up queued behind write requests. In the actor model, an actor handles only one request at a time, an important benefit when updating the views in the actor, but not very good for querying those views. View retrieval was quick--but only when no updates were queued.

To remove the dependency on a single threaded model, the implementation was changed to use Reliable Dictionaries to store the views instead of Reliable Actors. The service can now handle queries while simultaneously updating the views, which allows Control Center to be consistent while delivering great performance.

The Device Status service stores the state of all the devices in the platform, including their health state and latest readings. The development team is currently investigating the Azure Time Series Insights (https://azure.microsoft.com/en-us/services/time-series-insights/) service, now in public preview, as an alternative for storing the collected readings.

Process managers and notifications using the actor model

In Control Center, Fudura manages device operations such as starting and stopping device agents. Control Center uses the Device Controller service to send a command to the correct Protocol Gateway over HTTP. The Protocol Gateway then executes the requested command.

As the platform evolves, support for more complex scenarios is required. The team is developing a Device Management service containing process managers that coordinate processes consisting of multiple commands. The service also monitors progress. For example, the process for provisioning a new device consists of the following steps:

1. Send a RegisterDevice command to the Device Registry service to save the device configuration.
2. Wait for the DeviceRegistered event to occur. This event is raised by the Device Registry service after the configuration has been successfully saved.
3. Send a StartDeviceAgent command to the Device Controller service.
4. Wait for the DeviceAgentStarted event to occur, which signals that the device agent has successfully started and will begin collecting telemetry.

The steps above represent the ideal flow. In production, though, various things can go wrong, so the process managers are designed to retry commands if the correct events are not received in the allotted time.

The Reliable Actors programming model is a great fit for the Device Management service, because process managers are autonomous units of work that can be performed in parallel with each other. Each process manager is modeled as a single actor type.

When starting a process, a new instance of the corresponding actor type is created. Each process manager actor instance is assigned a random actor ID. The actor ID is used for correlation; it is piggy-backed onto commands and events so that incoming events can be routed to the process manager actors that are waiting for them.

Process managers can take some time to complete depending on the type of process. The caller that starts the process manager should not be blocked until the process manager has completed. To solve this issue, a reminder is scheduled to fire immediately when the process manager is called. The real work is done in a new call context when the reminder has fired, allowing the original caller to continue doing other work.

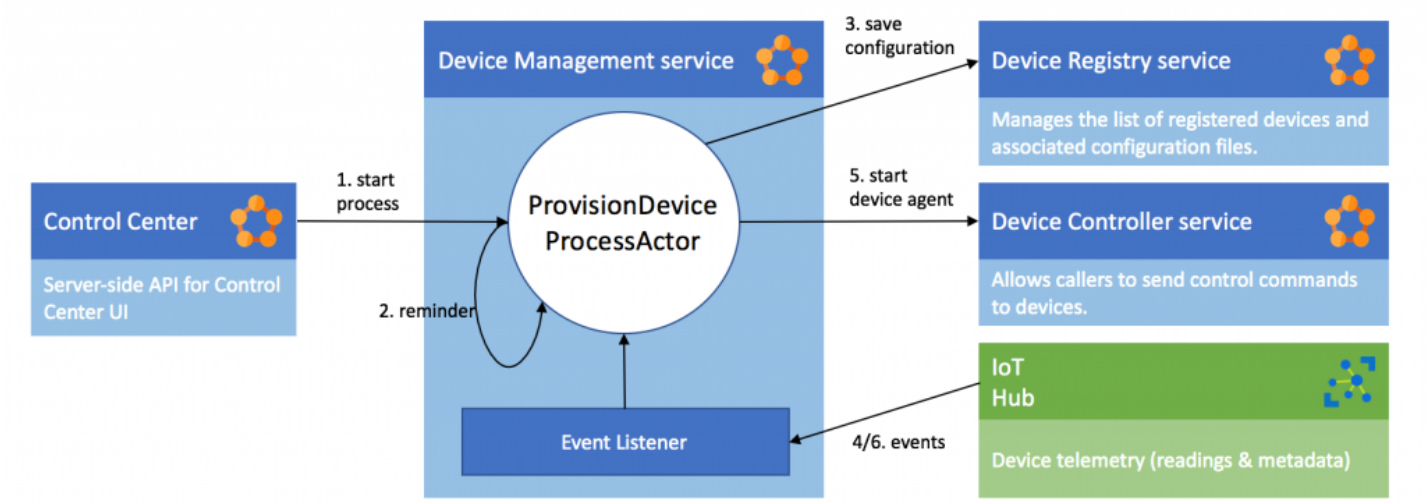

Figure 3. Overview of process managers in the Device Manager service.

> *"The Reliable Actors programming model is a great fit for building process managers, because they are autonomous units of work that can be performed in parallel."*
>
> *- Sander Molenkamp, Info Support Solution Architect*

To report progress, the process managers call the Notification service, also built using the Reliable Actors programming model. It contains two types of actors: the UserActor receives notifications targeted for a specific user, while the TenantActor receives notifications targeted to all users of a specific tenant.

Actor events are used for these notifications. Actor events provide a way to send best-effort notifications from an actor to clients. Clients start listening for these events as soon as a user logs into Control Center. Each received actor event is forwarded to a SignalR Hub that forwards the notification to the user's Control Center instance running in a browser, where it is displayed.

Releasing and testing on Azure

At peak times, a team of 10 developers work on the MaaS platform. To work together efficiently, a comprehensive build-and-release pipeline was configured using the Visual Studio Team Services infrastructure. This pipeline uses four environments: local development, testing, acceptance, and production.

Developers use the Service Fabric local development cluster to test and debug new MaaS platform features. Local development clusters run the same code as the production environment (no emulators are involved). Developers can be certain that the tested code can be deployed to other environments.

To run integration tests with physical devices, developers can provision a test environment on demand based on the Git feature or release branch they are working on. Unlike a local development environment, no firewall or NAT routing constraints prevent testing, and there is no limit to the number of on-demand environments that developers can run consecutively.

Developers use Azure Resource Manager templates to automatically provision a Service Fabric cluster as well as all other dependencies, which can include Application Insights, IoT Hubs, storage accounts, and Stream Analytics pipelines. After a cluster is provisioned and the software is deployed, integration tests can be run. To reduce costs, an on-demand environment is deployed as a single node test cluster. By changing a variable in the release pipeline, it can be provisioned as a multi-node cluster, a useful technique when testing failover and other scenarios.

In test environments, Service Fabric application upgrades are disabled by default. When releasing a new version of the MaaS platform to the test environment, the application is removed and redeployed, leaving no state behind. This approach supports faster test cycles when the team is actively working on a new feature.

The acceptance environment runs continuously and connects to a representative set of testing devices. On each deployment, developers use Resource Manager scripts to set up the environment even though it runs continuously. Resource Manager scripts are idempotent, meaning a template can be deployed repeatedly without breaking anything. When developers make supported changes to a template, the acceptance environment is upgraded accordingly.

When a release has passed all required tests in an on-demand environment, it is deployed to the acceptance environment. Almost identical to the production environment, the acceptance environment is used to test the application upgrade process. After an upgrade is deployed successfully, final tests are run and, when approved, the new features move into the production environment.

Benefits of Service Fabric

The MaaS platform was built from the ground up using Service Fabric, which enabled the Info Support developers to:

- Develop high-availability, low-latency microservices.
- Use Service Fabric's flexible programming models provided by Reliable Services (stateless and stateful) and Reliable Actors.
- Run more microservices on the nodes compared to virtual machines or Azure Cloud Services, for a cost-saving, higher density deployment.
- Future-proof scalability. Nodes can be scaled in and out and up and down as needed. Service Fabric automatically distributes the microservices to make optimal use of the available resources.

Summary

When Fudura needed a more flexible, cost-effective platform for collecting and distributing device telemetry, MaaS was the answer. Info Support helped Fudura to make the transition, using Service Fabric to build a microservices-based architecture deployed in the Azure cloud.

Service Fabric greatly reduced the complexity of building a distributed application. It handles reliability, scalability, and high availability, so the software team can focus on creating business value, thereby shortening the time to market. And now that the Info Support team can perform incremental deployments in a controlled, predictable way, they're more agile. The deployment of on-demand test environments is automated using Resource Manager templates, and multiple features can be tested at the same time with real devices connected to the Azure cloud.

Service Fabric is the heart of the new MaaS platform that includes IoT Hub and Stream Analytics. Now Fudura can provide customers detailed insight into their energy flows at a very competitive price point.