Putting energy data to work with pub/sub

Going beyond REST APIs, and embracing the power of a new paradigm

26 April 2021

AMMP

Executive Summary

This paper explores the prevailing ways in which energy data is exchanged between parties – specifically in the context of data from PV and PV/hybrid systems. It lays out the most common established approaches, including "data push" and "data pull", and discussed their respective advantages and shortcomings. The paper also presents the "publish/subscribe" (pub/sub) pattern of data exchange. While use of the pub/sub pattern is quite well established in other industries, such as financial services and industrial IoT, it is still not widely used in the context of renewable energy data exchange. This paper presents the advantages of the pub/sub approach, as relating to this context.

In summary, pub/sub has a number of distinct advantages over established approaches. First and foremost, it allows for providers of data (e.g. devices in the field) to send real-time data and events. This can for example ensure that critical events are acted upon promptly. It also provides a simple mechanism for eliminating gaps in historical data. Meanwhile, pub/sub allows for data consumers to subscribe to the specific data streams ("topics") that are relevant to them. Overall, pub/sub protocols provide rich data interchange capabilities. Meanwhile, they are straightforward to set up and maintain for both providers and consumers of data. Finally pub/sub protocols support best-practice security paradigms.

In light of this, this paper presents the case for a comprehensive embrace of pub/sub data interchange across the renewable energy data ecosystem.

1. Introduction

Increasing signs of climate change are unmistakable. If future disasters brought on by it are to be averted, the world needs to double down on the mission of making our planet's energy infrastructure cleaner, smarter, and more efficient. In this regard, big bets are being placed on **two trends**: towards **renewable energy generation**, and towards **digitization**.

The level of overlap between these two trends is also significant. Namely, modern renewable energy systems often possess comprehensive digital connectivity out of the box. This opens the doors to a promising array of real-world capabilities – promising to ensure that energy is generated, routed, and consumed in the most efficient way possible.

That said, despite a great deal of digitization in the world of renewable energy, **most of the data lives in siloes**, with few ways to easily access it, work with it, or get business value out of it. There is still a dearth of the kinds of open interfaces – or standards – that would be needed to foster a greater level of interoperability and value creation.

There is nonetheless progress in this regard. In AMMP's more immediate ecosystem – which constitutes PV and PV/hybrid systems – in the past couple of years SMA and Fronius have released new, well-designed, REST APIs, while Schneider's is currently in beta testing. Huawei's and Tesla's capabilities in this regard are also growing, and come in addition to existing services by the likes of Victron and SolarEdge, who were early adopters of open data access. These APIs allow easy (though not always free) access to inverter and plant data.

Beyond this, there is an unmistakable broader push for interoperability in energy, with for instance NIST promoting a common interoperability framework – with the explicit objective of moving towards cleaner and more secure energy infrastructure.

Despite some limitations – such as a general lack of standardization among these vendor APIs – these developments are certainly welcome, and they are a big step in the right direction. Yet they are only a small step towards a world of truly smart energy. **To bring this future to life, we need to embrace a paradigm that enables a much *livelier* exchange of real-time data and events. Such event-driven architectures are most likely to be enabled by "publish-subscribe" ("pub/sub") patterns of communication**.

The use of pub/sub communication patterns for exchange of renewable energy data is the topic of this white paper.

The next section presents the most common models for exchange currently in use – including REST APIs – and some of their pros and cons. Following this a primer on pub/sub communication patterns, and potential advantages these present with respect to traditional methods.

2. Prevailing traditional methods for data exchange

This section presents an overview of the most common ways in which renewable energy data is currently exchanged. The focus here is on methods for data exchange over longer distances, and/or between different parties – usually transmitted over the Internet. I.e., the focus is not on data exchange between devices at a single site, where fieldbus protocols are dominant, and APIs are less relevant.

In this scenario there are two relevant messaging patterns:

- **Request-Response**: A client makes a request to a server, which provides a response
- **Publish-Subscribe**: All clients connect to a broker; there they can publish data to topics, and/or subscribe to topics in order to receive data published by others

Request-response is the workhorse of most Internet communications; it is the underlying pattern of HTTP requests, and thereby REST APIs. It is also the focus of this section.

When it comes to exchanging energy data, there are **two ways in which the Request-Response pattern are implemented**:

- **Data Pull**: When a client is interested in some specific data, it makes a request to a data provider's API endpoint, and the provider sends the data in response
- **Data Push**: Periodically, or when specific conditions are met, a data provider will push data to a client endpoint. In this case the data is part of the "request" that the provider initiates, while the client's response is generally just an acknowledgment of receipt

In most cases the "data provider" is the entity that initially obtains the raw data from the site – usually the equipment vendor, or a monitoring service that directly reads out on-site equipment. The "clients" are the parties interested in receiving and working with the data.

Figure 1 presents a graphical illustration of these patterns, and the Pull vs Push methods.

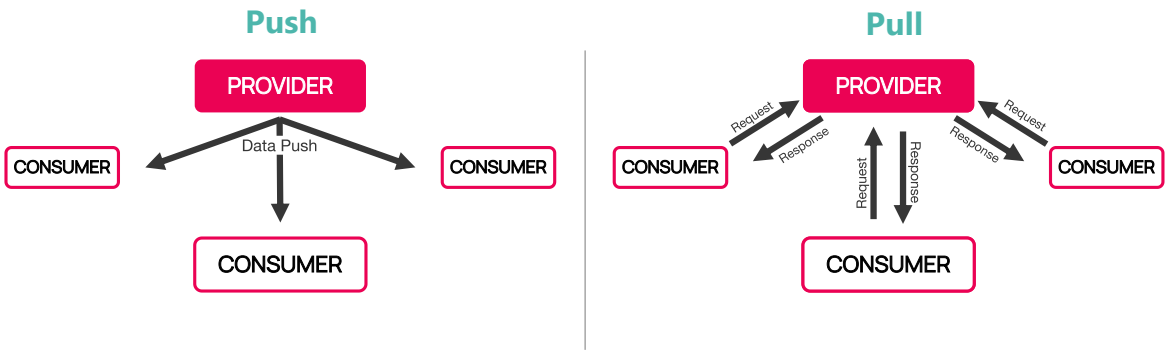

Figure 1: Push & Pull methods with request-response pattern

a. Data Pull

The Pull method is the most common one currently implemented. It is the model for data retrieval that virtually all REST APIs provided by PV and battery inverter vendors support. Through this, each client can make a request such as "what was the PV production at site X for the last day?" or "what is the current state of charge of the battery bank at site Y?"

While setting up a new REST API service can be a fair undertaking for the data provider, on the client side things are generally quite straightforward – given that all the client needs to do is authenticate and make HTTP requests. This is therefore both simple and useful. It enables a whole new domain of connected functionality, and added busines value. **Yet, for anything beyond basic reporting, this model proves quite inadequate. There are at least three major pitfalls, which are fundamental to this architecture**:

1. **Events cannot be detected in real-time**. The client only receives new information if the clients requests it. There is no mechanism for the data provider to alert the client to the availability of important new information or events. If there is a system failure, for instance, the client may not find out for quite some time, delaying potentially time-critical action.

2. **Recent historical data is often unreliable**. This is a more nuanced point, but one that the team at AMMP encounters quite often. Many energy systems are in areas with unreliable connectivity, where communications outages lead to gaps in historical data. Most on-site data acquisition systems do provide a level of local buffering, so the data is not lost altogether, but is simply pushed to the provider's data store with some delay. Therefore, over time such gaps in data often do get filled. However, from the

perspective of a client requesting recent historical data from an API, there is essentially no way to find out if or when a gap would be filled. The best the client can do is to at least note that a gap exists, and keep repeating a data request until the gap potentially gets filled, thereby using up resources on both the client and provider ends.

3. **The data provider's server load scales poorly**. In an increasingly interconnected energy landscape (integrating e-mobility, smart load management, etc), the data in question is likely to be relevant to a large number of stakeholders. Each stakeholder/client will be making its own requests to a given API endpoint, potentially querying data for an increasing number of systems. This workload is further exacerbated by the above two points, where issue (1) incentivizes clients to poll for the latest data frequently in order to not miss important events; and issue (2) incentivizes them to poll historical data repeatedly. While higher resource requirements can of course be catered for by allocating more resources, a REST API simply does not look like an efficient interface for exchanging this kind of data.

b. Data Push

Reflecting on the above, one may conclude that the Push model could provide a decent solution to the points raised. Using that method, the data provider now "has the initiative", and can push data and events to clients as soon as these become available. Evidently, this model largely takes care of issues (1) and (2) above. Meanwhile, issue (3) can be dismissed by virtue of the fact that the data provider is now simply processing and pushing out the same set of data, regardless of the number of clients, or their actual needs.

Indeed, historically Push has been somewhat common in the context of large-scale PV monitoring – specifically as an implementation where each client sets up and runs an FTP server, to which a provider periodically uploads CSV data files. By modern Internet standards, FTP is essentially obsolete – and it was never the right tool for this job anyway. Yet this, in and of itself, is not sufficient grounds on which to dismiss the Push approach as a whole.

Instead, there are several fundamental practical limitations to the Push method, that mean that these days it is not commonly preferred:

1. Each client needs to set up and maintain their own endpoint for the provider to push to. This needs to be made accessible on the public Internet (with appropriate certificates and security), or on a VPN set up between client and provider. Doing this setup – and ongoing maintenance – is often not a trivial undertaking.

2. The data provider needs to keep track of all the endpoints it is pushing data to. Any updates to endpoints (e.g. if a client is added, or moves), generally require manual reconfiguration of the provider.

3. A client has no control over what it receives from the provider. Generally speaking, there is no mechanism for a client to specify that it's interested in data for specific systems but not others, or data with a specific time interval. Making changes like this again usually involves a manual reconfiguration of the provider.

4. If a client temporarily goes offline for any reason, in most cases the provider will not attempt to resend any data that was missed – leading to permanent data gaps.

5. There is no established way for a client to send instructions upstream to the provider – e.g. if the client needs to control a piece of equipment

All in all, data Push is mostly appropriate in fairly static situations; where a provider is sending data to a modest number of organizational counterparts, with mostly fixed requirements, and where all clients have the resources to set up and maintain dedicated endpoints for receiving data. The Push model breaks down in situations with more dynamic client-provider relationships.

3. Pub/sub – or *How I learned to stop worrying and love the data*

In many ways, pub/sub combines the best of both the Push and the Pull worlds, while going easy on complexity. In pub/sub, there is a broker; the data providers connect to it, as do the clients. The providers publish data to relevant topics – such as "system1/inverter3/power". The subscribers subscribe to the topics they are interested in, and receive all incoming data for those topics as soon as any is published by a provider. **As a baseline, that's all there is to it. The providers share the data they have, while clients get what they need – in real time, with minimum setup and resources.**

This is illustrated with a simple diagram in Figure 2. It's worth noting is that while Figure 2 depicts a situation where a provider sends data to a number of consumers, in a Pub/Sub paradigm all broker connections are essentially equal, and all can be bi-directional. So the consumers are also free to publish data – such as commands or requests – that the provider can subscribe to and act on.

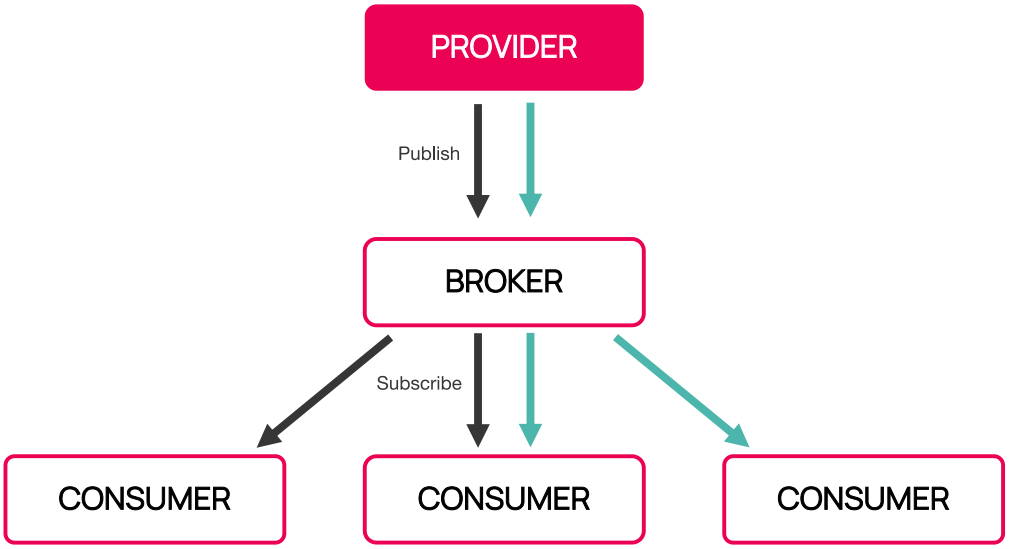

Figure 2: Multiple consumers receiving different payloads via Pub/Sub pattern

Table 1 provides an overview of the three different methods discussed so far: Pull, Push, and Pub/Sub

	Pull	Push	Pub/Sub
Message pattern	Request-response	Request-response	Publish-subscribe
Common protocol implementations	HTTP	HTTP, FTP	MQTT, AMQP
Common payload formats	JSON	JSON, CSV (for FTP)	JSON, binary (CBOR, Protobuf, …)
Pros	- Simple client implementation - Client defines and initiates requests	- Provides real-time data and events - Low resource requirements for provider	- Real-time data and events - Client chooses what to receive - Simple client and provider implementations - Low resource requirements
Cons	- Not suitable for real-time data and events - High resource requirements for provider	- Complex client implementation - Client has no control over received data - Static peer and data exchange models	- Less appropriate for historical data - Not suitable for custom, or synchronous client requests

Table 1 – Pull, Push & Pub/Sub: An overview

Beyond these basics, pub/sub does also offer several additional layers of icing that are worth considering:

- Encryption and authentication options are on par with the gold standard commonly used to secure REST APIs over HTTPS.
- Topic-level authorization/ACLs can be used to precisely define what each connected party is allowed to do. E.g., which topics a provider can publish to, and which topics a client can subscribe to.

- As well as specific topics, clients can use wildcards to subscribe to nested hierarchies
- Quality of Service definitions allow parties to flag how important a particular message or topic is, and the fidelity required when treating the data
- Brokers generally support replaying of messages that arrived while a client was away/disconnected, so no data is missed
- Message exchange is inherently two-way. Assuming the right authorization is in place, a client can also publish, and a provider can also subscribe. In fact, from the broker's perspective, there is no inherent distinction between the two. This allows for powerful control and feedback mechanisms.

60-second primer on pub/sub protocols

There are two main ones: MQTT (Message Queuing Telemetry Transport) and AMQP (Advanced Message Queueing Protocol). MQTT is somewhat lighter and simpler, and more IoT-oriented – i.e., geared towards communication with devices in the field. AMQP has some more advanced functionality (e.g. message tagging and flexible queueing), and is more geared towards general business messaging applications. The functionality we discuss here largely reflects MQTT's capabilities, as implemented in typical open-source brokers such as Mosquitto, with some common add-ons. AMQP and brokers such as RabbitMQ (also open-source) do offer more advanced functionality that may be attractive for energy data applications.

- *Caveats*

  There are of course still use cases where pub/sub does not quite match the functionality of a REST API. The main scenario being a client wanting to request a specific bit of historical data, or a particular report. This *is* still certainly possible through a pub/sub model: the client can publish its specific request, the provider picks it up and process it, then return the result via the broker. Such a model does in fact also allow asynchronous processing of the request, which can often be an advantage where custom reports are concerned. Though it is undoubtedly more complex for a client to implement than a simple HTTP call to a REST API.

4. Conclusion

Given the advantages of pub/sub, broader adoption appears inevitable; though the pace of this is still tepid. In the world of renewables and PV, so far one major vendor – Victron – has fully implemented pub/sub as a means of data provision: with the majority of available system data being published in real-time to their MQTT broker.

While progress is of course often gradual, the benefits of embracing pub/sub are numerous, and the value that the model unlocks across the ecosystem are likely to be immense. This will be especially true as network effects through broader adoption kick in. It's worth noting that influential institutions like the Eclipse Foundation are pushing for standardization on MQTT communication in Industrial IoT, as part of a drive for achieving true interoperability in the space.

The tools are already in our hands. We just need to put them to use – in order to shape a cleaner, smarter, better tomorrow!

About the author

Svet Bajlekov is co-founder and CEO at AMMP Technologies. He has spent the last three years helping emerging energy companies derive value from their operational data. He previously served as CTO at E.ON Off Grid Solutions, where he led work on PV mini-grid deployments and operations across Sub-Saharan Africa, as well as the development of innovative software solutions for off-grid energy systems.

Svet holds a PhD in Atomic & Laser Physics from Oxford University, and an MSci in Theoretical Physics from Durham University.

He can be reached at svet.bajlekov@ammp.io

AMMP Technologies — Who we are

AMMP provides digital solutions for energy service companies in emerging markets. Our vendor-agnostic SaaS platform offers remote monitoring and management for portfolios of renewable/hybrid energy systems. AMMP allows energy companies to obtain an integrated view of asset performance across a range of vendors and technologies—such as solar PV, batteries, diesel generation, and smart metering.

Through this, AMMP allows operators to streamline their processes, improve response times, cut down on-site visits, and significantly reduce O&M costs. The platform has been refined over years of on-the-ground use in operating complex assets in remote locations.

Find out more at www.ammp.io, or write to us at contact@ammp.io