Adventures in Alternative Energy
Data Monitoring
Goal

To monitor energy generated by thousands of green energy installations across the USA by creating a hub for energy, reliability, and analytics data.
A monkey wrench

10,000 devices post data at 15-minute intervals at 1-minute resolution.

Additional data-driven feeds send second-by-second data

Data needs to be stored for years for reliability analysis
Not your grandaddy’s app

This application can’t be classified as a *pure* reporting application, data warehousing app, CRUD app, or otherwise.

Because it is a hybrid, the app raises compatibility issues with schema design and Rails.
Technology stack
Ever hit a bottleneck?
Many devices report at the same time

It’s gotta be fast

Writing directly to the database is too slow,

10,000+ connections!

Shouldn’t be IO bound

Only occasional data loss is acceptable
What's up with that router?

Generating solid test data is often overlooked

Router moves posts to the appropriate environment. Can be duplicates or data subsets

Router needs to be very lightweight and fast
Testing the stack

Shops tend to skip stack-testing

- A simple db dump in a dev environment is sometimes not enough
- One or even a few pieces may pass individual tests, but these are worthless without testing the application as a whole
- Real test data is often an issue
- Many ways to solve this issue
Data posts have value

With data in the queue we have to decide what to do with the XML

Not all of the XML is needed for display or reporting

XML may vary greatly from one model to another

Engineers need access to the raw posts
Queue is not persistent.

We want all data in the queue in memory, so we need our solution to be fast

Can not wait for complex transformations
Log DB

Data goes into the Log DB first, which acts as a message log to

- track XML changes
- give visibility into posted data
- rebuild operational databases when necessary
- log data waypoints from different pieces of the stack
Proper logging and instrumentation is critical to long term success

Engineers need as much information as possible from units in the field

Changing field conditions wreak havoc on debugging

A central location for logging makes viewing system-wide issues much easier
XML in MySQL

Why use MySQL for Log DB?

NoSQL may be a great option, but not ideal for searching raw XML

Flat files may work

MyISAM was chosen for simplicity. It enables the ability to path into XML, and row headers enhance queryability

Archive storage engine in future
Querying raw XML is as bad as
Keep it simple

Consider hidden costs of resources

The latest and greatest is not always the best solution to your problem

Occam’s razor. Always work to reduce complexity

In our situation, MySQL met our needs.

Other tools may meet the needs and increase flexibility, but the benefits of reduced complexity outweighed more flexibility.
A simple batch ETL process

A series of Ruby and shell scripts make up the ETL process

The ETL transforms and modifies several pieces of data from the XML before putting into a normalized structure
No ETL Tool?

This added another layer of unnecessary complexity for us

Higher learning curve

Easier to start small writing exactly what we want in Ruby
Data modeling challenges

We have to duplicate some data to try and enhance the report ability of certain data points.

Some fields are bit-masked, which makes them a bit of a challenge to plan for.
Bitmasked fields

MySQL allows for a SET data type which can be handy, however frequent additions can be painful.

Storing raw bitmasked fields in a bigint can be handy, however realize that these cannot be effectively indexed.
Some problems

Single-threaded implementation

Data transformation occurs at scheduled intervals not at arrival time
How much data do you really need?

Don’t assume that you’ll need the data forever.

Determine granularity, lifespan, and data subsets.
This will be huge

Long term this could be 100's of gb if not Pbs

10k units reporting every minute of every day for 10 years, with hundreds of metrics per unit
Details

The Operational DB

Highest resolution data needed for a short time span. It consists of hundreds of metrics and is used for technical support.

The Historical DB

Long-term data only requires 10-15 metrics. Having a LogDB means we could retrieve more metrics if the need arises. A less-granular data resolution is good enough.
Details

Populating the Historical DB

Data is replicated from Operational DB to the Historical DB on a 90-day rolling window.

Triggers fire on insert to move a number of columns to long-term tables.

A 90-day purge on the Operational DB keeps it clean and fast while keeping the long-term historical data intact.
Details

Benefits

Storing data separately gives the company longer-term metrics at a lower data resolution.

This long-term data can be used to:

- analyze hourly historical performance
- view weather data across multiple days
- find trends in peak performance based on geography
Correlating data

When you give users something cool they always want more

We started building small data marts for items like fault data that correlate our monitor data with other systems

The small one-offs have grown
Data warehouse

Coming soon

To merge data into a unified system from sales, support, marketing, and Q.A.

This system will look for power trends

- What geography offers optimal power generation?
- Which firmware releases are most reliable?
- Which installations and models are most efficient?
- How does adverse weather impact power?
Data warehouse

Coming soon

To drive smarter power distribution

- Plants can predict amount of power needed to augment alternative energy sources
- Mini datamarts collect and aggregate valuable data
- Reliability engineering teams can use field data about faults to drive better products to market
Front end

Built with Ruby on Rails
Pulls from multiple db servers
SOA for all dbs
Data is retrieved via RESTful API calls.

APIs pass back JSON which is rendered using the Google Visualization API.
Ruby on Rails challenges

Active Record pushed to its limits
Select * everything
Bad joins
Lots of unneeded queries at page load (solved by eager loading)
Views
Ruby on Rails challenges

Views support in migrations, schema dumper, and testing

Multi-db support not native

Clone operations want tables to be Active Record-y by default (pk int auto inc)

Sometimes moving to a NoSQL solution fixes joins and other crazy things that Active Record does
It’s okay

To use find by SQL
To look for third party plugins to help
To not use Active Record
Active Record

is not great when

using a custom schema
data warehousing
doing complex reporting
Rails developers love simplicity
hate thinking about the db
prefer to think about models and objects
have reason to love NoSQL