IntelliTect

Cloud-First AWS Solution for Energy Company's Critical Integrations

CASE STUDY

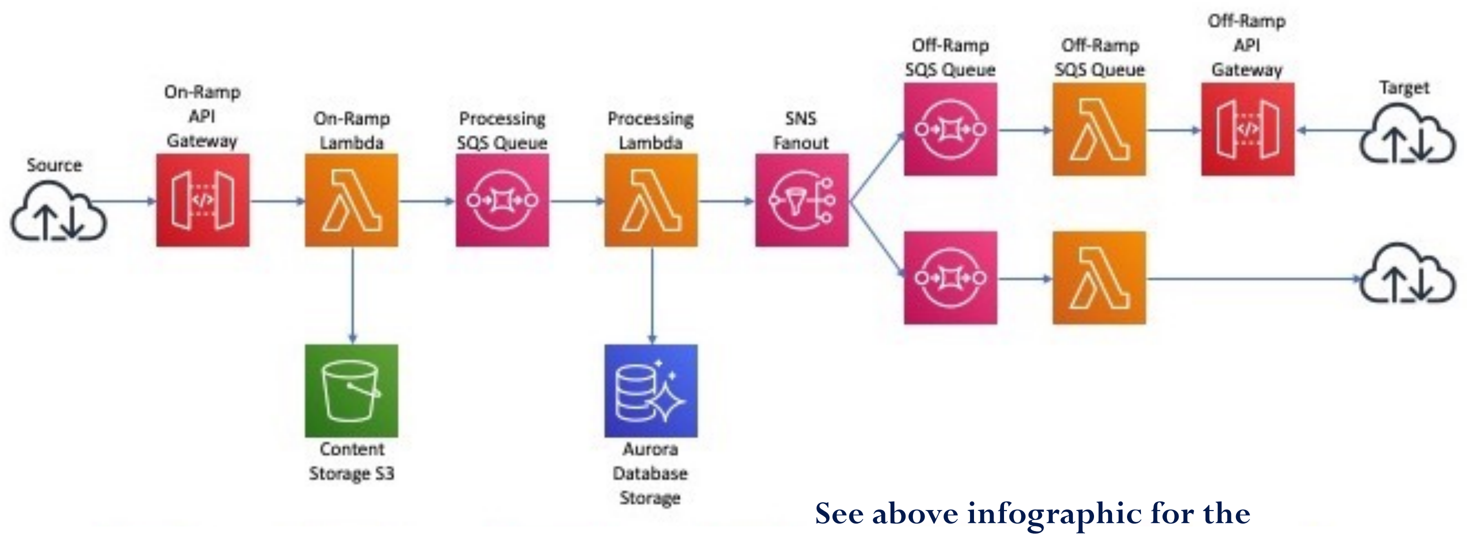
“ IntelliTect has been a joy to work with. They are more than capable, professional, flexible, and have helped us achieve a level of technical sophistication that I truly don't think we could have achieved on our own, in such a short time. ”

Senior Developer,
Fortune 100 Energy Company

Challenge: Integration Strategy

To consolidate information technology resources and streamline and enhance business operations, an American multinational energy company selected several Software as a Service (SaaS) applications to support their business needs. They needed an integration strategy for exchanging data with these new SaaS components and their existing and newly built facilities and tooling. The energy company requested a cloud-first approach that would be scalable and resilient. This approach would need to support many future features and eventually replace their legacy system.

The energy company selected the IntelliTect team for this project because they previously assisted in moving to Azure DevOps for their internal development projects. IntelliTect also provided key resources for an internal sales web application that was also hosted in Amazon Web Services (AWS).



See above infographic for the architecture's various AWS components.

A Cloud-First Solution

IntelliTect met with the energy company's team and gathered requirements for the project. The existing team already had an investment in AWS and had made significant strides in operationalizing cloud resources. It was clear that a core set of AWS services would meet the needs of this project in a streamlined and cost-effective way. The core cloud-first solution broke the process up into three phases.

Phase 1 – On-Ramp: Load the data from the source and queue it for processing

This solution takes data from a source and loads it with a Lambda on-ramp. There are many different types of on-ramps to handle the various formats and delivery mechanisms from both internal and external providers. The data is then transformed and placed onto a Simple Queue Service (SQS) to be processed. This process also handles any required batching and debatching operations.

Phase 2 – Processing: Validate, store, and massage data as necessary for consumption

Next, the processing Lambda picks up the message from the queue and checks it for validity. A failed validation is met with extensive log tracking and triggers a process to alert the operations team. If necessary, the data is then augmented and saved to the database for reference and potential use by subsequent messages. Once the Lambda completes processing, the result is sent to a Simple Notification Service (SNS) to fan out the message to all the recipients. This subscription model allows for additional consumers to be easily added to messages. From SNS, the messages are placed on a queue that holds them until they're delivered to the target systems.

Phase 3 – Off-Ramp: Fanout with queue to send the data to target systems

Finally, the off-ramp Lambda picks up the messages from the off-ramp queue and will either send or make it available to the target system. This may be a push to a SaaS system or a data lake. It could also make the data available via a call to an API Gateway endpoint.

Using queues between each major processing component ensures that the system won't lose data even if one system goes off-line.

Results:
A Robust Infrastructure

Deployment through the various staging environments was seamless and provided confidence for the production release. Consequently, the solution scaled effectively and clearly displayed how the serverless components ramped up to meet the operational load's needs. In the end, the project provided a robust infrastructure on which future integrations could be readily built.

Additionally, by choosing cloud-native, serverless components, features like scaling and resiliency are built into the architecture. Scaling across a region is easy to implement and works seamlessly. By using Terraform to stand up the AWS infrastructure, standing up another instance of the system is very simple in a disaster recovery scenario.

All code was written in C# using .NET Core and runs in Linux-based Lambda functions. Using Docker, unit tests and integration tests provide the team with a robust environment that delivers quality results.

This project was unique in that the energy company was very open to cloud-first solutions and was willing to fully adopt a serverless mindset for their architecture. As a result, this approach addressed and solved many difficult operational issues like uptime, resiliency, security, recovery, and scaling through the fundamental architecture. This strategy resulted in a solution that will serve their company for many years to come.

IntelliTect and the fortune 100 energy company continue to partner on cloud-based projects.

Skills and Services

Amazon Web Services (AWS)
Cloud Strategy/Services
ALM/DevOps
Integration
Architecture
Analysis
Mobile/Web App Development
Project Management
Software Development

Want More AWS?

Go to intellitect.com/cloud-migrations to learn how IntelliTect can simplify your company's transition to the cloud.

For more about IntelliTect's development process, go to intellitect.com/developmentprocess.