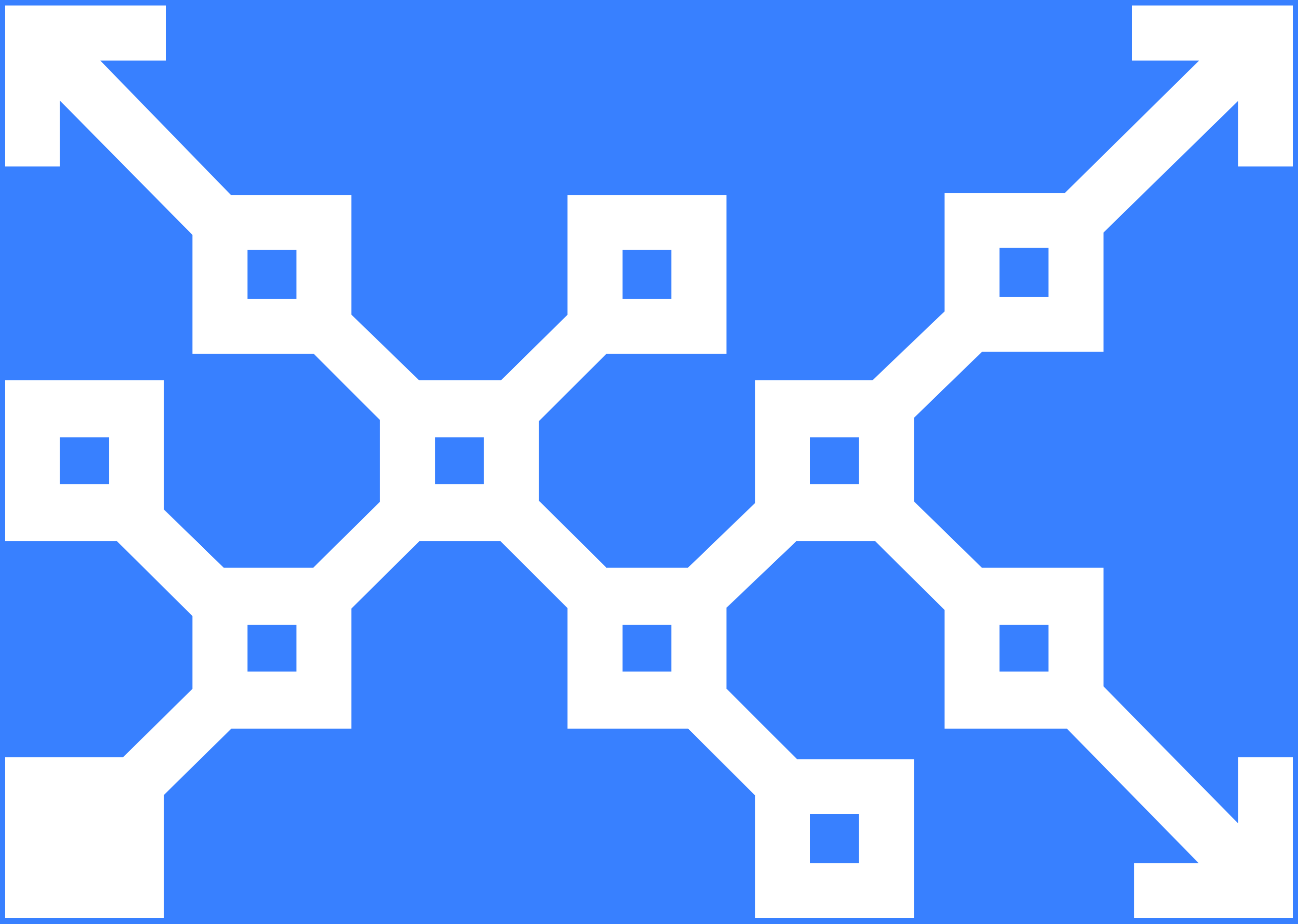
WHITE PAPER

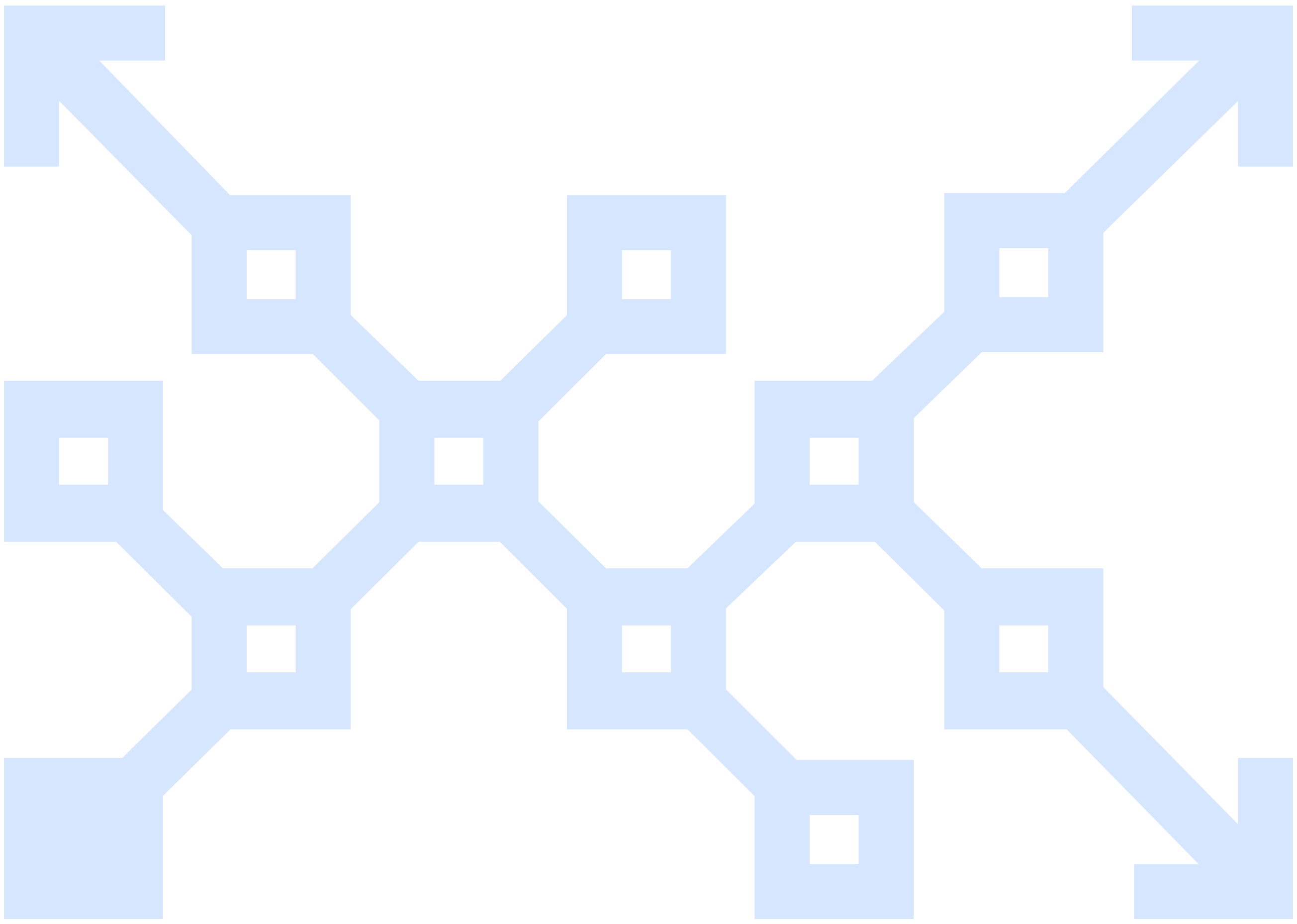
Cutting-Edge IT: Moving from Nodes to Applications at Wells Fargo



puppet

Guest Author: Nate Loomis, DevOps manager at Wells Fargo

Contents



Intro

At Wells Fargo, our top priority is helping our customers succeed financially. This has a direct impact on how the bank manages technology and the applications with which our customers interact.

For the IT team at Wells Fargo, “managing technology” means striving for constant innovation to bring the company closer to its customers. From Wild West stagecoach routes to online banking, Wells Fargo has always been willing to innovate if doing so helps its customers’ financial success.

The results of the bank’s innovation are self-evident: Wells Fargo is the world’s No. 1 bank by market capitalization. With 265,000 employees worldwide, 17,000 are dedicated to IT. The IT teams manage an infrastructure that supports **1 billion** transactions per day.

To support such a large volume of transactions, the bank maintains a robust infrastructure that includes:

- A server footprint of 120,000 nodes
- 55,000 network devices
- Over 2,300 applications in production
- Thousands of products in active use
- 4,000,000 configuration items

All of this means that we have over 6,000 ways that we deliver change to a server.

With an infrastructure of this magnitude, the Wells Fargo IT team needed a simple and consistent solution offering complete and shared visibility into each change. More importantly, in a corporation the size of Wells Fargo, we needed to plan carefully because missteps in an application’s architecture could live on after even the application itself.

For IT to create true business value, you must be able to drive change at the application level and relate it back to the specific business process.

To implement a design that started with a business solution and ended with a way to manage change for all the bank’s applications, the IT team embarked on a DevOps journey. This paper discusses that journey, and how the team implemented a design, using Puppet, for deploying applications that deliver business value to Wells Fargo customers, and to the corporation itself.

IT exists to serve the business

Today, IT is often built around technical solutions: the hardware and operating systems that make up an infrastructure. This creates a limited view of the customer experience and, consequently, takes little note of business values and goals.

The Wells Fargo team knew we had to move away from organizing around the physical bits of tin. Manual handoffs based on change requests and work orders were time-consuming and costly. To get ahead of the game and drive customer satisfaction (the key to delivering business value), we had to change the way we thought about managing and delivering services. The applications and services we provided had to be re-oriented around the bank's business goals, not the tin.

What we needed was a common contract with the rest of Wells Fargo.

The application is the common contract between development, operations, governance and the business.

The common contract

The IT team needed a vision for how DevOps would mature within our own team, and we needed to plot a course that expressed this vision. At the outset of the DevOps journey, there were multiple teams managing change to any given server, and sometimes changes would overlap. Consider, for example, the implementation of Tomcat: At any given time, the team could have one set of configuration values coming out of a patching cycle, and another set of values applied in an application release.

Though the team did exercise a strict software development lifecycle, it wasn't uncommon for different teams to have their own lifecycles. Because of this disparity, multiple teams might manage change to a server, but often the change (and its effects) wasn't actually visible until the server was already in production.

Application managers weren't free of pain either. They had to fill out form after form after form just to get an incremental change applied.

What we had was a complexity of process.

In today's interconnected world, you can't afford to run applications that aren't up to date; if you do, you're probably vulnerable to a security flaw somewhere in your application architecture. A key driver of change and velocity at Wells Fargo is our team's desire to keep up with patch-level security. To remain secure, we can't live at the bleeding edge — instead, we need to be at the cutting edge, ensuring that our applications are kept up to date as quickly and efficiently as possible.

To meet this standard, our IT team needed to improve collaboration, speed, and consistency. We needed to get all technical teams coordinating on a common set of artifacts.

To achieve this goal, we shifted to managing our application infrastructure in a declarative way with Puppet. Teams involved with a given application would now coordinate on a common set of artifacts to describe that application, ensuring that all changes were versioned and visible to all the teams involved.

The new collaboration and consistency across teams — from security to compliance to development — ensured that any configuration overlaps were caught during the design phase, before applications went into production. Most importantly, the IT team realized that groups of servers hosting applications needed to be identified by their business value — not just by hostnames.

We'd already adopted the notion of servers as cattle, not pets, and that drove more focus on the application ID as an immutable identifier for infrastructure supporting the application. The application ID usually defaults to the application that the business partner uses, and other systems and dependencies are in the background.

While the benefits of restructuring applications as sets of microservices were clear, we knew some encapsulation would still be needed to provide clarity across development, operations, governance and our business partners. If servers — derived from a static list — were transient and immutable, how would the team report on licensing, or prove compliance with corporate policies? Replacing hostnames with application IDs enabled us to match infrastructure to specific business functions, resolving these issues.

By creating models of our applications aligned with business function, we could manage the state of the key components necessary to deliver on the application's business value. The team moved to working with declarative, machine- and human-readable artifacts — often referred to as *executable documentation*. Taking advantage of the abstraction Puppet provides, we were able to free ourselves from thinking about our architecture as hardware, and focus instead on the APIs and services we needed to provide value to customers.

Puppet's declarative language enabled the team to use data from our application models to provide several improvements:

- Enhanced portability for our applications.
- Better forecasting of changes headed towards production.
- Additional context for the **coarse-grained APIs** in our service layer.

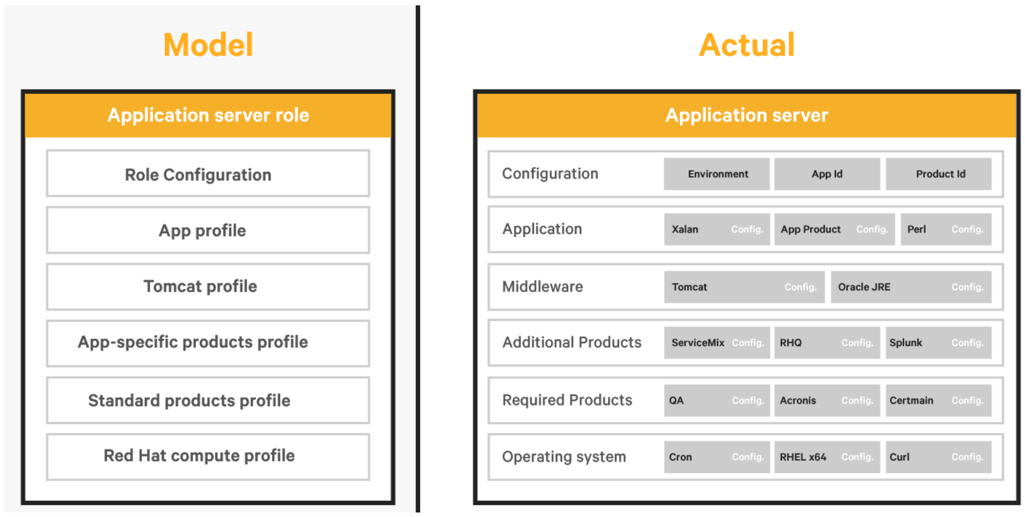
This added insight enabled us to move from a node-centric (or hardware-centric) view to an application-centric view, and begin modeling applications that reflected business processes and priorities.

The node view

At Wells Fargo, a node has one role, and the state of that node is managed by Puppet. Each node is built with a set of profiles, and each profile represents a layer of the software stack fulfilling a specific technical function.

A node's profile contains reusable modules that allow the team to construct different arrangements of servers. Further, each profile has a base configuration that is specific to a package or product. This allows the team to use class parameters via Hiera (Puppet's key/value lookup tool for configuration data) for defining how an application could configure and use a product.

We were also able to pre-harden these products according to corporate security baselines. We used Hiera and class declarations to establish both product configuration and application-specific configuration, creating a key delineation of state between the reusable elements and those elements that were configured for a specific application.

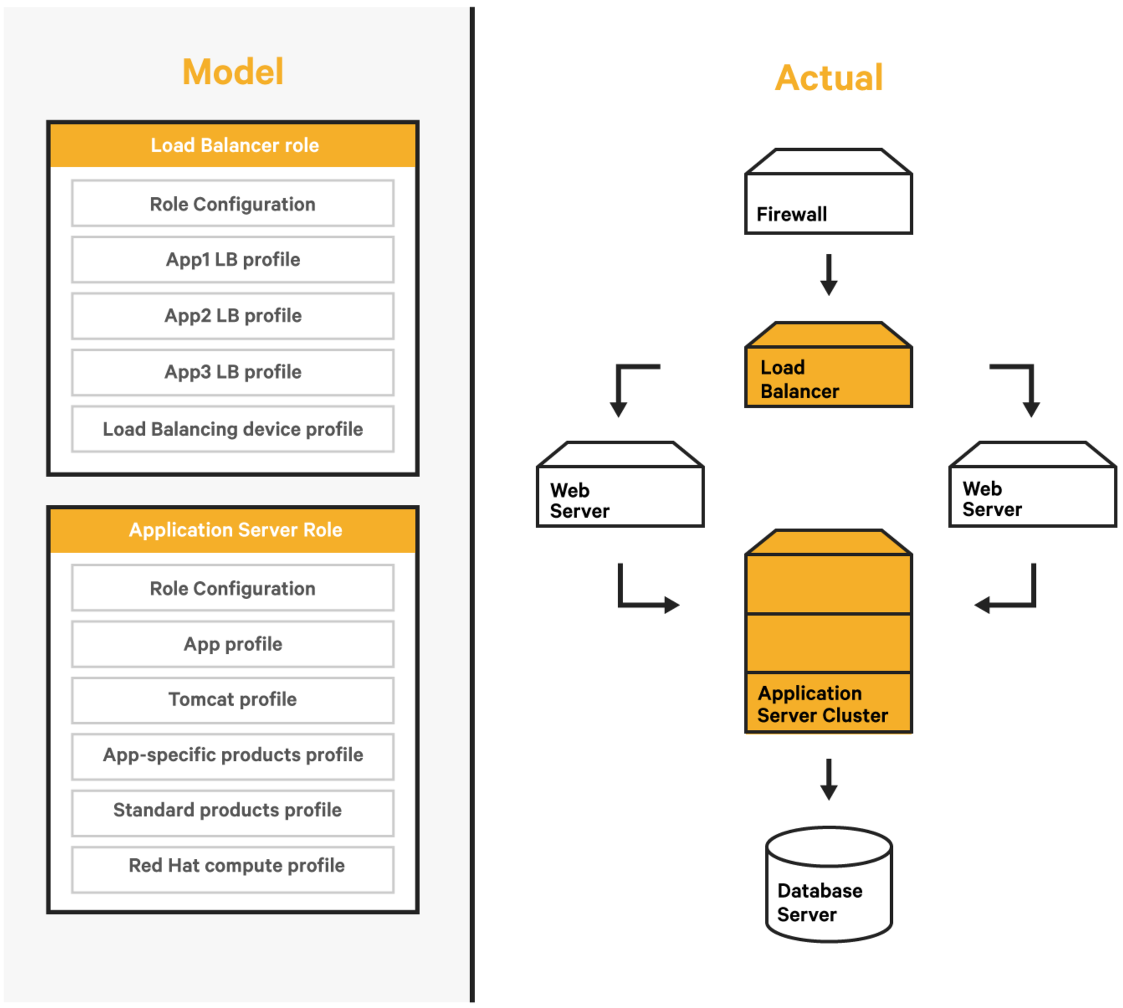


While the node view achieved its business function, it didn't really deliver business value, because it didn't deliver a complete business process. The team needed to effectively model *all* the components of the application in order to deliver business value.

From node view to application view

We began looking at other components of the application stack. Could we use the same business-centric approach to model the web server, the application server, the load balancer, all as separate roles? The team tested these application models against 250 different scenarios put forth by our business colleagues — and met with astonishing success.

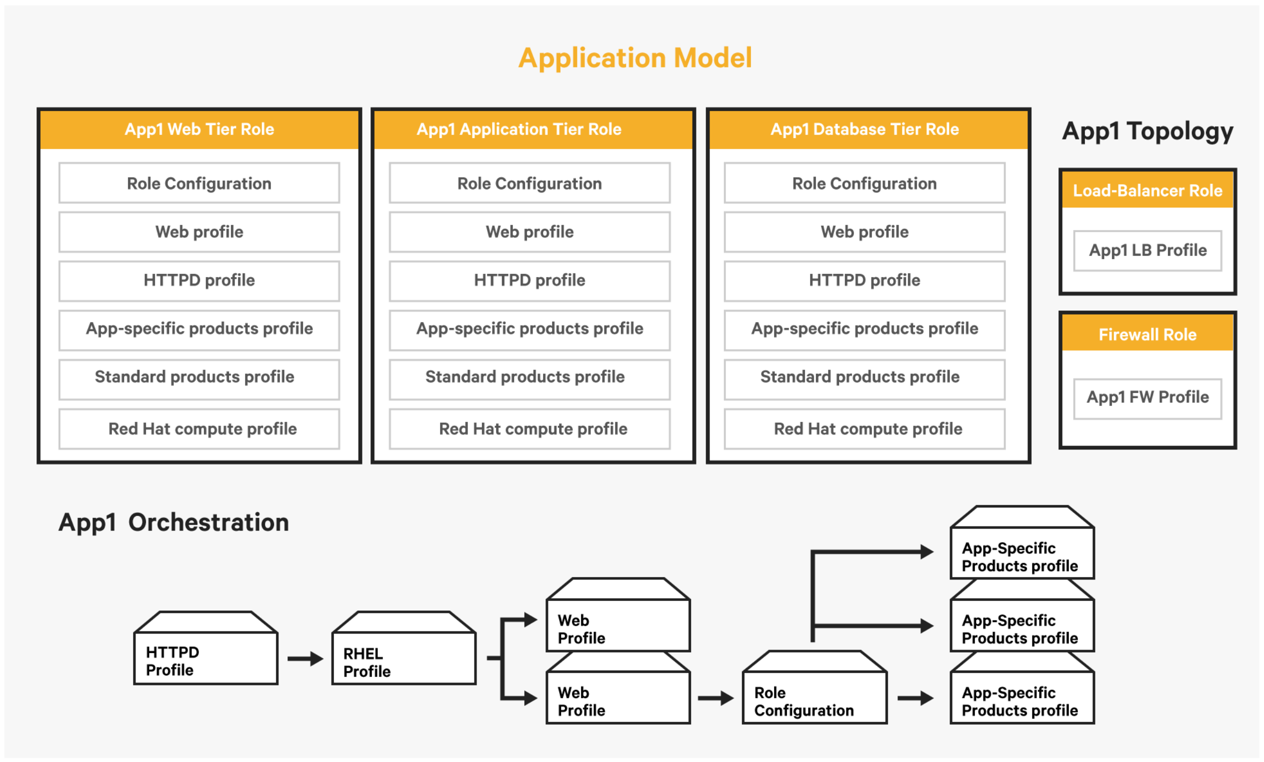
Before adopting an application-centric view, it would take months to get a server and deploy an application that the business could use. After adopting the application-centric view, a business user could request a server, go get coffee, and by the time they got back to their desk, find the server had been provisioned, deployed, configured for the application, joined to a cluster, and was now accepting traffic. What used to take months now takes minutes.



Modeling the application and the business process

Today, Wells Fargo has roles representing entire stacks of business function: one stack for the web server, another for the application server, and another for the database. The team also has services such as load balancing as a service, which leverages a standard role and profile to provide a load balancer serving many applications.

The biggest gain, however, is that we now have an orchestrated process for delivering change. This process is comprised of transparent APIs and backing services that enable the team to report change, update the change management database, create software orders, and report on asset use — in near-real time.



Now that we have these application models, we are ready to check out how portable this model is, specifically in a cloud-broker setting that would allow the team to move applications from one cloud to another. We are interested in seeing whether application modeling can help us keep an application separate from the implementation schemes that handle the necessities of each cloud broker.

In addition, the IT team wants to help our asset management partners with license tracking. Products deployed in cloud environments must have the correct licensing terms; otherwise, a company, and especially one as large as Wells Fargo, can incur large and unnecessary costs.

Application portability and managing costs are just two examples of business needs that IT can now fulfill so much better than before. Another incredibly useful gain from application modeling is the ability to report on change *before* it happens, as opposed to reactively tracing and analyzing what happened after the change has taken place. For our IT team, having the ability to view, measure, and report on a change before it gets to the server is critical. Now we can predict and report on change using Puppet's simulation capability, known as "no-op."

Running Puppet in no-op mode is one element of how we at Wells Fargo interpret the CALMS model: Culture, Automation, Lean, Measurement, and Sharing. Running in no-op mode gives our IT team the ability to both measure *and* forecast change in our software supply chain. We now have more visibility into the lifespan of a server than ever before. Governance and compliance can see change headed to the server *before* the change ever occurs. This is very valuable for other industries besides banking, since any company can plan better when it has the ability to predict.

The security team at Wells Fargo benefits as much as every other team. Vulnerabilities can now be tracked down and removed from infrastructure quickly and completely. More importantly, however, analyzing the application model even before the application is put into production can reveal potential vulnerabilities.

Forecasting is difficult, but as model-based applications mature, it will become easier and even more useful. Our security team now realizes the data they will obtain from forecasting could change the way IT defines services. Business processes used to be delivered by a single, monolithic application; now business services are increasingly dependent on a set of orchestrated services, whether these are microservices within a Wells Fargo application or just part of a standard service-oriented architecture (SOA). When an application is deployed, one change can impact downstream systems and services, potentially breaking that application. But modeling the entire process enables all the technical teams at Wells Fargo to see where a change could cause issues, and see the issues before the change is deployed to production.

All this testing and proactivity may seem complex and daunting, but it doesn't need to be. It can be part of business as usual.

Take an API-first approach

Many companies turn to DevOps in order to align IT better with business needs. When you build model-driven applications to meet business goals, you should take an API-first approach, as many savvy companies do today. The success of **Intel's Internet of Things (IoT)** strategy is dependent on strong APIs. The CEO of Apigee believes we're in a **new era of API-driven design**. **Oracle has extended its API management suite** to capitalize on growing revenue opportunities. **IBM Bluemix allows companies to discover how other developers are using their APIs**, and design around that feedback.

As you consider how to put your APIs forward in your service layer, it's best to ensure they are coarse-grained and drive the change to the data provided, and not directly to the API. Ensure that they accept a data structure, because if you can provide the data structure to the service's API, you can give that service, and potentially other services downstream, the context they need to function properly.

The key is to allow your API to change without having to force additional change to your orchestrated service layer. This allows you to change the data structure, add an additional value, find the service that needs to interact with that value, and write the code there — all without having to assess how that impacts every API of every downstream service.

Working this way lets you form a more semantic relationship between you and your customer, and allows the technology to act as a transparent medium through which you interact. And by evolving the data structure without having to change the entire service layer, an API-first design allows you to grow and evolve your service layer.

Summary

For Wells Fargo, managing technology means constant innovation to bring the company closer to our customers and to help them succeed financially.

Our DevOps journey at Wells Fargo started with the needs of the business, and ended with a new way of managing applications by making the application the lens through which infrastructure is architected and managed. This gave our IT team several capabilities it never had before:

- Enhanced portability for our applications.
- Better forecasting of changes headed towards production.
- Context for an API-driven service layer.

When you think of delivering value to your business, think of how we applied the application-centric model to our infrastructure planning. Your organization can do that too, and even if you aren't the world's biggest bank, you can forge a direct relationship between IT and business value. And just like it did for us at Wells Fargo, Puppet can help you get there.

About the author

Nate Loomis manages the automated configuration engineering team at Wells Fargo and has worked with Puppet for about three years. Nate focuses on creating coarse interfaces to an event-based architecture with the flexibility to deliver on Wells Fargo's complex requirements. He has been programming since he was 7 years old, and has worked as a developer, an architect and in many other roles over his career. For the past decade, Nate has focused on deployment automation and configuration. He has a passion for tinkering with technology, and is a strong supporter of the open source community. Nate lives in Charlotte, N.C., with his wife and two kids.

Learn how you too can drive change with confidence, and easily orchestrate ordered deployments across your infrastructure and applications: Read about orchestration with Puppet.