Developers

Salesforce Developers Blog

Integration Architecture for the Salesforce Platform

GREG COOK

Editors Note: This post is part of a "Guest" series entitled Enterprise Architecture with Force.com. Our guest blogger, Greg Cook, is a managing partner of CloudPremise and currently holds all seven Salesforce certifications.

Editors Note: This post is part of a "Guest" series entitled Enterprise Architecture with Force.com. Our guest blogger, Greg Cook, is a managing partner of CloudPremise and currently holds all seven Salesforce certifications.

What are the components of a good Salesforce Integration Architecture? As a Salesforce Architect it is your role to lead your company in the evolution of it's Integration Architecture. A good architect must understand both integration architecture and integration patterns. The difference between the two is analogous to designing the highway versus driving cars on the highway. The Salesforce1 Platform offers architects and developers a wide array of integration technologies and recommended patterns (the cars). However, without the correct Integration Architecture and technology infrastructure (the highway) your projects and solutions will be at risk for performance, scalability, data integrity, and many other problems. This post shares patterns gathered from my experience in creating an effective Integration Architecture for clients, and shares a reference design drawn from many of these. You should also consult the official Salesforce Integration Patterns guide when architecting your Salesforce solutions.

The best Salesforce Architectures are not based upon incumbent technology, singular architecture approaches, or corporate politics. The best Salesforce Architectures are based upon delivering business value. What this means for the architect is to focus on what are the business's requirements, roadmap, and needs for which you will offer technical capabilities. In other words – you need to see where the business wants to drive, and figure out which highways and roads are necessary to support the amount of traffic. Idealistic architecture (for example, 100% Services Oriented Architecture) may cripple your ability to provide the capabilities needed by your business when they need them.

The Integration Architecture supports a Mix of Batch Processing and Real-time Services Middleware

Good Salesforce architects have learned that the best integration designs support both batch and service-based patterns. This means having multiple types of middleware at work. I have had clients with three to four different integration platforms in their Salesforce architectural landscape. This is because no single solution can effectively meet all of your requirements, and once again the idealistic architectures are not as important as supporting the business's needs.

The Integration Architecture is Based Upon Business Service Level Agreements (SLAs)

A mature organization and architect will attempt to define SLAs for data and process integrations. These SLAs have an important role on projects as they may drive a few miles you do not need a highway. However if you are going on a road-trip I hope you aren't taking side-roads! Define your solutions based upon your business's service-level requirements.

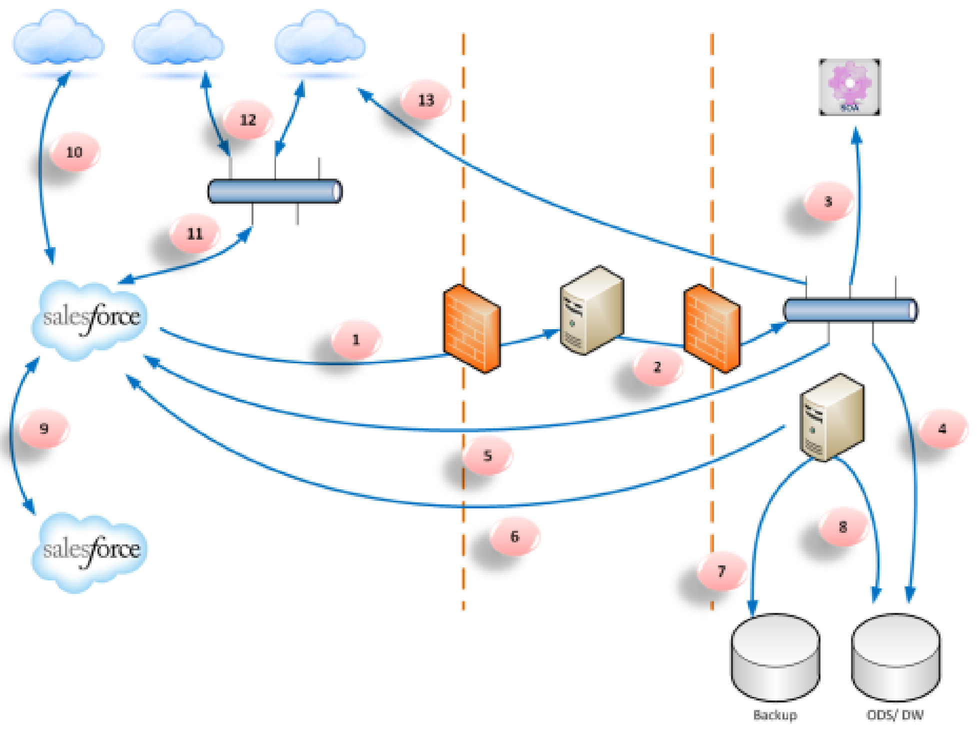
One important aspect of managing SLA's on the Salesforce1 platform is to understand if and when to use asynchronous processing. The multi-tenant design of Salesforce makes it impractical to use asynch patterns when conforming to SLAs. A batch or @future job may not finish processing as quickly as needed, and there is very little you can do about that (apart from adjusting the SLAs). But as CCE Principal Architect Bud Vieira points out, "This may depend on whether your SLAs are determined simply by response times of individual queries, inserts, deletes and updates (as in a transactional process), or throughput of an overall operation. For example, time to complete a stock buy versus how quickly a sales account realignment process can be completed."

The Integration Architecture Has a Clearly Defined Standard for Applying Different Integration Use Cases

As your landscape evolves and your Salesforce expertise matures, the goal is to define a set of capabilities and standards for all Salesforce integrations at your company. Each project should not have to define when and where to use what technologies, how and when to authenticate, etc. These architecturally significant designs should be standardized for your enterprise. This is where a Center of Excellence or Architecture Review Board comes into play. Each Salesforce project should be subservient to a higher level integration

A Typical Enteprise Salesforce Integration Architecture

Let's take a look at a reference Salesforce Integration Architecture. This may or may not look like your existing landscape – however this reference is based upon years of work at many Fortune 500 companies. The reference design also does not recommend one technology vendor or solution over another – rather the goal is to understand the technical capabilities that you can (and probably should) consider as your Salesforce Platform landscape matures.



Let's take a look at the most common integration use-cases and how they apply to your Salesforce Integration Architecture. The direction of the arrows in the reference model is not necessarily the way the data is moving, but rather the way the integration connection is being established. This is a critical aspect of

Cloud-to-Ground (Salesforce Platform Originated)

In Cloud-to-Ground use cases you are attempting to push a transaction (message or data) from Salesforce into your On-Premise infrastructure.

Capability #1 – The Salesforce originated message is relayed to a DMZ (demilitarized zone) service end-point. This could be a firewall, a services gateway appliance, or reverse proxy. You must work closely with your security team to define this layer as opening the corporate firewall to inbound web traffic is a high security risk. This is where much (if not all) of your security authentication from Salesforce Platform occurs. Whitelisted IPs, two-way SSL, and basic HTTP authentication are some of the ways to authenticate Salesforce into the DMZ layer.

Capability #2 – The message is relayed from the DMZ security zone into the trusted On-Premise infrastructure. The message is usually destined for an Enterprise Service Bus (ESB) and durable message queue. The ESB also would handle any transformation, mediation, and orchestration services required by the detailed integration requirements.

Capability #3 – Depending on your Enterprise Architecture the ESB maybe pushing the message into the SOA infrastructure. These web-services are providing consumer agnostic data and business process services to the Enterprise. The Salesforce Platform can become a consumer (and later a producer) of these SOA services. By reusing existing SOA web-services you can save your project a lot of time and money as opposed to integrating directly into the source system. If you do not have a SOA layer your project may be responsible for integrating directly into the legacy application.

(including an ODS – an operational data store). Most commonly (but not always) in a Cloud-to-Ground scenario this transaction would be a database READ. The Salesforce Platform can read data from the database in real (or near-real) time.

Ground-to-Cloud (On-Premise Originated)

In Ground-to-Cloud use cases you are attempting to push AND pull data from Salesforce from your On-Premise infrastructure.

Capability #5 – A mature Integration Architecture should be handling all of the real-time calls into Salesforce from the ESB. However if you do NOT have an ESB, this step would occur from each separate application requiring access to Salesforce. From a security standpoint it is much better to handle all of the calls to Salesforce from a centralized integration middleware. You can use oAuth or user/pw session based authentication to Salesforce. The middleware may already have a session with Salesforce so that you would not need to authenticate again for every transaction.

Capability #6 – Many integrations can be accomplished in a batch design. This is often the cheapest and fastest way to get data in and out of the Salesforce Platform. I would recommend a robust ETL solution for all Salesforce environments. The role of the ETL is to move large data volumes using the Bulk API where possible.

Capability #7 – As a Salesforce architect you have a responsibility to your company or client to off-load your Salesforce data into a replicated copy. My argument for this is that while Salesforce's database is not likely to have outages (who may or may not be able to restore it exactly as necessary). A key architecture choice here, although it may seem trivial, is to make sure these kinds of replications are processing diffs, and never whole tables whether records have changed or not. It's a scalability killer, and so needs to be thought out up front.

Capability #8 – The ETL is also responsible for moving data in and out of your database infrastructure. Often data is necessary to be staged into Salesforce (Accounts for example) from the EDW. Also pulling data down from Salesforce into your EDW may be much easier when done using batch processing patterns.

Cloud-to-Cloud

Capability #9 – If you have multiple orgs (Single-org versus Multi-org Strategy) you will often have the need to integrate between the Orgs. Salesforce makes this (sometimes too) easy via Salesforce2Salesforce. You can also directly contact another Org via RESTful web services integration. The Salesforce Platform's road-map includes the ability to consume other org's data via Hub/Spoke which may also be a good way of providing read-only access across your org landscape.

Capability #10 – The Salesforce Platform's robust integration technology makes it very easy to integrate point-to-point with other systems. While it would be recommended for some solutions (Google Maps mashups, etc) I would try to stay away from using Apex as your primary integration technology. While it is more than capable at handling transformation, error handling, and retries – these types of requirements should be pushed to middleware if possible. The more that Salesforce is made to be the hub of integration activity, the more time

Capability #11 – Rather than using the Salesforce Platform to be your hub of cloud-to-cloud integration activity many companies have moved towards Cloud based Integration-as-a-Service packages. While not true ESB's per se, many integration vendors have started providing cloud based solutions for managing your cloud-to-cloud use cases. Because these solutions are specifically tailored for the Salesforce Platform (and other popular SaaS vendors), the time to build and deploy an integration can be radically reduced as opposed to using the On-Premise ESB.

Capability #12 – The cloud service bus can handle service mediation, transformation, routing, error handling, etc to your other cloud based end-points. Having to build durable and resilient integration solutions inside of Salesforce can be expensive and very complicated. Middleware should be used where and when possible.

Capability #13 – Some companies prefer to broker all integrations through their ESB, including Cloud-to-Cloud use cases. My warnings here are this: the cost of highly resilient ESB's can be EXTREMELY high. If the service levels between Salesforce and Workday, for example, must go through your on-premise technology, you maybe shooting yourself in the foot. Now your "Cloud" solution is piggy-backing on the same technical infrastructure, cost, service levels, and release timeline of your On-Premise solutions. Tread lightly and make sure to design your Integration Architecture first and foremost about delivering BUSINESS VALUE.

In Summary

dependency on batch integration technology or a total reluctance to use anything but Real-Time services design. However one of the reasons I enjoy what I do so much is that I have learned that there is no glass slipper in Salesforce Integration Architecture. One size does not fit all and no one solution can be the best for all or your requirements. It is up to you as the architect to analyze, recommend, and implement a variety of integration capabilities that will enable your team, clients, and company to realize the powerful transformation of moving to the Salesforce1 Platform.

Read the Entire Series

- Enterprise Architecture: Single-org versus Multi-org Strategy
- Designing Enterprise Data Architecture on the Salesforce1 Platform

Helpful Resources

Integration Patterns and Practices

Apex Best Practices: The 15 Apex Commandments - Salesforce Developers Blog

JAMES LOGHRY

The 15 Apex Commandments: Salesforce best practices and guidelines for developing applications with Apex code and Visualforce

January 26, 2015

More Blog Posts

Salesforce Event Relay is Generally Available

SASHA GOLUBOVA

We're excited to announce that Event Relay for AWS is now Generally Available! Event Relay is a big part of how we're building a unified developer experience that spans the Salesforce and AWS platforms.

Get timely developer updates delivered to your inbox.

Sign up now

salesforce

DEVELOPER CENTERS

- Heroku
- MuleSoft
- Tableau
- Commerce Cloud
- Lightning Design System
- Einstein
- Quip

POPULAR RESOURCES

- Documentation
- Component Library
- APIs
- Trailhead
- Code Samples and SDKs
- Podcasts
- AppExchange

COMMUNITY

- Trailblazer Community
- Events and Calendar
- Partner Community
- Blog
- Salesforce Admins
- Salesforce Architects