A New Approach to Enterprise Architecture: Reactive Systems

How a reactive data layer can bring performance, scale and resilience to your application architecture

Contents

Introduction

Transformation has always been difficult for any industry or organization. But today, disruptive software startups and emerging technologies are demanding that all businesses, across all industries, define and navigate a transformational path that embraces the app economy and represents a new focus on digital products and software-defined architecture. How will your business react?

For developers, architects, and product managers, the question of "how" can fall to assumptions – relying on legacy technology and trusting in traditional integration patterns. The "what" and "why" also needs careful thought to overcome the realities of today's dependence on the internet.

This paper will take a critical look at de facto standards for data integration and distribution, offer recommendations where reactive systems and realtime messaging can help, and propose a new reactive data layer that adapts and evolves with today's business requirements.

The Internet – Opportunity and Cost

The internet has evolved faster than anyone could have imagined - back in 2000, the total number of network users was only 400M. Today the internet has over 3.3B users , and China alone has 721 million users.

With this growth, a single app today handles more traffic than the entire internet infrastructure did just a decade ago.

Application architecture and integration requirements have undergone a sea change over the past few years. The number of end-users, connected devices, and applications are increasing exponentially. All the while, enterprise data resources, systems of record and computing infrastructure are pushed out to an array of new cloud-based services that are connected – yet separated – by an unknown network. This unknown and unmanaged network resource is of course the internet, and it sits at the center of today's app architecture.

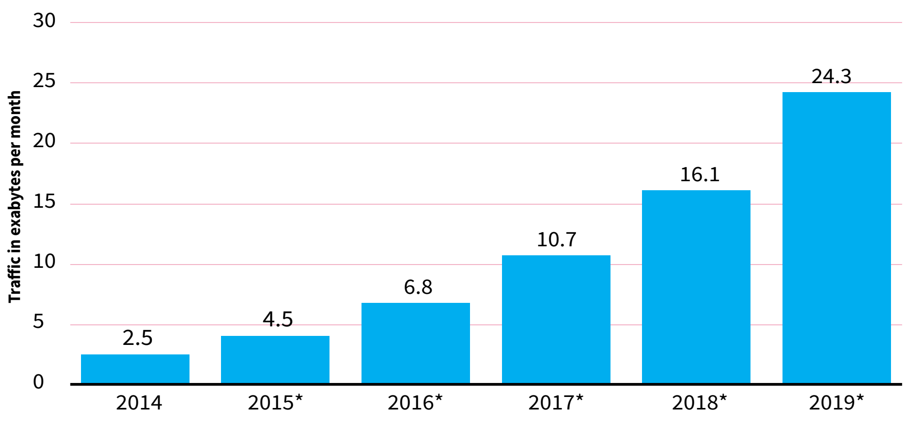
The internet has evolved faster than anyone could have imagined - back in 2000, the total number of network users was only 400M. Today the internet has over 3.3B users[1], and China alone has 721 million users.

With this growth, a single app today handles more traffic than the entire internet infrastructure did just a decade ago.

Quantifying this growth translates to some truly staggering statistics:

- Annual global IP traffic will pass the Zettabyte (1000 Exabytes) threshold by the end of 2016, and will reach 2 Zettabytes per year by 2019.[2]
- Globally, mobile data traffic will increase 10-fold by 2019 to 24.3 Exabytes per month, up from 2.5 Exabytes per month in 2014.

Global mobile data traffic from 2014 to 2019 (in exabytes per month)



Source:
Cisco Systems
© Statista 2015

Additional Information:
Worldwide 2014

[1]http://www.internetlivestats.com/internet-users/
[2]http://www.internetlivestats.com/internet-users/

Considering these numbers, it's easy to see that we're facing issues of scale and an ever increasing reliance upon the internet in our day-to-day lives. With this in mind, it's also easy to imagine that some integration paradigms, system architectures and app technologies of the past may not be the best choice for today, and certainly won't scale for the future.

The Impact on Enterprise IT

Traditional enterprise IT is reliant on monolithic solutions from a small number of vendors. Until recently, there was little need for these solutions to interact with other systems, never mind with the outside world. Even over the past three to five years, most organizations have been able to design and architect for well-known IT problems and tightly controlled application integration scenarios.

But the world of "traditional IT" has come to an end. Today businesses, partners, customers, and employees all demand more flexibility and capability.

But the world of "traditional IT" has come to an end. Today businesses, partners, customers, and employees all demand more flexibility and capability.

Rather than a tactical approach to integration, a broad enabling fabric will deliver the breadth of capability users demand with the performance, scale and resilience they expect. So when it comes to IT infrastructure, it is no longer viable to simply invest in products and services that cover only the requirements of today.

Architects, in any environment, must think about creating a sufficiently generic layer of infrastructure that can be applied to many different projects and serve the unknown requirements of tomorrow. With this approach, you naturally move away from a model of buying technology that solves just one problem, to instead investing in broad capacity that can be leveraged across multiple problem areas. Reusing capacity improves efficiency and effectiveness, while reducing costs and increasing the potential return from projects.

Defining Reactive Systems

The challenges of internet scale and the limitations of traditional IT, clearly demonstrate a need for a new approach to enterprise architecture. A new architecture has evolved to let developers conceptualize and build applications that satisfy these demands. We call these reactive systems.

Reactive systems are inherently more flexible, loosely-coupled, and scalable – making them faster to develop and easier to change. This allows developers to build systems that deliver highly responsive user experiences with a realtime feel, backed by an elastic and resilient application stack. Defined by the Reactive Manifesto[3], there are four key features architects and developers should follow.

Responsive:
The system responds in a timely manner. Responsiveness is the cornerstone of usability and utility. For end-users, responsiveness also means that problems may be detected quickly and dealt with effectively. Responsive systems focus on providing rapid and consistent response times, establishing reliable upper performance thresholds so they deliver a consistent quality of service. This consistent behaviour is reflected in error handling, builds end user confidence, and (importantly) encourages further interaction.

Resilient:
The system stays responsive in the face of failure. This doesn't apply only to highly-available, mission-critical systems — any system must be responsive during and after a failure to be considered resilient. Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other. Ensuring that individual components can fail and recover without compromising the system as a whole is key. Clients should never be expected to carry the burden of unexpected or undefined failures.

[3]*http://www.reactivemanifesto.org*

Elastic:
The system stays responsive under varying workload. This implies a design that has no contention points or central bottlenecks, resulting in the ability to shard or replicate components and distribute inputs among them. Reactive systems support predictive, as well as reactive, scaling mechanisms by monitoring performance in realtime. They achieve elasticity in a cost-effective way on commodity hardware and software platforms.

Message Driven:
Reactive systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling and isolation. Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message queues in the system and applying back-pressure when necessary. With non-blocking communication, the system will allow recipients to only consume resources while active, leading to less system overhead.

Understanding Messaging

Message-Driven

As described in the manifesto, a message-driven architecture is one of four key components within a reactive system. Typically, this integration pattern can be event-driven or actor-based.

Event-Driven:
This architecture is based on events which are monitored by zero or more interested parties (clients). Different from imperative programming as the client doesn't need to wait for a response from each request. Events are not directed to a specific destination, but rather available to interested end-points.

Actor-Based:

An actor-based system is a conceptual model to handle concurrency. It's important to understand that, although multiple actors can run at the same time, an actor will process a given message sequentially. The idea is similar to object-oriented languages, but the main difference is that actors are completely isolated from each other and they will never share memory.

In this paper, we focus on event-driven concurrency and the realtime messaging layer that provides the reactive infrastructure a given service will use to publish and subscribe to events.

Realtime

The world of "realtime" data is perhaps misunderstood and there can be strong opinions on the definition – here, we are discussing realtime web technologies that enable applications to react to events as they happen.

The world of "realtime" data is perhaps misunderstood and there can be strong opinions on the definition – here, we are discussing realtime web technologies that enable applications to react to events as they happen.

Across much of today's application integration there is no ability for systems to react in this way. Typically, system A (the client) will ask system B (the server) if new data is available based on a set of query parameters. So to really understand the value of realtime technology, we need to discuss the limitations of this integration pattern:

Polling is used by the vast majority of applications today. In this model, the client application repeatedly polls a server for data. Based entirely on the foundations of the web, these applications leverage the HTTP protocol, that allows the fetching of data in a request/response pattern. The application *asks* (requests) the server for data based on some query parameters, and waits for a response which is often of unknown size. If no data is available, an empty response is returned. The problem here is that empty responses, or multiple responses containing duplicate data cause a huge amount of overhead.

Long polling is a variation of polling designed to accommodate scenarios where the server does not have new data available. When an HTTP request is received the server will hang on to the connection until new data does become available, the server responds, closes the connection, and the process is repeated. This creates the effect of event-based responses, but in reality this is a costly hack.

Applications are no longer delivering "point-in-time data", and instead, focus on realtime, event-driven, responsive, engaging data.

Rather than polling back-end systems for changes, realtime messaging should be used to create responsive, event-driven applications. This allows you to scale your applications, without adding unnecessary load to the back-end. Integrating realtime messaging seamlessly with modern development frameworks for the web (Angular, Meteor, React, etc.) means users see data changes as they happen. This means applications are no longer delivering "point-in-time data", and instead, focus on realtime, event-driven, responsive, engaging data.

Data Efficient Messaging

But to achieve efficiency, we need to look beyond the integration pattern, to the data itself – and think critically about "what" data is being moved around, rather than simply "how".

We've concluded that event-driven architecture and realtime messaging are the foundation of how reactive systems must be architected. But to achieve efficiency, we need to look beyond the integration pattern, to the data itself – and think critically about "what" data is being moved around, rather than simply "how".

Application data is often complex, data delivery is complicated, and network costs can quickly skyrocket. From IT leader to product owner, enterprise architect to developer, finding the most efficient mechanism for distributing and consuming data should rank high amongst integration requirements.

In most messaging products, each and every update (message) sent to or from an application is the data payload. Meaning applications and services are paying to transport data that doesn't need to be moved. This includes:

- Redundant data
- Out-of-date data
- Duplicate data

Data efficiency is a core component of Push Technology's realtime messaging, and various unique and patent-pending features are offered to achieve this.

1. Network communication is performed by an event-driven kernel that uses non-blocking I/O to interact efficiently. The lock-free design exploits the way that modern CPUs access memory to avoid contention between sessions. This allows a single server to scale linearly across CPU resources, and achieve very high message rates.
2. Messages are serialized in a compact binary form. A small binary header is used to frame each message.
3. Topics provide stateful streams of data to each session. Once a session is subscribed to a topic, a subsequent update is sent as a "delta", (i.e. the difference from the previous value), if doing so will reduce the amount of data sent. This happens transparently to the application, so doesn't affect the ease of use: the delta is automatically applied by the client SDK, and passed to the application as a new value.
4. Messages that can't be immediately delivered to a session are queued. If the network connections fail, bandwidth is limited, or the client is simply slow, messages can back up on the queue. The platform can conflate the queue to remove messages that are stale or no longer relevant, or combine multiple related messages into a single consolidated message.
5. In addition to automatic throttling and conflation, the rate of messages delivered to a session can be manually constrained to prioritize one session over another or to place limits on bandwidth utilization.
6. Sessions subscribe to topics using wild card selectors. Unlike traditional messaging products, when new topics are added, sessions with matching selectors are automatically subscribed.

This data efficient approach to messaging means that most development teams see savings in bandwidth of up to 90%.

The 5 V's of data complexity

- Volume – the sheer scale of data we can or need to access. Distributed across many services and systems.
- Velocity – the speed at which data is being generated, the blurred lines between consumer and producer, and the need to move data around in realtime.
- Variety – the ever growing differences between data resources (structured & unstructured), data models and data location.
- Veracity – the unknowns, inconsistencies and fragmentation of data, that are making it increasingly difficult to harness.
- Value – the inherent business opportunity that exists within and between data sets, and being able to unlock this value at the right time.

Efficient Realtime Data Has Many Applications

Realtime Data Delivery for Mobile Apps

Applications are no longer delivering "point-in-time data", and instead, focus on realtime, event-driven, responsive, engaging data.

Many of today's mobile applications are point-in-time representations of data, refreshing information only when a user explicitly asks for an update. But interactive applications are infinitely more engaging, updating in realtime as new data becomes available. App developers need to enhance their traditional request/response data integration, and leverage realtime messaging. This delivers not only better application experiences that react to business events as they happen, but also provides a bi-directional communication channel for application data.

Extend Existing Middleware over the Internet

Enterprise messaging and middleware has long provided effective connectivity between legacy systems that would otherwise be unable to talk to each other. However, with application data, services and computing resources now widely distributed across a multitude of cloud platforms, a new approach to application integration is urgently required. Legacy middleware was designed to handle data traffic across dedicated and managed corporate networks. Organizations now need a data distribution layer that can adapt to the realities of the internet, connect the explosion of cloud services, and react to the ever mobile nature of all users.

Decouple Backend Systems from Frontend Apps

Short term business objectives and time-to-market demands often mean tactical integration decisions are made for many cloud and mobile applications, rather than considering a scalable long-term view across application architecture. Apps should leverage a reactive integration and abstraction layer, that hides the complexity of your data model, and decouple applications from potential backend changes. Then app developers can spend more time building features, and less time fixing integration problems.

Data Backplane for Microservices Architecture

Microservices have emerged as the preferred approach to build scalable applications that can adapt as requirements change over time. In contrast to monolithic applications which typically have a single relational database, a microservices architecture requires a mechanism to manage data communication between a potentially large number of services that are often running different technology stacks and located across multiple cloud services. By using an event-driven data backplane that offers the SDKs your developers need, a given microservice can easily publish events that other services (existing or future) may subscribe to.

Optimized Integration for IoT

An effective IoT architecture requires an event-driven integration platform that offers predictable performance and latency as connections increase, can intelligently distribute only necessary data, and also ensures connections are secure.

The massive scale associated with the Internet of Things presents some unique challenges in terms of application architecture, integration and data movement. With around 5 billion connected devices today, and the market expecting to grow to 50 billion connected devices by 2020, organizations must act now to ensure they can support the speed and scale of this growth. An effective IoT architecture requires an event-driven integration platform that offers predictable performance and latency as connections increase, can intelligently distribute only necessary data, and also ensures connections are secure.

Realtime Event Processing & Analytics

Big data resources often exist in various forms, across multiple data centers, presenting challenges for event processing and analytics. When integrated with data-efficient messaging, organizations can provide realtime distribution of this data that manages the complexities of connectivity across the internet.

Introducing a reactive data layer

Modern applications do not adhere to a typical client-server model, where one program (the client) requests a service from another program (the server). More typically, modern applications consume and produce data across multiple endpoints at the same time – all over the internet. Hence the need for efficient realtime messaging and a reactive programming model.

This translates to three fundamental lessons for data integration:

1. **Data** is key. What apps really care about is the real business data, what has changed and when it changed.
2. Business value exists in the **now**. End-users expect experiences that are as close to realtime as possible, and rely on having the best available data.
3. The **internet** lets you down. Since everyone is using the same network to exchange data, the developer that overcomes the internet's many obstacles wins.

At Push we have defined the concept of a reactive data layer (RDL) that gathers data from all systems and cloud applications and caches it in a single, live (in-memory) data model that transmits data between the reactive data layer, producers, and consumers. The work of filtering, synchronizing, and optimizing the data is performed by the reactive data layer.

By removing complex and strong dependencies between components, the reactive data layer can flex and scale with minimal impact on the application architecture it supports.

This approach provides fast access to large volumes of data as well as predictable, reliable scalability for applications. By nature, it promotes the development of applications that are event-driven, vs. request-response integrations that use conventional data layers and force unwanted coupling between components and subsystems. By removing complex and strong dependencies between components, the reactive data layer can flex and scale with minimal impact on the application architecture it supports.

Here is a checklist of features from Push Technology that are needed for an effective reactive data layer.

✓ Publish / Subscribe Integration

A reactive data layer provides a publish and subscribe integration model. The server stores data against a hierarchy of topics, where each topic has a current value that can be queried in an ad hoc fashion. More commonly, a client session subscribes to the topics that are of interest to them, and the server will push each change to the topic as a stream of events.

✓ Dynamic Data Model

Client sessions subscribe to data topics using wild card selectors - when new topics are added, sessions with matching selectors are automatically subscribed. This avoids the need for separate topic life-cycle processes, and data objects can be frequently created and destroyed without impacting existing applications.

✓ Value-Oriented Programming

A value-oriented programming model is a fundamental feature of a reactive data model. Applications are built against an API that provides streams of values, rather than individual messages that need further decoding. Client SDKs provide a common programming model, making best use of the features particular to the implementation language. This frees developers to focus on application concerns, rather than data integration.

✓ Inverted Data Grid

A reactive data layer can be thought of as an inverted data grid. Like a data grid, the reactive data layer stores values in-memory – yet traditionally these are optimized for query, and primarily support a polling paradigm. In contrast, the reactive data layer is optimized for a realtime, event-driven communication. Often deployed as a specialized cache, data grids offload processing from a back-end system. The reactive data layer offers the same benefit and it is particularly designed for delivering streams of realtime data.

Summary

Forward-thinking enterprise architects and developers should look to add a reactive data layer to their core technology stack. Just as a business intelligence layer provides a single reference point for customer or reporting information, a reactive data layer provides a single view of data from all systems and cloud applications, creating a living, breathing data model.

With Push Technology products, your reactive data layer can deliver and exceed the standards of the reactive programming model in terms of scalability, resilience, and responsiveness by overcoming the challenges presented by disparate back-end systems, the unknowns of the internet, and client device complexity.

About Push Technology

Push Technology provides realtime messaging, optimized for streaming data over the Internet.

Better performance, scale and stability are crucial problems for all developers. The solution? A reactive data layer that can grow adapt and scale as required. With our products Reappt and Diffusion – and their unique approach to streaming data – developers are armed with realtime messaging that provides the foundation for your reactive applications.

Try Reappt on IBM Bluemix for free or download Diffusion today.

We make the Internet **work** for the mobile-obsessed, everything-connected world

LONDON
+44(0) 2035 880900

SAN JOSE
+1 408 780 0720

www.pushtechnology.com