WHITE PAPER

Microservices in Production:

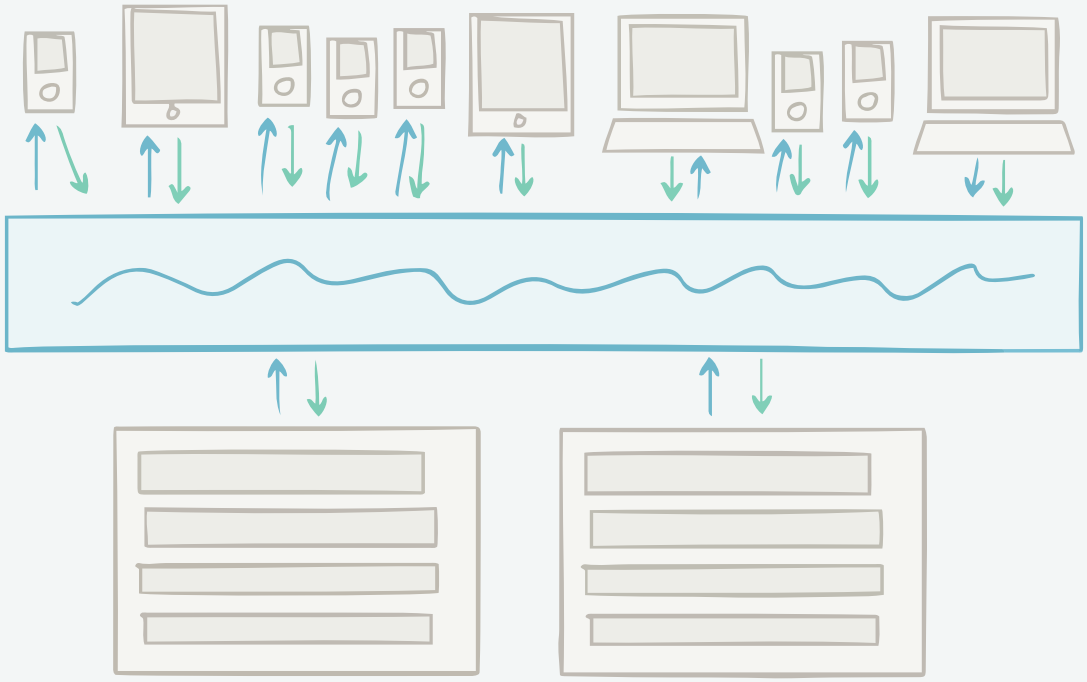
What Ops Should Know About Reactive Systems

Table of Contents

"The more I thought about Reactive the clearer it became that businesses, not just infrastructures, need to act in this way. Businesses need to be Reactive because you can't predict the future and they will need new technical architectures to support the change, which looks a lot more like web computing. Agile, bursty, lean—that's the future of business."

James Governor
Co-founder, RedMonk

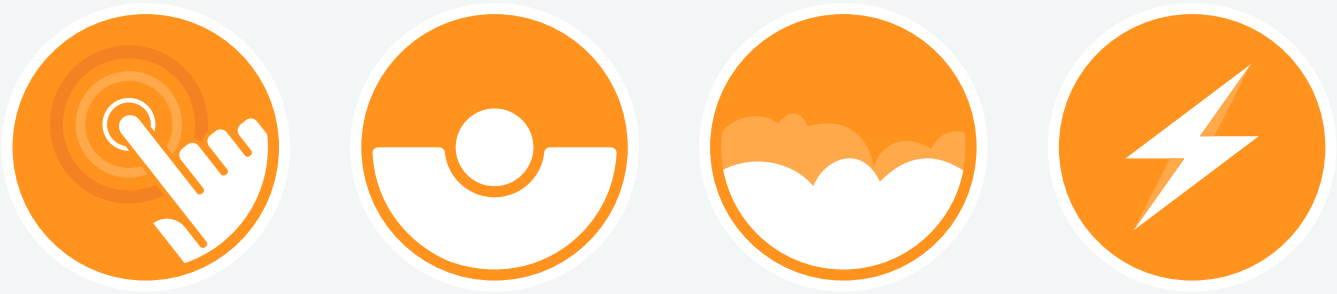
Microservices in production: no longer just for early adopters



Whether you call them *microservices* (the industry favorite), *picoservices*, *uniservices*, *decoupled services*, or even "*SOA done right*", one thing is clear: enterprise deployment of microservices-based applications is no longer a fringe concept enjoyed by a handful of early adopters.

While microservices–which may not be so very small, in fact–are being embraced across technology platforms, they have found a real home in Reactive systems. Indeed, a recent survey on Reactive systems of over 3000 respondents revealed that microservices-based architectures account for just over 50% of applications being built and deployed by enterprises today.

Reactive systems, as described by the Reactive Manifesto as **message-driven**, **elastic**, **resilient** and highly **responsive**, are fueling the new wave of applications that are deployed on everything from mobile devices to cloud-based clusters running thousands of multi-core processors. Users expect millisecond response times and constant uptime.



The largest enterprises like Facebook, Google, Netflix and Twitter are measuring their data in Petabytes—but even smaller companies are hungry to keep up with these big players, and strive to mimic the same quality of user experience.

However, this places enterprise Operations teams in a difficult position, since deploying, managing and monitoring distributed, microservices-based systems has not been the main focus until recently.

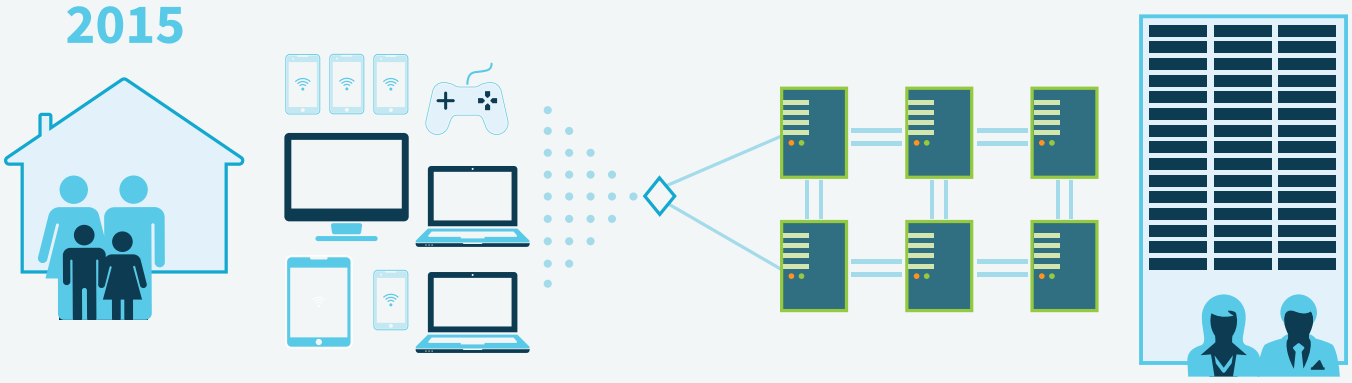
In this white paper, we'll review just how today's demands on Operations are simply not met by yesterday's software architectures and technologies; the pressure facing enterprises to manage resilient, responsive applications is brutal, yet most existing technologies available today are not designed to deploy and manage Reactive, microservice-based systems running on clusters. **It's due to this fact that Operations face a higher risk of downtime by using inappropriate tools/practices at a time when being unavailable is more costly than ever.**

What got us here: the growth of mobile and IoT (2005-2020)

The rapidity of technological innovation is always challenging for enterprises. If we only look back to 2005, you'll notice that we didn't have the iPhone, Facebook, Twitter, Netflix streaming, Kindle, App Store, Spotify or Big Data tools. Enterprise architectures looked something like this:

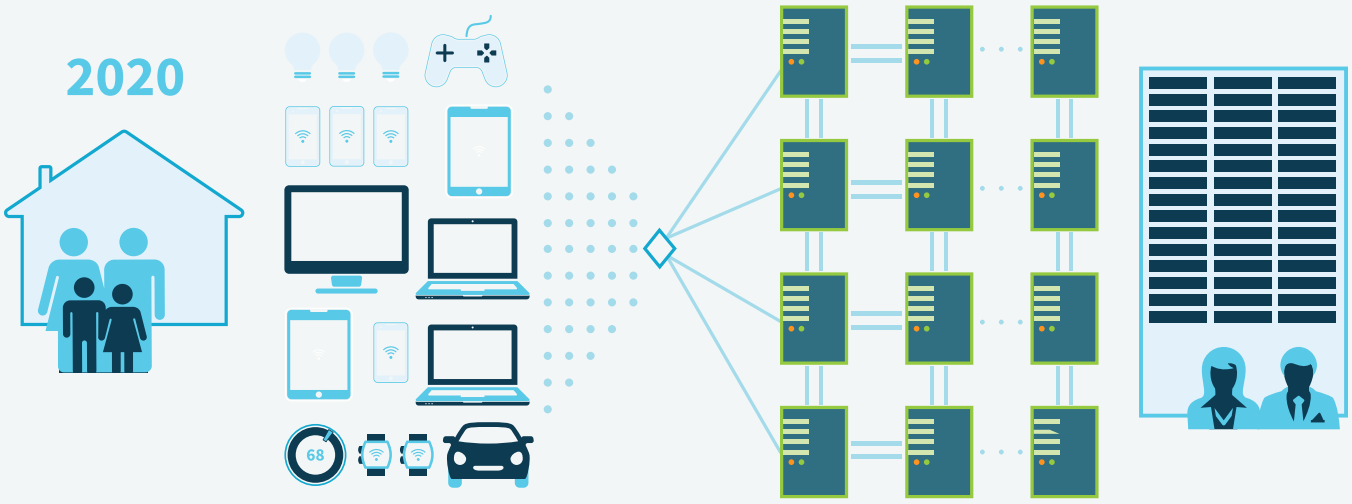
In 2005, the average household of consumers had one or maybe two computers—a desktop and laptop, let's say—which would connect to applications running on one or two redundant servers. Internet connectivity wasn't especially fast, and applications weren't especially fast. Let's face it, by today's standards nothing was really that fast. And most of us were fine with it—back then.

Jumping forward to 2015, we see a distinct difference in the network topology of enterprises:



Now the very same household now has multiple devices per person—all of which may use different protocols and platforms. Multiple PCs, mobile phones, tablets and e-readers are built with the intention to deliver instant response times and constant availability. In this scenario, we have 10 or more devices connecting to multiple application server instances.

Much growth over the last decade is a direct result of the explosion in mobile devices. This put a lot of pressure on internet services and enterprise systems during that time, but many now think that an even larger wave is is approaching fast in the Internet of Things (IoT). **Gartner predicts** that the IoT ecosystem will lead to the existence of over 25 billion connected devices by 2020 (and a **more recent estimate** by the FTC puts this at 50 billion). This is what enterprise architectures will need to look like in order to support just a single household then:

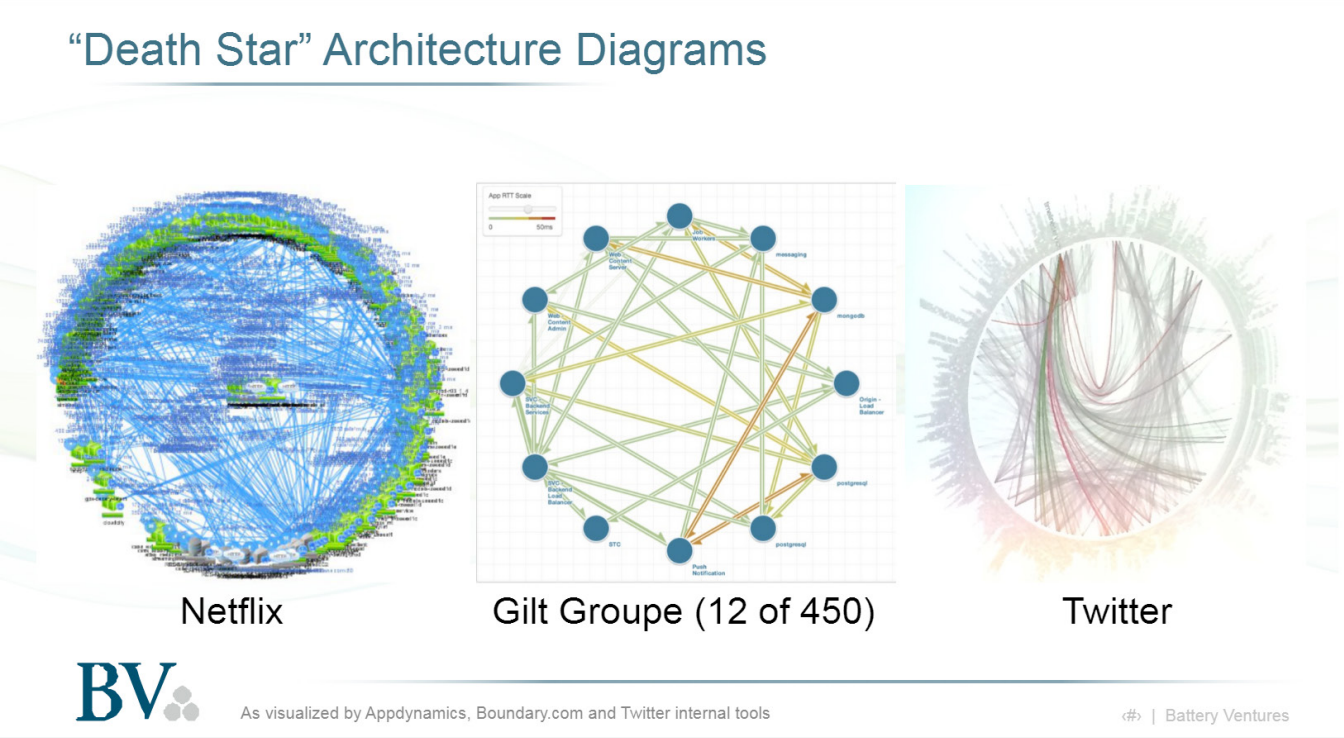


Naturally, the connected "things" we use today will look as advanced in 2020 as an iPhone 3G does in 2015. Enterprises will build new systems—and refactor many legacy ones—in order to accommodate this new generation of intelligent wearables, home appliances, health devices, automotive elements, lights, windows and locks that will join up with the huge base of mobile devices. Many tens of devices per household will connect constantly to tens or hundreds of servers, 24 hours a day, 365 days a year.

If this is the future, then we believe that a coherent approach to systems architecture is needed, and that all necessary aspects are already recognized individually: by creating systems that employ an asynchronous, non-blocking message-driven approach, teams can build systems that are by nature more resilient and elastically scalable, resulting in a systems that remains responsive regardless of what's happening to it.

What do Reactive microservices-based systems look like today?

As we said above, Reactive microservices-based systems are no longer just for the early adopters, yet it's the early adopters that have spent the most time with them. Consider these architecture visualizations of Netflix, Gilt and Twitter:



Source: **http://www.slideshare.net/InfoQ/migrating-to-cloud-native-with-microservices**

What we see in this image are massive distributed clusters of microservices, based on asynchronous message-passing and multi-core/multi-threaded processing on, ideally, inexpensive commodity hardware. With this approach, these systems remain resilient and elastically scalable, making them predictably responsive at all times.

Many of us still remember the Twitter "fail whale", back when they were running Ruby on Rails instead of Scala, but no more. By architecting distributed, decoupled, message-driven systems, the petabytes of data generated each day by Netflix, Twitter, Gilt, iHeartRadio, Zalando and others is safely carried through asynchronous, non-blocking, self-healing, massively-scalable Reactive systems.

But just developing Reactive applications is one thing. What about running these tremendously active and robust architectures in production? What about your Operations team?

What Reactive microservices-based systems becoming mainstream means for Operations

Enterprises have begun embracing the concept of architecting Reactive applications based on microservices for many reasons: they are more flexible, loosely-coupled and scalable. Using the **Lightbend Reactive Platform** technologies Akka, Spark, Scala, and Java, microservices-based applications are easier to develop and amenable to change, plus significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster.

Reactive applications are actually safer, more predictable and less difficult to orchestrate than traditional systems; however, they differ from traditional architectures and this unfamiliarity must be managed using new approaches. Armed with existing solutions, is your Ops team confident that they have the tools needed to handle the upcoming onslaught predicted for 2020?

We will be the first to say that the challenges of deploying and managing Reactive applications are not obvious to the casual observer. Without technologies designed specifically for distributed architectures and microservices-based systems, there are several reasons why we believe enterprises are struggling to deploy and manage Reactive applications:

Existing Operations processes and methodologies are inconvenient for Reactive microservices-based systems

Existing Operations tools and technologies have not been designed from the start for Reactive microservices-based systems

The costs and risks caused by improper tools are higher than ever

Existing Operations processes and methodologies are inconvenient for Reactive microservices-based systems

The demands placed on systems are ruthless nowadays. Starting with global expansion in mobile technology and continuing now with IoT, Operations deals with exponentially greater loads; yet for all their efforts, services provided by airlines, banks, retailers and telecommunications providers are still perceived to be sluggish.

This presents a challenge for Ops: armed with the technologies of yesterday's monolithic systems, how can they manage completely new microservice-based architectures?

Operations still battles with challenges when it comes to launching even a single app on a single server; without automation to ensure process repeatability, you are left with a higher likelihood of errors that lead to downtime.

When developing applications, experimentation with microservices in a dev environment is encouraged and relatively risk-free. Conversely, deploying and managing applications in production can be incredibly risky. Sometimes enterprises suffer so greatly that a drastic decision is made: in a **recent poll of companies** that dealt with unexpected downtime, 1 in 5 of them "let go" at least one employee as a result of the outage. There is little room for experimental improvements when any change to the *status quo* can have severe results.

This is why we see Reactive systems built to withstand failures gracefully and without resulting in downtime becoming more mainstream.

Existing Operations tools and technologies have not been designed from the start for Reactive microservices-based systems

In response to these challenges, the market has embraced a handful of useful Infrastructure and DevOps technologies, e.g. Amazon EC2, Docker Puppet, Chef, Ansible and many more. But enterprises soon discovered, especially with Reactive applications, that these tools and services are really best for supporting the infrastructure needs of a single application and server (or two). These tools are useful in many cases, but have not been designed from the beginning with the needs of distributed clusters of 5, 10 or 50 nodes in mind.

If you imagine the challenges of manually deploying a single app to a single server, then multiply that risk by N-servers to see what it's like when deploying a Reactive, microservices-based system to a cluster, where a combination of reactive and predictive scaling is often needed.

Additionally, if anything goes wrong then a rollback of the whole deployment is necessary; being able to do so at the push of a button necessitates automation, versioning, **immutable deployments** and support for rolling upgrades. Without this type of set up, the risks and unpredictability of the roll-out can be huge in scope.

The costs and risks caused by improper tools are higher than ever

What makes a great user experience today? The list is long, but probably would start with Responsiveness (aka availability). At this very moment, your customers are doing everything they can to stretch your applications to the limit, and they don't want to wait. In an age when extreme loads on servers make downtime more likely, it's also more expensive than ever before. Here is some data on the real-life costs of service downtime:

Gartner published that the industry-recognized average cost of downtime is equivalent to roughly $5,600 each minute (over $300,000 per hour)

Avaya's 2013 survey of over 200 IT professionals in large companies revealed that 80% of them lost revenue during downtime—the average was approximately $140,000 per hour, with financial sector companies losing an average of $540,000 per hour

For all the cost that goes with it, achieving 99.999% availability is prohibitively expensive for most enterprises. The general impression is that traditional, full stack architectures cannot be up 100% of the time: many of us architectures cannot be up 100%: many of us remember how **Knight Capital lost $440 million** due to incorrectly processed trades during a failed deployment, killing the entire company in just 30 minutes.

Reactive systems with microservices-based architectures are designed with the understanding that they will often fail—but gracefully, isolating and replicating faulty instances while maintaining system responsiveness, resilience and elasticity. In fact, the motto for the Akka project is "Let it fail".

Final thoughts and a proposed solution

Microservices-based applications excel in the environment enabled by Reactive systems, starting with an asynchronous, non-blocking, message-driven foundation to enable a high level of resilience and elasticity to create a system that is always responsive regardless of what's happening to it. Enterprises are pushing to deploy microservices-based applications to keep up with competitive pressures and higher demands for a responsive, best-in-class user experience.

Yet the gap continues to increase between the enterprise development teams building Reactive applications, and the requirements needs by Ops teams to deploy and manage them. Ops is unable to tackle these challenges with what they have, as we reviewed:

- **Existing Operations processes and methodologies are inconvenient for Reactive microservices-based systems**
- **Existing Operations tools and technologies have not been designed from the start for Reactive microservices-based systems**
- **The costs and risks caused by improper tools are higher than ever**

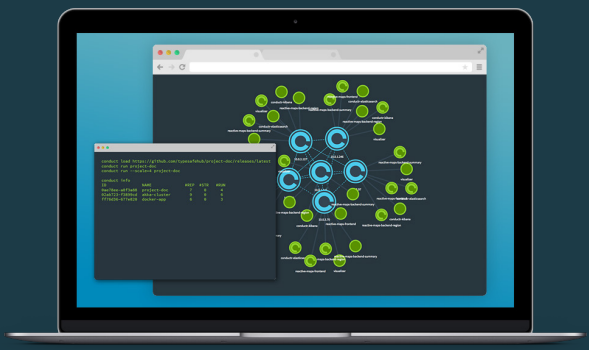
With these factors adding to the already challenging task of re-architecting traditional full stacks to become more Reactive, the last thing Operations needs to worry about is how on earth they'll be able to keep these systems online.

What's they need is a non-disruptive, low overhead technology that was designed for asynchronous, distributed, fault-tolerant systems from the ground up. This tool needs to be a "conductor" of clusters across availability zones, that lives inside the cluster and self-heals failed applications, nodes and even network partitions.

With our tool **ConductR**, we are dedicated to fueling Operations teams with the technology platform they need to intelligently deploy and conveniently manage their microservice-based applications, using the same Reactive principles that developers embrace.

LIGHTBEND CONDUCTR

Strengthen system resilience and improve elasticity



Handle failures gracefully, scale elastically and embrace change in your Reactive system:

- Automated cluster startup and dynamic service discovery
- Load balancing at high scale and self-healing features
- Infrastructure agnostic, developer sandboxing and consolidated logging

We bet your system isn't as resilient as you'd like. Contact us to schedule a private demo of ConductR.

REQUEST A DEMO

Lightbend

Lightbend (Twitter: **@Lightbend**) is dedicated to helping developers build **Reactive applications** on the JVM. Backed by Greylock Partners, Shasta Ventures, Bain Capital Ventures and Juniper Networks, Lightbend is headquartered in San Francisco with offices in Atlanta, Switzerland and Sweden. To start building Reactive applications today, **learn about Reactive Platform.**