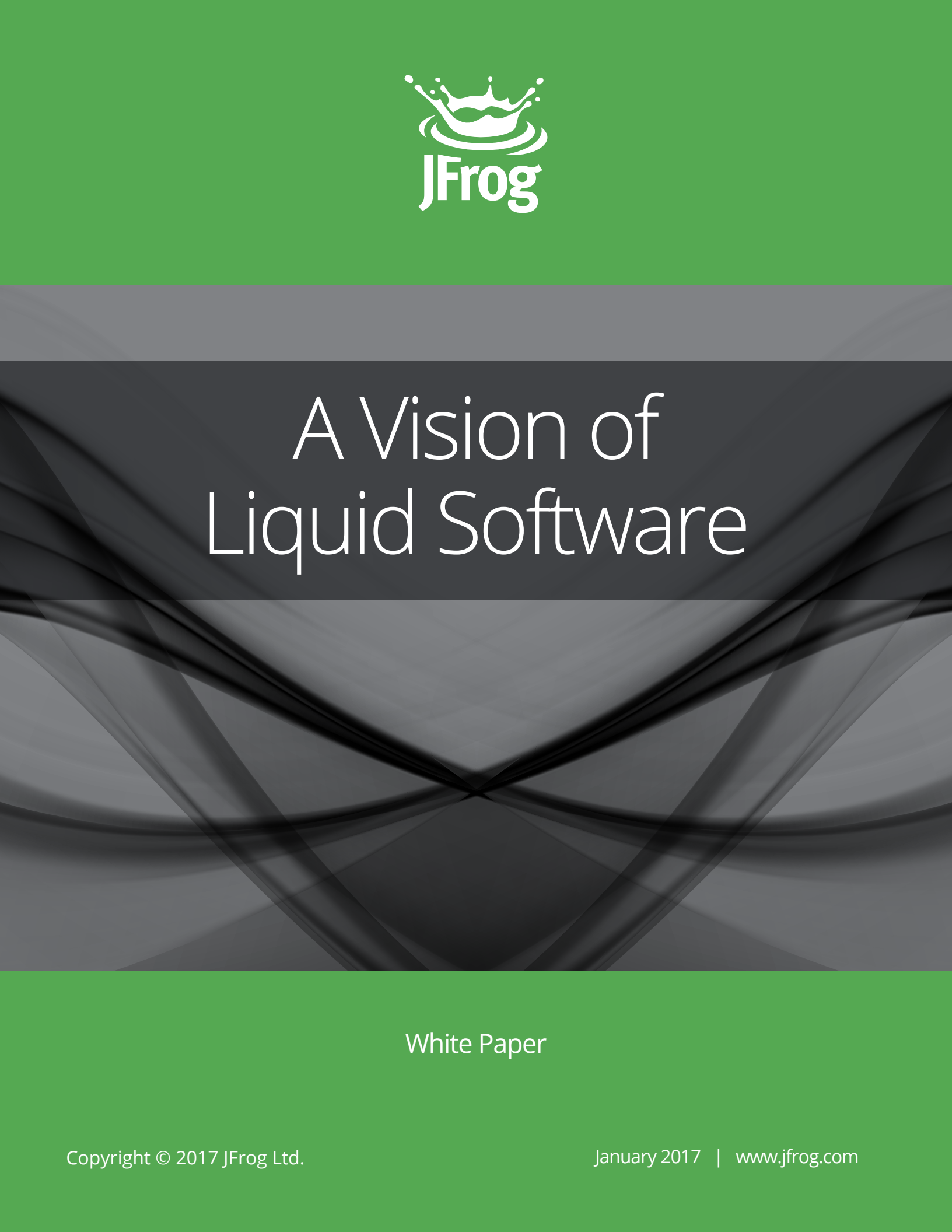


A Vision of Liquid Software

White Paper

January 2017 | www.jfrog.com

Table of Contents

Executive Summary.....3

Release Fast or Die.....4

Maintaining a Competitive Edge.....4

Fixing bugs and issues.....4

Removing Security Vulnerabilities.....4

The Vanishing Version.....5

Liquid Software and Continuous Update.....6

The Last Mile.....7

Trust and Security.....7

Software Quality.....7

Versionless Software.....8

Transparency and Coordination.....8

Decreasing Release Cycles.....8

Conclusion.....9

Executive Summary

In modern industry, every company is (or at least has) a software development organization that must develop websites, mobile apps and other software. The capability to release software fast is critical to the company's success for several reasons. First and foremost, a company must be able to constantly improve its offering in order to differentiate from the ever-present competition. Second, bugs and issues must be fixed as quickly as possible; losing customers due to a malfunctioning product is never an option. Third, any security vulnerabilities found once a product is in production must be remediated immediately, and there are many more reasons.

For each release of software, it is normal to assign a version number, however, this too is changing. The concept of a version is melting away, and this is due to three factors:

- **Short release cycles:** Software automation has vastly reduced release cycles. New releases are delivered so frequently that customers are no longer concerned with the version number. They just want to know that they are safely running on the "latest" version at all times.
- **Distributed software:** Software is becoming more distributed and is often delivered as a collection of micro-services on the cloud. Since each microservice is updated on an independent release cycle, the aggregated "macro" version is constantly changing making it meaningless.
- **Internet of Things (IoT):** Managing software updates for the billions of devices connected to the internet can only be efficiently managed through automatic updates without human intervention, and these devices are not concerned with a version number.

Looking forward, as release cycles get shorter and microservices get smaller, we can imagine a world in which at any one time, our systems' software is being updated. Effectively, software will become liquid in that products and services will be connected to "software pipes" that constantly stream updates into our systems and devices; liquid software continuously and automatically updating our systems with no human intervention. This is the next big challenge of the DevOps revolution. Just as we turn on a tap, expect water to come out without having to think about it, and trust the quality of the water, our systems and devices should be continuously and automatically updated with software we can trust and consume safely. However, there are still some challenges to address.

Trusting automatic updates cannot be taken for granted. Robust security mechanisms must be in place to confirm the safety of software that is streamed to our systems and devices. The quality of software updates must also improve so companies can be fully confident they will not break existing systems. Then there is the vanishing version to which a viable alternative must be found, so that if something does go wrong, a vendor can precisely identify the software that is responsible for the issue. There must also be a way for automatic updates to be coordinated with the systems they are updating to verify full compatibility. Finally, release cycles are still not short enough. Vendors are still struggling with available tools, the pace of DevOps adoption, code size and automation to continue reducing development time.

As release cycles continue to get shorter and tend to zero, software will become liquid, continuously flowing to automatically update the compute edge. To get there, automation must be extended until it is pervasive in all software development organizations. Standard and trusted security systems will have to authenticate and validate software flowing through the pipes, and exhaustive metadata will provide transparency to ensure full compatibility. As these systems and mechanisms continue to improve, the emergence of continuous and automatic update of liquid software is only a matter of time.

Release Fast or Die

Modern industry has evolved to the point where a company's survival depends, amongst other things, on how quickly it can release software and keep it updated on the compute edge. While this may be more intuitive in technology companies, it is less obvious for companies that, on the face of it, are in other verticals. So first, let's remove that distinction and blur the lines. Maintaining complex websites, mobile apps, assembly lines, sales funnels, distribution chains and many other functions requires software, and much of this software is developed within the company. Therefore, in practise, every modern company is (or at least has) a software development organization and must have the capability to release software fast. Here are just a few reasons why.

Maintaining a Competitive Edge

In any marketplace, a company must be able to constantly improve its offering in order to differentiate itself from the competition. As soon as one company adds a new feature that is in demand, others scramble to add it and catch up. To stay ahead of the competition, a company needs to constantly set the pace with new capabilities and leave the scrambling to others.

Fixing bugs and issues

Nothing is more frustrating to an end user than software that doesn't work. Users have short spans of attention and very little patience when "clicking the button" doesn't produce the expected results. They very quickly take their business elsewhere and browse to the ever-present competition. Whether the company is running an eCommerce site or a reservation system, the presence of a bug can very quickly lead to millions of dollars lost.

Removing Security Vulnerabilities

The cyber security market has been estimated to reach $170 Billion by 2020. This is a testament to the amount of effort companies have to expend to keep their software safe. Security vulnerabilities detected in a company's systems must be remediated as quickly as possible. The faster a company can release a software update to remove a detected vulnerability, the quicker it can protect its software.

So it's clear that releasing software fast is paramount to a company's success, and each release is given an identifier - the version number, however this too is changing.

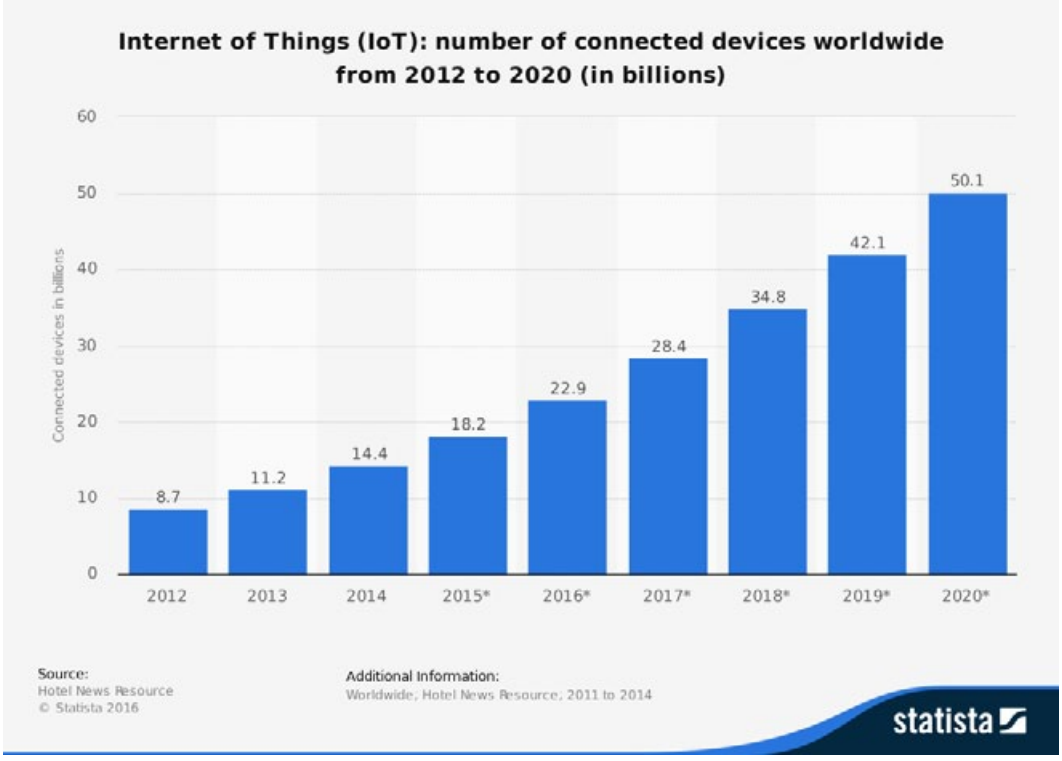
The Vanishing Version

In the old days of monolithic, bi-annual software deliveries, a release was a colossal event that took weeks to prepare, and due to the amount of changed code, was a risky undertaking. On the receiving end, customers would spend a lot of time unpacking the software, validating its different parts, and testing its integration with current systems until finally (and carefully) deploying it to production. Each monolithic version was ceremoniously tagged with a version number. This was a critical parameter of the software since it identified all the parts that worked coherently together, and was also needed for support staff to work through any issues reported by a customer. Today, the concept of a "version" is melting away due to three factors: short release cycles, distributed software and the Internet of Things (IoT).

Software automation has vastly reduced release cycles. Deliveries have accelerated so much that some forward-looking companies are releasing software several times a day. Customers no longer want (or even have the capacity) to unpack the software, test it and validate it for every single release. They just want to know that they are on the "latest" stable version at all times and that it will integrate smoothly with their currently running systems. The actual version number is irrelevant.

In addition, software is becoming more and more distributed. What might once have been run as a fully on-prem localized installation, is today, run as an array of microservices on the cloud where each microservice is updated on an independent release cycle. With each "micro update" the "macro version" changes, which effectively makes it meaningless since nobody is really aware of what version they're running on.

Finally, the number of devices connected to the internet is exploding with predictions reaching over 50 Billion devices by 2020. As more and more devices get on the grid, managing the update of their software can only be handled efficiently through automatic updates without human intervention, and devices don't really care what version they're running.



Liquid Software and Continuous Update

We started with monolithic, manual software releases with a distinct version number, and got to software built from distributed microservices released in rapid, automated cycles, but with no meaningful version number. If we now extrapolate the current state into the not-so-distant future, we can imagine a world where release cycles get so short, and microservices become so small that at any one time, it's likely that something is being updated. In effect, software will become "liquid" in that we will be connected to "software pipes" that stream updates into our systems and devices; liquid software continuously and automatically updating our systems with no human intervention.

While eyebrows may be raised at this idea, the concept is not foreign, and in fact, we already encounter it in our everyday lives. We turn on the tap, and take it for granted that water will come out. We trust the municipal authorities to make sure the water is clean enough to drink, or provide us with alerts and warnings that it may only be used for bathing or irrigation, or not at all if water sources have become severely contaminated. Similarly, when we plug a device into an electrical outlet, we expect the device to receive a steady supply of electricity at the right voltage so our device "just works". Similarly for cooking gas or any other utility we take for granted. None of us tests our utilities; we just trust their quality. So why can't we just connect our systems and devices to "software pipes" and trust that our software will be continuously updated, safely, robustly and automatically? This is the next big challenge that DevOps must address.

The Last Mile

While there is still a way to go, we are already closer to Liquid Software and Continuous Update than most people and companies think. In many cases software updates are already automatic. Take mobile devices as an example. Most of us set a checkbox in our cellphone's configuration to allow automatic updates of its firmware. The update just happens, and is even smart enough to wait until we're on a Wi Fi network so as not to burden our data plan. Similarly, when we download an app, we allow the vendor to stream updates to us automatically through the corresponding app store. But even those of us who don't allow automatic updates will click "I accept" when prompted without really thinking about it. We won't bother reading any EULA we are confronted with, and even if we did read it, it's unlikely that we would understand its legal jargon. At the end of the day, we just want the latest software without having to think about it.

This is the mode of operation we are moving towards for enterprise software, but for this to be widespread, there are still some challenges to overcome.

Trust and Security

When we turn on the tap, or plug into an outlet, we place trust our utility provider. That trust is based on past experience and the knowledge that the regulatory authorities are monitoring the relatively few utility providers we use every day, and ensuring that nobody is tampering with them. An enterprise software organization is likely to be "nurtured" by many different providers. Security mechanisms must be in place to validate the identity of each provider and confirm that the flow of software continuously updating our systems has not been tampered with.

Software Quality

Before an enterprise deploys a periodic update to its production environment, it will run a battery of tests to make sure nothing breaks. Nevertheless, production bugs and system outages are still costing enterprises billions of dollars each year. For enterprises to trust that continuous update is robust and will not break their production systems, liquid software quality will have to be be much better than the updates we get today. This places a heavy burden on vendors to provide software updates with a very high level of confidence.

Versionless Software

As already discussed, software is increasingly becoming versionless. End users (certainly those that are allowing automated continuous update) won't be able to identify the software they are running by a version number. That also means that if some disastrous bug is discovered in an update, there's no option to roll it back. And if the customer contacts the vendor's help desk to report a problem, or even to just get help with a feature, there must be a way for support personnel to identify exactly what the customer is running.

Transparency and Coordination

Since software will become versionless, there must be a way to coordinate what is currently running with the new software updating it. The automated mechanisms that install the latest updates flowing through the pipe will need a way to determine that the update is compatible and desired.

Decreasing Release Cycles

Software vendors are still struggling to shorten their release cycles. There are many barriers to overcome such as availability of tools, the pace at which DevOps practises are adopted,code size, implementing fully automated pipelines and more. Some companies are succeeding. In 2011, Amazon publicly announced that it was deploying code to its production systems every 11.6 seconds. By the end of 2014, Amazon was deploying every second (albeit, to its own systems). So while not every company is Amazon, their case shows that it's possible.

Conclusion

Software development organizations continue to release software at an ever increasing-pace with ever-decreasing cycles. Today's challenge for enterprise software is to embrace this unstoppable trend and get release cycles down to zero to achieve controlled, secure, continuous update on everything that uses software, from computing systems, to smartphones and tablets, to consumer electronics to simple connected devices in the IoT. Once that is achieved, **software will be liquid**, flowing continuously from development environments, through distribution platforms to perform automatic updates. To get there, the same automation that has reduced release cycles to mere seconds (in some cases), will have to be extended further until it is pervasive in all phases of software development in every development organization. The security risks inherent in automated systems will be mitigated by standard mechanisms that authenticate the production end of the software pipe to the consumption end, and validate that the software flowing through the pipe has not been tampered with. The whole update process will have to be mediated through exhaustive metadata that clearly describes the software and provides the transparency that software consumers need in order to understand what is being installed on their systems. At the end of the day, through the use of standard security systems, and metadata, the emergence of continuous and automatic update of liquid software is only a matter of time.