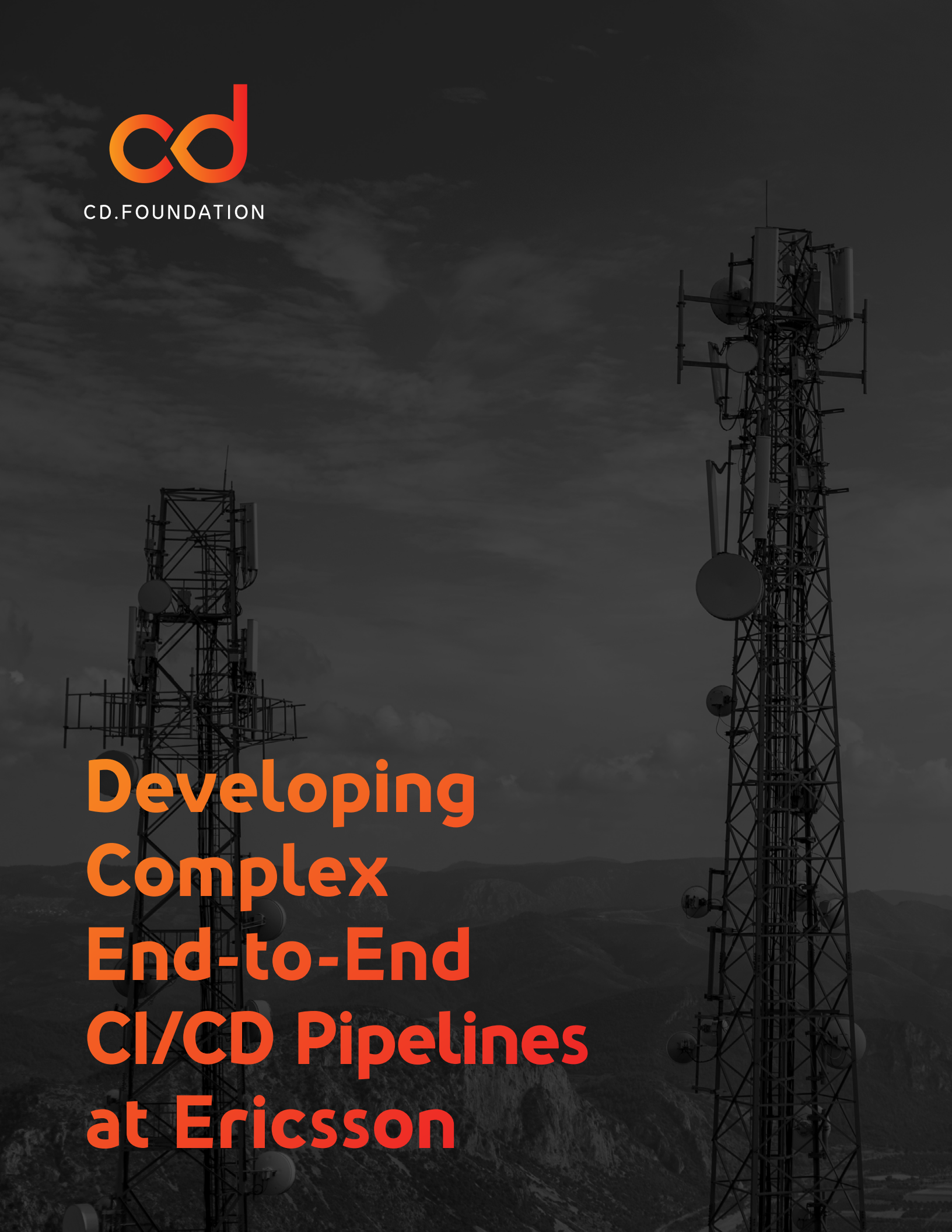

CD.FOUNDATION
Developing Complex End-to-End CI/CD Pipelines at Ericsson

# CHALLENGE

How does Ericsson gain scalability and observability from the beginning of the library component pipeline all the way to the end of the customer deployment pipeline? Ericsson has hundreds of products developed by thousands of teams that may have different ways of working and use different tools as part of their Continuous Integration and Continuous Delivery (CI/CD) pipelines. The CI/CD pipelines developed and used by the teams are interconnected, often going beyond internal organizational boundaries. The use of various tools that don't share a standard interface is a challenge that needs to be addressed. Ericsson has very high demands on its CI/CD pipelines and production systems, such as traceability and performance indicators from the commit of the lowest library component up to the customer deployment of Ericsson solutions.

# SOLUTION

A common event specification supported by the variety of tools and technologies gives the interoperability needed to achieve the scalability and observability required by Ericsson.

For this user story, we will use the following definitions of observability and scalability.

**OBSERVABILITY:** In distributed systems, observability is the ability to collect data about program execution, internal states of modules, and communication between components.

**SCALABILITY:** Scaling software development to handle products with thousands of developers and hundreds of product components.

# IMPACT

The event specification developed by Ericsson, Eiffel, allowed different parts of the organization to achieve scalability, traceability, and visibility within the CI/CD pipelines due to the integration of protocol into various tools such as Gerrit, Jenkins, and Argo Workflows. This further allowed the organization to bring new tools into the pipelines, such as artifact repositories and test frameworks, without impacting the existing pipelines.

**BY THE NUMBERS**

- 20K developers
- Over 3 million CI/CD pipeline-related events per day

ERICSSON

**CHALLENGES**
Interoperability, Traceability, Scalability

**INDUSTRY**
Telecommunications

**LOCATION**
Global

**PUBLISHED**
May 2023

**AUTHORS**
Kristofer Hallén, Principal Developer, CI Tools Architect
Emil Bäckmark, CI/CD Architect
Mattias Linnér, CI/CD System Architect

**PROJECTS**

cd events Jenkins Spinnaker

# TACKLING CHALLENGES IN INTEGRATING CROSS-ORGANIZATIONAL CI/CD PIPELINES

Long gone are the days when everyone coded a small piece of software that was compiled towards a specific hardware platform and delivered as such. Nowadays almost any application has dependencies on infrastructure software in multiple layers, with virtual machines, containers, orchestration engines, and so on.

At Ericsson, we've established a complex end-to-end software delivery flow with thousands of teams and tens of thousands of developers. In this case study, we'll tell you how we went about addressing scalability issues with CI/CD technologies while achieving traceability and reproducibility. We'll also share how they can be tackled using event-driven integration pipelines based on experience from large-scale software development.

Nowadays almost any application has dependencies on infrastructure software in multiple layers.

# ABOUT ERICSSON

Ericsson is a global company with headquarters in Sweden that has been around for over 140 years and now has over 100 thousand employees worldwide, and over 20 thousand of them are developers. As one of the leading providers of Information and Communication Technology (ICT) to service providers, Ericsson enables the full value of connectivity by creating game-changing technology and services that are easy to use, adopt, and scale, making our customers successful in a fully connected world.

Ericsson has a strong background in telecom hardware. Hardware is still an important part of today's customer offerings, but Ericsson has become, to a large extent, a software company. We develop software for all kinds of deployments, ranging from embedded bespoke hardware to Commercial off-the-shelf (COTS) hardware and in the cloud. We produce a lot of different deployment variants as our customers—ranging from operators providing high-coverage services in rural areas to those providing high-capacity services in metropolitan areas—have very diverse needs. We also have customers running Continuous Deployment to live networks. This case study will provide valuable input to developers of CI/CD services and products and institutions that face challenges while establishing and running complex end-to-end CI/CD pipelines.

# BACKGROUND

In 2010, Ericsson had waterfall development with teams specialized in different aspects of the software development lifecycle: development teams, integration teams, system testing teams, and so on. We didn't implement a lot of automated integration or system tests and had few automated deliveries between teams and products. In the best case, we had cron jobs that were scheduled, sometimes running tests daily, and we had test scripts located on shared folders on a server located somewhere within the organization.

There was an expectation from our customers that we should be able to release and deliver software faster and more frequently which was a catalyst for our investment in tools that could help us achieve those goals. We started to deploy more advanced tools and began using Hudson. We replaced the cron jobs that ran our regression tests with Hudson jobs. The development teams sometimes triggered the unit tests automatically on source code changes, using tooling developed internally on top of Rational ClearCase which was the main source control management (SCM) system in use in Ericsson at that time.

Automated integration started with development teams and integration teams connecting their pipelines together, but the integration and system tests still were mostly scheduled. We started moving the test scripts to the SCM system, enabling version-controlled test environments as well as the possibility to trigger pipelines automatically based on the updated test scripts.

Soon after, in 2011, Jenkins replaced Hudson. We migrated from Rational ClearCase to Git for most of our source code, and more and more of the test code was also added to Git. Around this time, Continuous Integration became a buzzword and our automated interaction increased when the system test teams connected to the integration teams' pipelines. We reduced manual handovers with these automations and decreased time from commit to quality-tested products from weeks to days. The manual interaction points decreased as more dependencies were integrated and with that, the quality increased because we had more precise delivery process predictions. This shift in tooling and process increased our ability to react to market changes and gave us the possibility to bring new features and corrections to customers faster.

The automation journey was not trivial, we had a lot of team-specific Jenkins instances, managed by the teams themselves. This was mitigated by introducing centrally managed Jenkins instances but each team had unique needs for specific plugins, which was costly to handle. There seemed to be no end to these increased integration points and it is something we are still encountering as we continue to integrate more and more teams, products, and tools.

As a global company, we have teams in multiple locations and time zones, each with their needs for builds, integrations, and different toolchains. This added complexity makes it very hard to know the origin of the commits and where they are in the delivery chain. A lot of manual work was also needed to figure out the configuration management part: what does the release contain and how well was it tested?

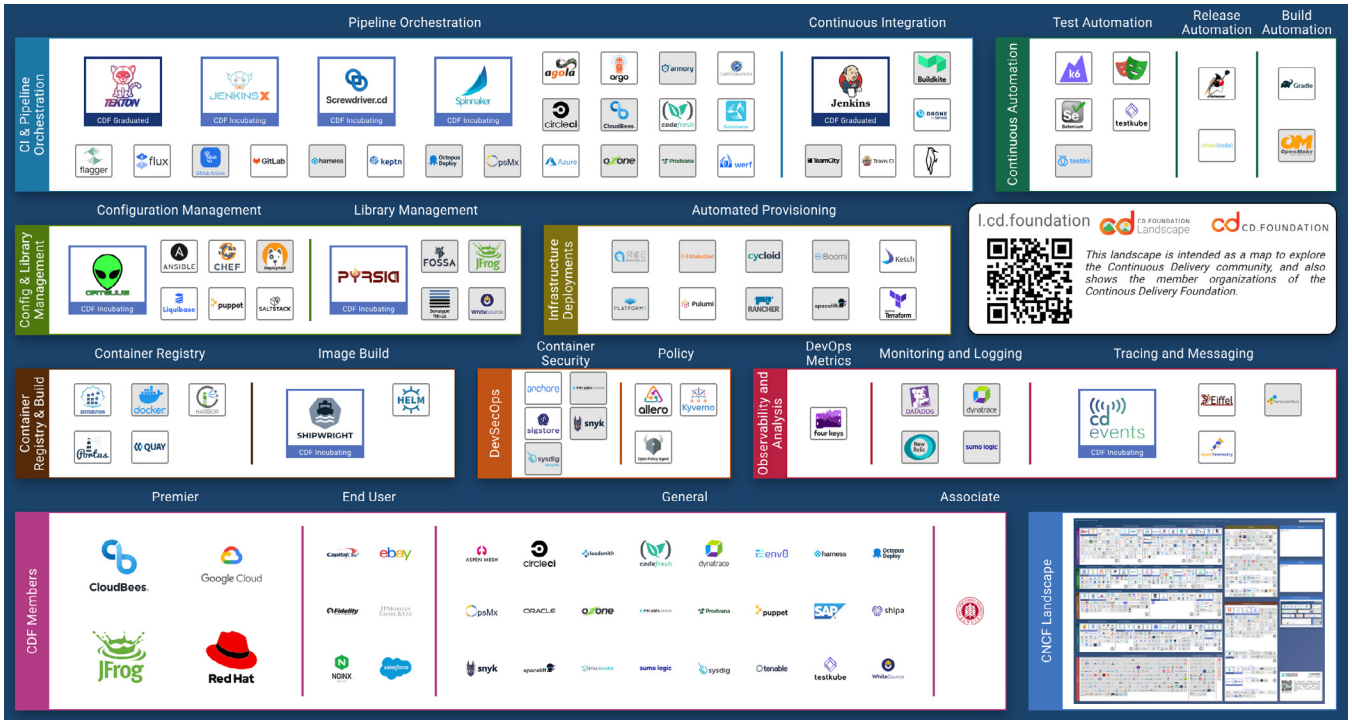
In concurrence with the increased automation, we received increased market demands to react quickly with top-quality service. At the time, our setup didn't scale. We needed large-scale Continuous Integration and Delivery and needed to integrate small changes often with fast feedback.

# WHY DON'T WE JUST ALIGN TOOLS?

As already mentioned, Ericsson needs to support a number of different products and technology stacks in our CI/CD system, from software development for our own silicon to microservices in the cloud. This level of complexity makes it practically impossible to align all parts of the toolchain. Additionally, the CI/CD Landscape is constantly evolving without a single solution to rule them all.

We want to be able to change our toolset based on our changing requirements whenever necessary.

The current trend is that the number of interaction points between tools will continuously increase. In the past, Ericsson had a few servers and tools that were responsible for handling the pipelines. Today the multitude of tools and services adds another level of complexity to the CI/CD cake.




As an example, let's say we have these generic tool types in our CI/CD system: SCM system - Pipeline engine - Build System - Artifact Repository - Test system - Deployment tool, and we have these tools in place to build up a certain such CI/CD system: Gerrit - Jenkins - GNU Make - Proprietary artifact repository - Proprietary test framework - Proprietary deployment tool. Now, there could be multiple reasons to replace one or several of those tools with something else. For example, there could be a wish to migrate to something like this: Bitbucket - Tekton - Shipwright - Artifactory - Testkube - Flux CD, and at the same time incorporate tools from new domains to increase the functionality of the pipeline, like Pyrsia and Harbor.

Migrating everything at once is not a robust way to keep CI/CD operational. How could that be done in an iterative manner and at the same time keep existing monitoring capabilities and the history of all performed builds? One way to deal with this is to make sure all tools use and communicate through a standardized protocol on a standardized channel with broadcast and publish-subscribe possibilities.

Most tools have their own custom interfaces and languages. We could integrate point-to-point with plugins between these components and translate the languages between them, but a standardized protocol would bring interoperability out of the

box. This approach is comparable to spoken languages. Instead of the Swedish authors writing this case study in Swedish, which would force many readers to find a translation tool that might give you the right understanding of the text, we agree on a common language, in this case, English. This can be used to create understanding even though we in our own context speak another language.

# SOLUTION: COMMUNICATION IS THE ANSWER

How can we achieve scalability, traceability, and observability without sacrificing the flexibility of using different tools and technologies as part of our production systems? What about using broadcasted events sent as messages by the CI/CD pipelines to communicate activities such as the successful build of a new artifact, the confidence reached in testing that artifact, and the availability of a new release?

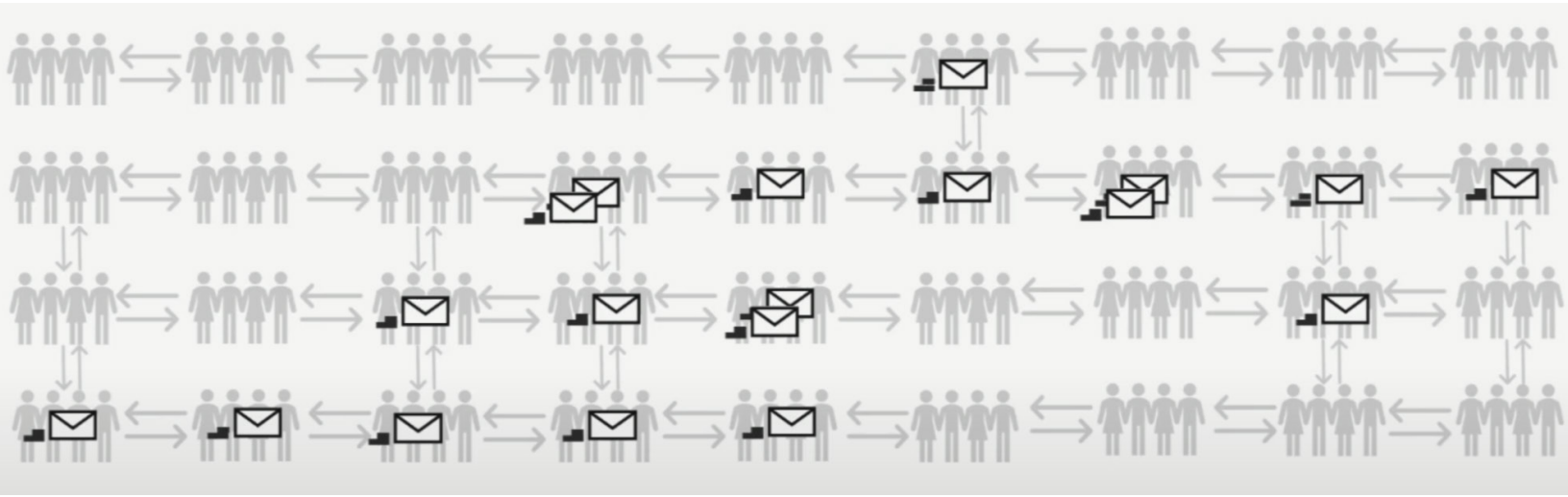
Apart from signaling new artifacts, test results, and releases, many other challenges can be dealt with by the use of events. Instead of subscribing to release emails, polling the websites of the product, or waiting for some tweets to announce a new release, the CI/CD pipelines could listen to standardized events. This means that when your CI/CD pipelines know, you will also know because you're probably monitoring your pipelines. You can also build/configure custom visualization solutions that take its input from the event data and data referenced from it.

**EVENTS IN ERICSSON'S CI/CD PIPELINE**

Ericsson started using events long before the CI/CD pipelines became too complex. We sent events internally to notify about:

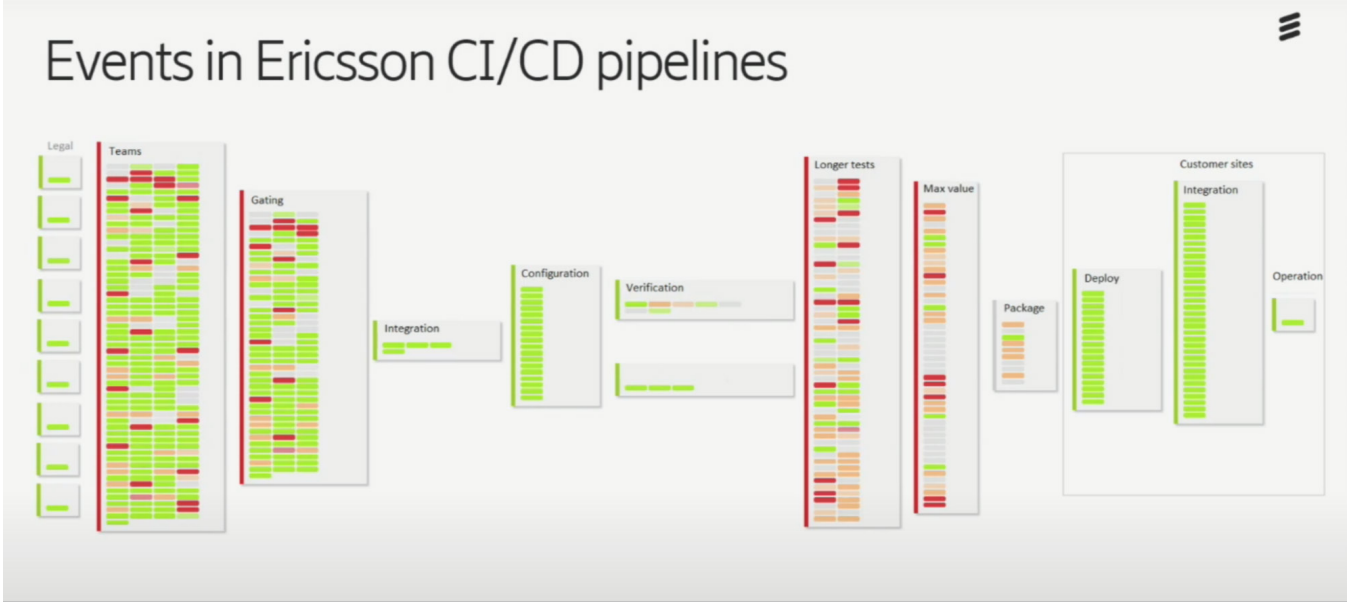
- new builds,
- artifacts being uploaded to artifact repositories,
- quality level that a certain artifact had reached,
- after each test activity,
- and new releases.

The people integrating the software down the chain could then decide on what pre-defined quality level they wanted to integrate that upstream product from. The events sent were standardized in a protocol that was agnostic to the underlying technology stack. We could replace the technology in the stack without affecting interconnected pipelines and other consumers of that event data.



Developers at Ericsson created a high-level language used in this protocol, using words that we used in conversation, which was a great bonus since it helped us establish a common spoken language across the organization. The protocol evolved to include more and more notifications, and in 2016 it was open sourced under the name Eiffel. It has helped Ericsson and other companies to achieve scalability, traceability, and visibility in complex end-to-end CI/CD pipelines.

Eiffel is based on the concept of decentralized real-time messaging providing traceability and KPIs throughout your pipelines, across platforms. One important aspect of the Eiffel protocol is that the events are linked together, forming a Directed Acyclic Graph (DAG) of notifications from the CI/CD system. This linking made it possible to visualize and measure the pipelines in a tools-agnostic way. The picture below shows a real-world example of such visualization with ongoing and finished activities through our network of pipelines.




This protocol that we are using has artifact events notifying that new artifact versions have been created, published, or verified. We have source code events notifying that something is pushed or merged in a source control management system, or if a new baseline is created. We also have activity events, notifying about the triggering, starting and finishing of our pipelines, pipeline steps, test case execution, etc.

What the implementation of events helped us achieve isn't limited to commercial product development. We believe that events can help also when tackling similar challenges in integrating software between different open source communities as well as while bringing those open source components into commercial development as dependencies. The open source projects used by the commercial products could be automatically updated based on the information in those standardized events published by the open source communities which could open up a lot more possibilities from the supply chain security perspective.

# LOOKING FORWARD

Cloud native technologies are now also part of the telecom industry and we want to benefit from the solutions and principles in the CI/CD area. Tools like Jenkins are being challenged by e.g. Argo Workflows and, in the CD area, by Flux. These tools provide new capabilities for our flows but require expensive integrations.

Our teams now see the benefit of using events for interoperability, we can change the tools we use but the events are still there, keeping everything connected. The challenges we faced while establishing and running complex end-to-end pipelines and our approach to solving them using events has made us realize that this is something we should collaborate on in broader open source communities, between organizations and like-minded individuals. To start the conversations around this topic and push the CI/CD domain forward, we participated in forming the Interoperability Special Interest Group (SIG) at CD Foundation at the beginning of 2020. The conversations within the SIG resulted in the creation of a new Events SIG followed up by the creation of the CDEvents project which is currently being developed by contributors from several different companies and open source organizations in a collaborative manner.

CDEvents aims to create a common specification for CI/CD events to make sure the CI/CD ecosystem has a common event protocol natively supported by different open source CI/CD tools such as Jenkins, Spinnaker, Tekton, and any other tool in the CDF Landscape.

Our vision is to allow taking any such tools and connecting them together in a pipeline with very little effort as the CDEvents protocol binds them together giving us traceability and visibility of what happens in the pipeline. This would help the users of these technologies to not have to develop custom solutions and glue code to get them to talk to each other and instead rely on interoperability among them made possible by CDEvents.

We are proud that our work at Ericsson inspired and helped shape the CDEvents project at the CD Foundation.

Our vision is to allow taking any such tools and connecting them together in a pipeline with very little effort as the CDEvents protocol binds them together giving us traceability and visibility of what happens in the pipeline.


CD.FOUNDATION