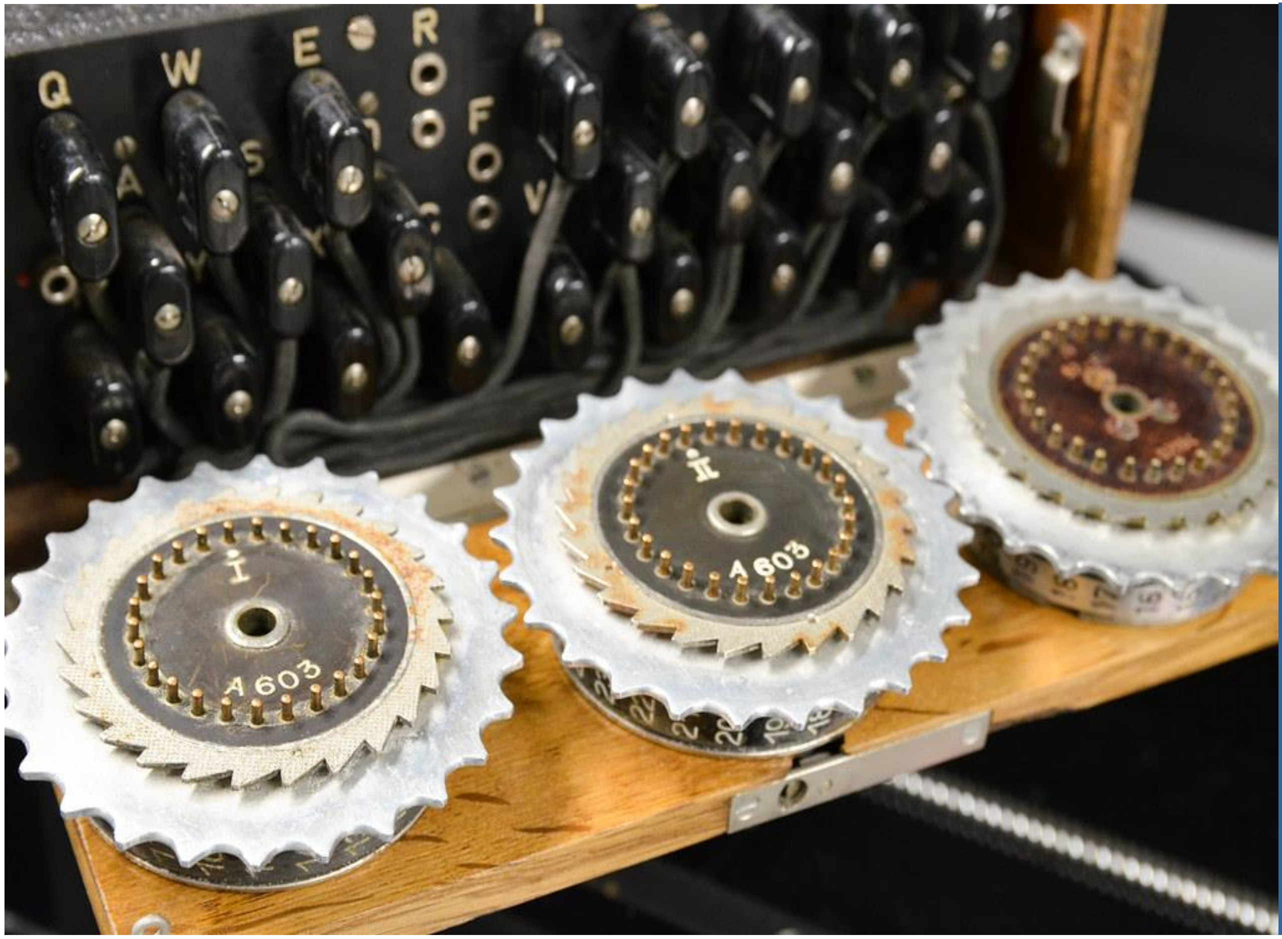
# Developer Productivity in a Cloud Native World

*An Intellyx Analyst Guide for Chronosphere*

*By Jason Bloomberg and Eric Newcomer*

## Table of Contents



**By Jason Bloomberg**

Founder & Managing Director
Intellyx

# The Three Paradoxes of Cloud Native Platform Engineering

Part 1 of the Developer Productivity in a Cloud Native World Series

Cloud native computing has come to dominate enterprise IT for every organization that requires dynamic flexible infrastructure at scale.

Kubernetes and the rest of the cloud native ecosystem Kubernetes and the rest of the cloud native ecosystem enable companies to build and deploy applications faster, more reliably and at a lower cost but can be complex and difficult to master. The promises of speed and scale may seem out of reach when development teams get caught up in the details. To build such software, developers require all the help they can get.

DevOps practices and tools are essential but are merely the starting point. To succeed with DevOps in cloud native environments, organizations hope to build out internal developer platforms (IDPs) following still-nascent platform engineering best practices.

***An excessive focus on developer productivity can backfire. Platform engineering can counterintuitively drive developers away. And the more complex cloud native deployments become, the more difficult they are to manage.***

The goal of platform engineering is to improve developer productivity. And if development teams are productive, then they should be able to deploy software at the velocity and scale that the business requires.

Even with IDPs in place, however, the goals of cloud native development set a high bar for the organizations that seek its benefits. For every step forward, they seem to take one step back.

An excessive focus on developer productivity can backfire. Platform engineering can counterintuitively drive developers away. And the more complex cloud native deployments become, the more difficult they are to manage.

This article – the first of a series of four – will explore these paradoxes. How can organizations achieve the business goals for their software efforts when so many pitfalls threaten their success?

## The Developer Productivity Paradox

Everybody agrees that developer productivity is a good thing. The more productive developers are, the more software they can deliver for every dollar their organizations pay them.

The problem with developer productivity, however, is how to measure it.

McKinsey tackled this question in a recent controversial article: *Yes, you can measure software developer productivity*. In that article, McKinsey theorized that such measurement can improve software development outcomes.

In particular, McKinsey posited that when developers focus their activities on building, coding, and testing software, then they are being productive. Other 'soft' activities, including planning, thinking, mentoring, and architecting, suck away developer productivity.

Given this perspective, it's clear why developers are pushing back. Any productivity measurement approach that follows McKinsey's recommendations will entirely miss the fact that developers that focus their efforts on the 'soft' part of their jobs are in truth more productive and valuable to their organization overall as compared to their colleagues who spend all their time with their hands on their keyboards.

Complex software initiatives like cloud native only exacerbate this disconnect over developer productivity. In the cloud native mindset, the most productive developers are the ones that focus on what makes for dynamic and scalable software that meets the business need, independent of how much time they spend coding.

Developers don't like to be measured with vanity metrics like number of pull requests. To optimize productivity, it's best to let the developers themselves decide what activities they should focus on.

## The Platform Engineering Paradox

Given the plethora of DevOps tools on the market, assembling the optimal toolchain can slow everyone down and lead to inconsistent results.

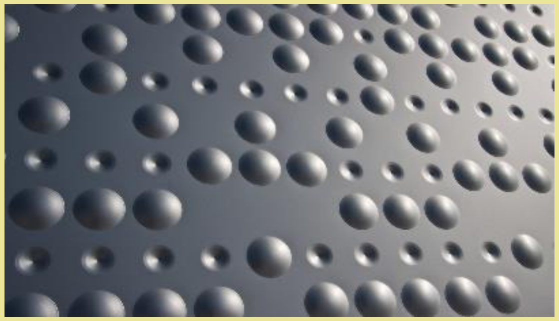
The solution: task a platform engineering team to build an IDP that includes the best set of tools for the tasks at hand. The goal of such a platform is to provide a 'golden path' for developers to follow, essentially a recommended set of tools and processes for getting their work done.

However, this golden path can become a straitjacket. When this golden path is overly normative, developers will move away from it to get their jobs done, defeating its purpose.

As with measuring their productivity, developers want to be able to make their own choices regarding how they go about crafting software.

As a result, platform engineers must be especially careful when building IDPs for cloud native development. Jumping to the conclusion that tools and practices that were suitable for other architectural approaches are also appropriate for cloud native can be a big mistake.

For example, observability tooling is an important component of any IDP, but most observability tools are a poor fit for the needs of cloud native developers. Instead, an observability tool like Chronosphere running on Google Cloud will give cloud native developers the insights they need to do their jobs, even in complex and dynamic environments.



***The only way to maintain an appropriate focus on the big picture while developing microservices is to have a grasp on all relevant data about the performance of the cloud native infrastructure.***

## The Cloud Native Paradox

This cloud native mindset brings an architectural focus to the work of developers.

Cloud native developers build individual microservices but must also keep tabs on the behavior of pods in clusters and clusters in fleets of clusters. In other words, they must focus on the forest while simultaneously focusing on the trees.

Developer productivity depends upon this dual focus. Any successful platform engineering effort does as well.

How, then, to resolve this forest vs. trees paradox? The answer lies in the data. The only way to maintain an appropriate focus on the big picture while developing microservices is to have a grasp on all relevant data about the performance of the cloud native infrastructure.

Too much data, however, is worse than too little, which is why cloud native observability from tools like Chronosphere, coupled with the cloud native mindset that Google Cloud brings to the table are essential for achieving the business goals of cloud native computing.

## The Intellyx Take

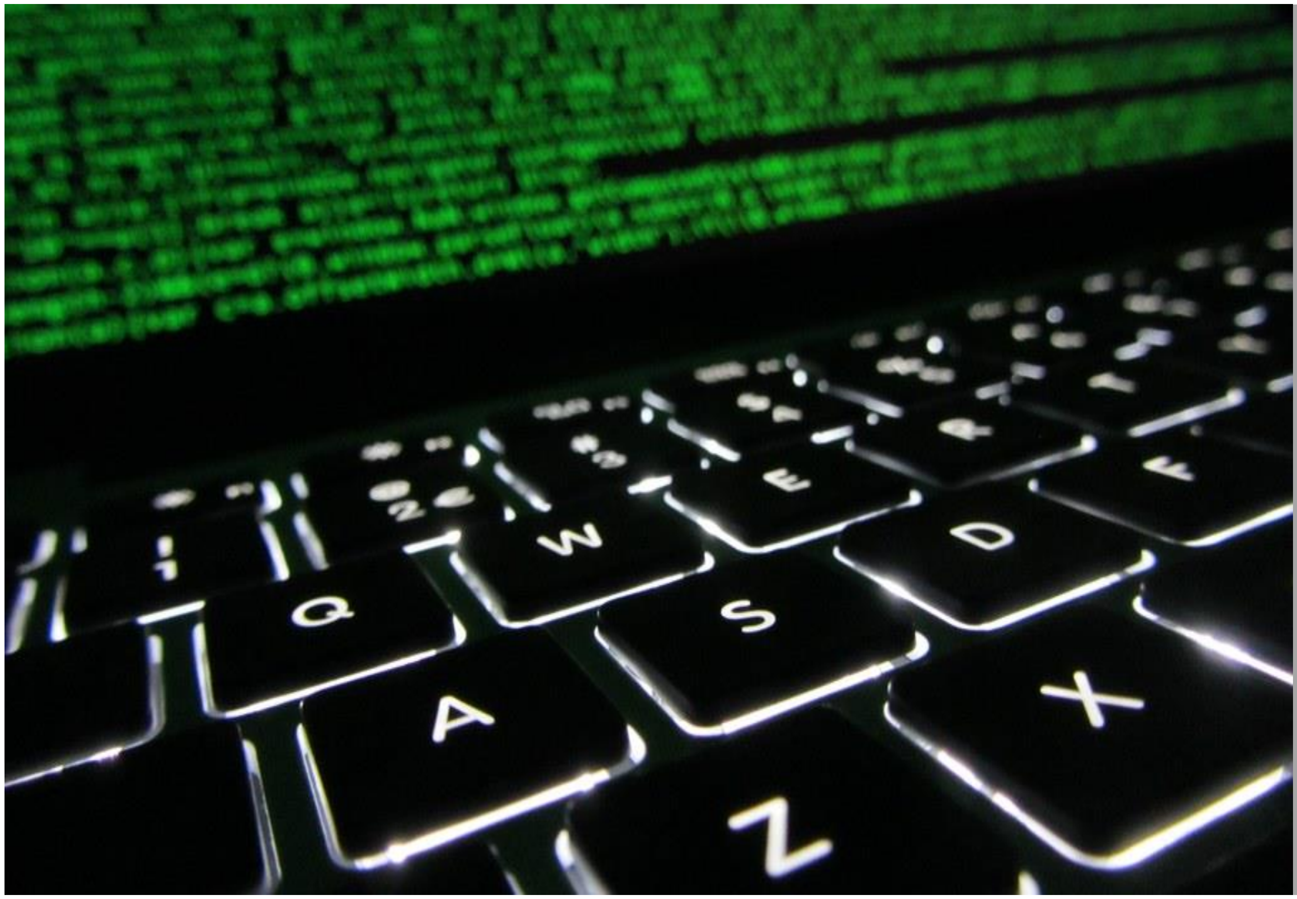
Resolving the three paradoxes in this article depends upon achieving the right balance.

Too much focus on developer productivity is counterproductive, but too little will leave areas for improvement unexplored. The answer: listen to your developers to strike the right balance.

An overly constraining IDP will lead to developers finding ways around it, defeating its purpose. Building an IDP that follows the cloud native mindset will help to achieve the proper balance.

Encouraging developers to grasp the big picture of cloud native computing at scale is essential for achieving the goals of the software effort, but without the proper observability data, they will never find the balance between this big picture and their day-to-day work of building and deploying microservices.

The entire notion of a cloud native mindset, therefore, comprises all these balancing acts. Listen to your developers, give them the right data, and let them find the balance.



**By Eric Newcomer**

CTO & Principal Analyst
Intellyx

# Why Platform Engineering is Different for Cloud Native Applications

## Part 2 of the Developer Productivity in a Cloud Native World Series

As the article above described, creating an internal developer platform (IDP) for cloud native computing is challenging because cloud native computing is different and complex and therefore it's challenging to get it right.

Getting it right, however, is key to achieving cloud native benefits for the business, enabling developers to build and deploy applications more quickly, reliably, and at lower cost.

Understanding why developer platforms are different for cloud native computing is an essential first step to getting it right.

## The Cloud Native Difference

Cloud native applications differ significantly from traditional applications primarily because they are designed and developed for cloud native "scale out" infrastructure, and typically use multiple cloud provider services.

Public cloud providers offer hundreds of system software services, any number of which may be relevant (or not) to the task at hand. It can be very time consuming to sort through these services to identify the ones you need to incorporate in your IDP.

Furthermore, developing cloud native applications typically involves breaking functions into microservices, packaging the microservice code into a container image with required artifacts, and deploying the containers to Kubernetes clusters.

Going down this path helps achieve cloud native benefits, but all software tools are not equal when it comes to microservice, container, and Kubernetes support.

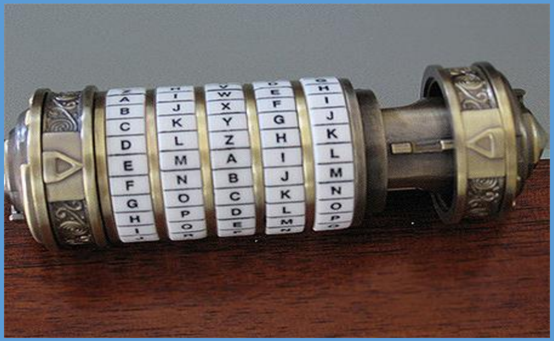
Traditional tools often don't support cloud native developers because they are not designed for the cloud native computing environment of microservices containers, and Kubernetes.

## Successfully Adopting Microservices

A microservice is a discrete unit of work, typically an application feature or function. A microservice exposes an API to encapsulate its function and interacts with other microservices over the network.

Cloud native developers need to develop and deploy microservices rapidly and at scale to keep pace with the modern pace of change and respond quickly to customer feedback and incidents.

Organizations successfully adopting microservices achieve the greatest benefits with small autonomous teams supported by a central internal development platform that enforces organizational standards and consistent tooling.



***Organizations successfully adopting microservices achieve the greatest benefits with small autonomous teams supported by a central internal development platform that enforces organizational standards and consistent tooling.***

It's much easier to patch and update an individual microservice than it is to redeploy a monolithic application just to fix a bug, for example.

Success with this model requires the right organizational structure and governance, the discipline to carefully coordinate breaking changes that impact multiple microservices, and the right combination of cloud native technologies and tools in your IDP.

## Platform Engineering for the Cloud

The goal of platform engineering is to create an IDP that works as a product for developers that abstracts away infrastructure issues and tooling concerns.

And because of the differences in the cloud native environment compared to traditional IT environments, these abstractions must work with microservices, containers, Kubernetes, and cloud provider services.

Cloud native best practices put additional responsibilities on developers. In traditional IT environments developers request the operations team to provision their databases, event stores, and secrets vaults, for example. Cloud native environments do not have operations teams that manually provision required software.

Instead, developers use an IDP that exposes declarative APIs, configuration files and scripts so developers can self-serve.

Consider a request to provision a database. The SRE or operations teams would have set up available databases in the IDP that meet security, availability and reliability concerns and monitoring “out-of-the-box.”

A developer declares a requirement for a relational database, for example, without having to specify which one. This level of abstraction permits the database to be swapped out for another database without impacting the application.

## Observability for Cloud Native Development and Deployment

Let’s assume that you work in a cloud native environment and develop microservices using containers and Kubernetes.

Let’s further assume that you also follow the best practice of organizing your developers into small autonomous teams responsible for at least one microservice throughout the entire development lifecycle, including production support.

All of this is to provide a great customer experience, develop and deploy rapidly, increase developer productivity, and release application code to production more quickly and with higher quality and resilience.

The success of these choices significantly depends on including the right observability tools in the IDP.

Traditional observability tools only monitor post-deployment code -- that is, they monitor code in staging and production. This is of course important but it's also important to reduce the number of incidents and outages that take time away from developing and deploying code that improves the customer experience and business competitiveness.

Observability solutions that simply capture data are inefficient and impede productivity. What's needed in the cloud native world of microservices, containers, and Kubernetes is a solution that  filters and tailors data specifically for small autonomous teams developing, deploying, and supporting microservices. Google Cloud and Chronosphere provide this for IDPs.

## The Intellyx Take

Engineering for cloud native applications is how you get the best value and greatest benefit from cloud native computing.

Engineering an IDP for cloud native environments requires not only abstracting the cloud provider service infrastructure but also incorporating an observability solution that provides immediate feedback as early and often in the development process as possible.

Observability in the IDP designed for cloud native applications provides useful data to developers as quickly as possible to keep them productive and happy.

Google Cloud and Chronosphere offer such a production-ready observability solution that can be included in your IDP to help fully realize the benefits of cloud native environments.

**By Jason Bloomberg**

Founder & Managing Director
Intellyx

# Developer productivity vs. developer experience:
## Does cloud native make a difference?

Part 3 of the Developer Productivity in a Cloud Native World Series

In the first article in this eBook, I discussed the three paradoxes of cloud native platform engineering: how measuring developer productivity can inadvertently lower it; how platform engineering efforts can backfire by being overly normative; and how cloud native developers must focus on individual microservices as well as the behavior of pods and clusters simultaneously.

In the second article, my colleague Eric Newcomer explored in depth the challenges platform engineers face when building internal developer platforms (IDPs) for cloud native developers.

He concludes that success requires the right organizational structure and governance, a good measure of discipline, and the right combination of platform tools in the IDP – in particular, observability tooling that tailors data specifically for the small autonomous teams that develop, deploy, and support microservices – services like Chronosphere and Google Cloud.

There are two threads that weave their ways through these arguments. The first is *developer productivity* – how much individual developers and teams can accomplish in a given amount of time.

The second thread is every bit as important: *developer experience*, or DX. DX is a measure of how effectively developers can do their jobs with the tooling and platforms available, combined with morale and overall job satisfaction.

The more challenges thrown at developers, the more difficult it becomes to produce quality work consistently. Simultaneously, pressure to increase productivity in such situations can have a deleterious impact on DX – a situation that will surely backfire.

Cloud native is one area that presents such challenges. Building microservices in the context of a cloud native deployment at scale is difficult and complex. How, then, should organizations balance developer productivity and DX for cloud native initiatives? And ideally, how to improve both at once?

## Why Cloud Native Development is so Challenging

Cloud native development begins with microservices. And while there are many definitions of microservices (many not 'micro' at all), one popular definition is a *parsimonious, cohesive unit of execution.*

'Parsimonious' means as small as possible, but no smaller – the 'micro' part of the story. 'Cohesion' is a well-known characteristic of good software that means that each microservice does one thing and one thing well.

'Unit of execution' refers to the fact that microservices running in containers include the runtime components necessary to execute – an intentional contrast to the earlier generation of Web Services that were XML-based endpoints to software that required enterprise service buses (or other technology like servlet engines) to provide the execution context.

***To maintain productivity, cloud native developers must work with the DevOps team to keep track of all these microservice properties for every microservice all the time – keeping in mind the functionality of each microservice within the broader cloud native architectural context.***

Within the broader cloud native architectural context, furthermore, microservices require additional considerations. They automatically scale up and down within pods, and thus developers must design them to be stateless. As a result, managing state with microservices requires special care to gain the benefits of dynamic scaling.

To maintain productivity, cloud native developers must work with the DevOps team to keep track of all these microservice properties for every microservice all the time – keeping in mind the functionality of each microservice within the broader cloud native architectural context.

In other words, cloud native development is difficult and complicated – challenges that can easily impact both developer productivity and DX.

## Maintaining DX in the Face of Challenging Development

The good news: developers love a challenge. Most of them chose their career because of the inherent satisfaction in solving difficult puzzles. There's no better feeling than when a particularly difficult piece of code finally runs the way it's supposed to.

There is a fine line, however, between challenges that are fun and fulfilling vs. problems so difficult that working on them is like pounding one's head against a wall. For many developers, cloud native development is more of the wall-pounding kind.

Both developer productivity and DX depend upon shifting this equation toward the sorts of challenges that developers love. IDPs can help, as they provide developers with a recommended set of tools – but having the right tools is critical.

Of all the tools one might find in an IDP that contribute most to both productivity and DX are *observability* tools. Difficult problems are fine as long as developers have the information they need to solve them – and observability tooling like Chronosphere provides that information.

Not only does Chronosphere provide observability specifically for developers, it also specializes in observing cloud native environments.

Chronosphere has designed their platform to meet the observability needs of the cloud native developer. In particular, Chronosphere Lens goes beyond raw telemetry by dynamically generating service-centric views that support developers in both proactive exploration and reactive troubleshooting of system behavior.

## The Intellyx Take

When management focuses too much on developer productivity the DX can suffer, and thus adversely impact morale and paradoxically, productivity as well. It's important for management to have a light touch to avoid this problem, especially with cloud native.

Cloud native environments can become so dynamic and noisy that both productivity and DX can decline. Management must take special care to support its developers with the right platforms, tools, processes, and productivity metrics to facilitate the best outcomes – leveraging platform engineering to create and manage IDPs that facilitate cloud native development despite its inherent complexity.

After all, the complexity of cloud native development alone isn't the problem. Complexity presents challenges to be sure, but developers are always up for a challenge. Complexity coupled with a lack of visibility brings frustration, lowering productivity and DX.

With the right observability, for example, with Chronosphere and Google Cloud, developers have a good shot at untangling cloud native's inherent complexity, delivering quality software on time and on budget, while maintaining both productivity and DX.

## About the Analysts

Jason Bloomberg is founder and managing director of enterprise IT industry analysis firm Intellyx. He is a leading IT industry analyst, author, keynote speaker, and globally recognized expert on multiple disruptive trends in enterprise technology and digital transformation.

Mr. Bloomberg is the author or coauthor of five books, including *Low-Code for Dummies*, published in October 2019.

Eric Newcomer is CTO and Principal Analyst at Intellyx, a technology analysis firm focused on enterprise digital transformation. Eric is a well-known technology writer and industry thought leader.

He previously served as CTO at WSO2 and at IONA, led Security Architecture for Citi's consumer banking and was Chief Architect for Citi and Credit Suisse's trade and investment divisions.

## About Intellyx

Intellyx is the first and only industry analysis, advisory, and training firm focused on customer-driven, technology-empowered digital transformation for the enterprise. Covering every angle of enterprise IT from mainframes to cloud, process automation to artificial intelligence, our broad focus across technologies allows business executives and IT professionals to connect the dots on disruptive trends. Read and learn more at https://intellyx.com or follow them on X at @intellyx.

## About Chronosphere

Chronosphere is the only observability platform that puts you back in control by taming rampant data growth and cloud native complexity, delivering increased business confidence. Chronosphere is backed by venture capital investors Greylock, Lux Capital, General Atlantic, Addition, and Founders Fund. The team is distributed, with major hubs in New York, Seattle, and Vilnius. Learn more at https://chronosphere.io.