ambassador

Transforming Developer Productivity with Kubernetes

A guide to boost the cloud native developer experience for your team

Cloud-native, Kubernetes-based software development changes the game not just for developers but also for site reliability (SRE) and ops engineering teams. It creates a need for closer collaboration, changes each of these roles, and demands a centralized, paved path for developers to be able to create and run their apps.

To capitalize on the velocity cloud-native development makes possible, SRE and ops teams provide and support a developer experience that includes a self-service approach to developers taking on ownership of the full development lifecycle: not just code but also ship and run.

In interviews with SRE and platform teams, developers, and leaders from across industries and types of organizations, a number of common perspectives emerged about the new cloud-native normal.

This "shift left" includes the importance of:

- Developer ownership and enabling the "you build it, you run it" approach
- Understanding the changing roles of sre and ops teams in relation to developers, i.E. More consultative, fostering empowerment and support, less firefighting
- Developer education and organizational support for offering hands-on education and training
- Leadership buying in to and championing the end-to-end "developer as service owner" mindset to make it work across an organization
- A paved-path platform or developer control plane along with opinionated workflows to reduce developer cognitive load, get developers up to speed faster, reduce tool sprawl, and create ideal conditions for shipping software faster, safely.

The site reliability engineer perspective

We recently spoke with **Mario Loria, Senior Site Reliability Engineer (SRE)** from CartaX, an electronic marketplace for private securities. In a wide-ranging discussion that covered ground from the changing developer experience to the ideal role of SREs in a modern, cloud-native environment, a number of **key themes** emerged:

Mario Loria

Senior Site Reliability Engineer (SRE)

CartaX

An organization and its leadership needs to get behind the end-to-end "developer as service owner" mindset to make it work.

Developers should own the full life cycle of services but in most cases don't. Mario explained, "It should not be up to me as an SRE to define how your application gets deployed or at what point it needs to be rolled back, or at what point it needs to be changed, or when its health check should be modified." Developers should be capable of — and empowered — to make these determinations.

Developer education and mindset will need to change to embrace the "you build it, you run it" approach, with SREs helping to shape and support the developer-ownership mindset with appropriate platforms providing tools and an interactive self-service experience instead of riding to the rescue when things go wrong.

In this new environment, one of the best things SREs can do is focus on infrastructure and core services to support the main challenge and goal of a developer: shipping software safely at speed. The developer doesn't necessarily need to care about what platforms and tools are used but does need to be able to use them to, for example, canary a service or get service metrics.

Site reliability engineers (SREs) play a key role in guiding developers through the learning curve toward comprehensive self-service of the supporting platforms and ecosystem, and ultimately to service ownership.

To hand over complete ownership to developers, greater transparency and visibility into what's going on with their services is needed. To "liberate" developers from an overreliance on SRE firefighting, and facilitate developer autonomy in finding their own solutions, **a developer control plane centralizes and ties together the code-ship-run processes a developer needs to understand**.

The platform architect's perspective

In an **extensive interview** with Lunar Bank's Lead Platform Architect, Kasper Nissen, the discussion ranged from centralizing tooling for developers and creating opinionated workflow, for example in the form of a developer control plane as a way to provide a single pane of glass and reduce the complexity of building, shipping, and running applications for developers taking on full ownership of the lifecycle. Some of the **key takeaways** included:

Kasper Nissen

Lead Platform Architect

Lunar

Shifting left and putting more responsibility onto developers shoulders requires organizations to deliver education and support to enable developer upskilling and promote a frictionless developer experience.

Platform and ops teams should create a "paved path" to reduce tools sprawl and accelerate developer ramp-up and productivity, which helps both developers and the ops teams.

Giving developers centralized tooling and opinionated workflows (such as developer-friendly GitOps workflows) empowers them to take ownership of the full code, ship, run equation and removes the cognitive burden of trying to learn a range of infrastructure that has very little to do with their work.

As a part of the shift left, a balance of freedom and responsibility for developers is a must. To get developers onboard, platform and opss teams have a responsibility to make that move manageable. As Kasper shared, "Platform teams can't just put all this responsibility onto developers and say, 'Go figure it out and become experts in all these different systems.'"

"Kubernetes is just the common framework. It's all the other stuff that's the hard part now: linking everything together and making it work", according to Kasper. If Kubernetes is accepted as a default, platform teams can focus on the developer experience touchpoints that are required components of rapidly and safely shipping software.

The fundamental aim is to set up a **self-service approach so that developers can take a quick pulse of the system: both the "single pane of glass" to understand what is going on under the hood and the "developer control plane" to integrate these different activities and control them centrally.**

The cloud-native leadership perspective

ambassador

Bjorn Freeman-Benson, the SVP of Engineering at Ambassador Labs, recently **shared his perspectives** on cloud-native development and how it has evolved, where it's going, and how to support the changing developer experience. Important insights included:

Bjorn Freeman-Benson

SVP of Engineering

Ambassador Labs

Enabling the new cloud-native paradigm requires a focus on uptime, collaboration across roles and departments, and SRE and ops teams facilitating developer self-service.

Development and operations teams can gain the most efficiency by working together. Complete separation of duty means that other teams have no insight into what the others are doing. As Bjorn points out, developers are more likely to have easier insight into emerging failure patterns and problems within their own services, while a separate operations team would have to spend significant time digging around to identify what is going wrong.

Collaboration is not the same as duplicating effort. As Bjorn explained SRE and ops teams' time is better spent building robust foundations to automate all the things development teams need to use but don't need to know inside and out. Essentially, everyone is focused on enabling developer self-service, which makes everyone's job easier and lets each function focus on their areas of expertise.

In production, customers prize uptime and availability over features. While both are important, customer expectations about what a service delivers focus mostly on uptime. Customers are considerably more upset about outages, failures, and poor performance than they are about a feature that doesn't ship. Supporting the business model and customer expectation, then, means shifting the development model, making the "run" component of "code-ship-run" balance.

As a part of the shift left, platform teams building tooling and automation, paving the path for developers, and, along with SREs, keep it clear of obstacles. "If you're building a SaaS piece of software, it's about operations as well as writing the software. That's the extension we've made as developers — to go from just developing to developing and operating. The friction of this shift in ownership is where the developer control plane concept was born.

A **DCP gives developers what they need to control and configure the entire cloud-development loop to ship software faster, without the distraction of trying to find and figure out a million different tools**. The developer's focus and creativity is better spent on creating, shipping, and running software that delivers value.

The developer's perspective

In a **chat about the developer experience**, the challenges of infrastructure, and the future of cloud-native development, former software developer at Google and, more recently, VP Ecosystem at the Cloud Native Computing Foundation (CNCF), Cheryl Hung, shared **notable themes**:

Cheryl Hung

VP Ecosystem

Cloud Native Computing Foundation (CNCF)

The much-discussed shift left, which puts more responsibility for shipping and running software in the hands of developers, has its upsides and downsides. Many developers just want to code, and companies like Google, enable this. But this ease can be a tradeoff: The developer never needs to learn how it works under the hood, which isn't helpful if a developer goes to work in a company that insists on developer ownership and autonomy. The developer is shielded from understanding the complexity of shipping and running their code. This can be positive, keeping the developer focused. At the same time, it removes the responsibility for considerations like provisioning resources, which would be valuable knowledge for full-ownership developers.

Infrastructure is hard, complex, and easy to get wrong. It's important for developers to get enough understanding of this both to identify problems in their code throughout the lifecycle and to reduce strain on the SRE teams that support them.

Providing a centralized source of truth creates a good developer experience, lessening the learning curve and providing a clarity of experience for developers without limiting their ability to seek out and learn platform tools beyond that portal.

An emerging common view among developers, architects, and SREs across different business types is that more developer self-service is better. This supports the thinking that platform and SRE teams can best support developers by creating the right abstraction layer to empower developers to do their jobs, regardless of how much ownership they have over the full process.

Balancing freedom with responsibility is central to empowering developers to move to an ownership mindset with the software they develop. Cheryl mirrored these experiences: "The more self-service you can provide from the platform to the application developers, the better. It saves time on both sides, and empowers both the developer and the platform team to focus on their core areas."

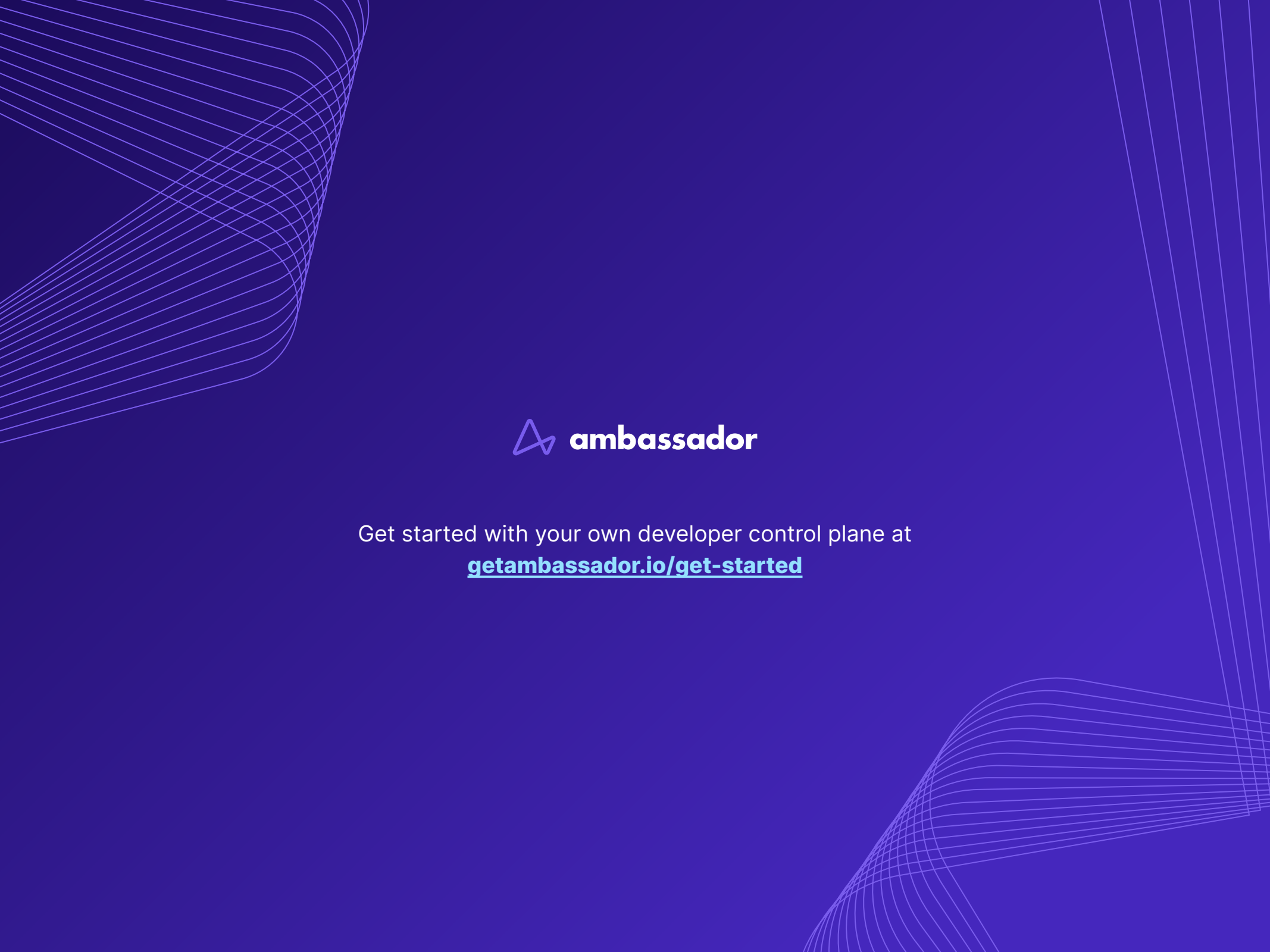
Cloud-native developers can **thrive given a happy medium, that is, by having a developer portal or control plane, it's possible to set a baseline experience without tying a developer's hands if and when they want to dive deeper into the underlying infrastructure or swap tooling**.

Centralizing the developer experience

Cloud-native development has forced organizations and the people working in it to reimagine software development. Development teams are adopting new tools and workflows, which has fundamentally changed the developer experience. The shift left has blurred lines between developer, operations and site reliability teams. Both the changed development experience, platforms and tools and shifting customer expectations make the case for:

- Full lifecycle ownership for developers, which supports efficiency
- Infrastructural and automation support from SRE and operations (not just firefighting), particularly with implementing a DCP
- Closer collaboration across teams that makes development and release more efficient from end to end.

Power developer ownership of the full development life cycle with a developer control plane, which enhances your existing technology stack and enables collaboration among your development teams without requiring devs to worry about managing configuration.



Get started with your own developer control plane at
getambassador.io/get-started