ActualTech Media

PLATFORM9

Creating an Optimal DevOps Experience with Distributed Kubernetes

CONTENTS

IN THIS PAPER

Developers need a consistent, stable platform to do their best work. That can be tricky when the platform is Kubernetes-based, as its distributed nature can create a lot of workflow friction. But using the right tools and implementing proper standards can go a long way toward greasing that pipeline.

Introduction

Kubernetes is widely recognized as a platform that enables highly efficient use of infrastructure, but organizations need to understand those benefits are maximized when the developer experience itself is optimized.

Developers are increasingly assuming responsibilities for systems operations. In the past, it was common to have a separate team of systems administrators responsible for deploying applications, monitoring resource use, and responding to incidents that disrupted services. Developers who use agile methodologies are more likely to employ practices that include responsibility for ensuring their software operates efficiently and reliably.

> The developers who are working in the code and revising it are in the best position to understand the cause of performance or reliability problems.

This is understandable, since one aspect of agile engineering practices is the frequent release of new versions of services. Rather than hold up the release of an update so that multiple features can be included, it's more efficient to release small changes continually. Continuous integration/continuous delivery (CI/CD) pipelines, coupled with version control platforms that promote collaboration, enable this kind of rapid release of new features. It also means that the developers who are working in the code and revising it are in the best position to understand the cause of performance or reliability problems.

PLATFORM ENGINEERING OPTIMAL EXPERIENCE

Developers depend on a stable environment to work. This entails high uptime, reliability, and performance. Platform engineering teams should treat the platform as a product. They provide this platform for developers, enabling them to create services for their customers.

This includes building teams, processes, and a culture that continually improves—not just sustains—the platform. Using agile approaches, developers can deliver initial applications on a platform they manage— but expect to have platform engineers take over responsibility for the platform.

Kubernetes can help deliver the optimal engineering experience. It's designed to automate and orchestrate reliable computing resources for containerized applications. One important aspect of Kubernetes is that it can be deployed in multiple environments, including:

- Centralized data centers, either on-premises or colocated in a third-party data center
- Micro data centers used in remote offices
- Point-of-presence locations such as retail stores
- Edge computing settings for Internet of Things (IoT) deployments

This ability to deploy Kubernetes to a wide variety of environments is a significant advantage over deploying customized, case-specific servers. With a single, common platform for executing workloads, developers can spend less time on operational issues with the help of tooling that supports the Kubernetes platform.

CONSISTENT POLICIES, PRACTICES, AND TOOLS

Consider the challenges of complying with regulations and policies while maintaining an agile, rapid-feature-delivery engineering environment. There are multiple dimensions of compliance that must be attended to.

For example, developers, who are also operations managers, need tools to help ensure authentication mechanisms are in place. In many cases, authentication and identity management services are provided by a centralized service that needs to be accessible from various Kubernetes deployments.

Highly distributed systems like Kubernetes are constantly generating, storing, and transmitting data. Many regulations governing privacy and the control of sensitive information have rules about protecting the confidentiality of data. To meet these requirements, it's a best practice to employ encryption for data in motion and for data at rest.

Kubernetes environments should be deployed in ways that provide these encryption services by default. Application

developers shouldn't have to learn the intricacies of configuring full disk encryption or setting up TLS connections between nodes. Role based access controls (RBACs) are essential for securing the platform. Given the large number of services and tenants, this can be a difficult task and requires tooling to support and maintain proper RBAC configurations.

Kubernetes should be deployed with controls in place to support other governance requirements. For example, security scans should be configured to run reliably on all clusters. Again, this is a necessary capability, but not one that should require significant developer time.

Tooling should be in place to help with capacity planning and cost control. Kubernetes is designed to allocate resources to workloads that need them. Those resource demands can, and often do, change over time, so it's important to monitor resource utilization and growth rates in workloads. If a cluster has insufficient resources, developers may be forced to limit features or find other workarounds to deal with the lack of capacity. Poor capacity planning can introduce significant friction in the development process and slow the creation of new services.

Organizations are increasingly adopting multi-cloud platforms, so you'll need to consider integration of different systems. Legacy on-premises applications and servers may be used alongside servers running in a public cloud, for instance. Kubernetes is well suited to these kinds of deployment models, but there must be tooling in place to maintain the reliability of these systems.

THE NEGATIVE IMPACT OF SHADOW IT

When appropriate tooling isn't in place and there's insufficient centralized support, developers will likely develop their own solutions to operational challenges. For example, when platform tools like CI/CD pipelines aren't centrally standardized, departments or teams of engineers may implement their own solutions.

This is problematic for several reasons. For one, it's inefficient to have multiple teams duplicating work. It also means that individual teams are responsible for maintaining tools and ensuring they're deployed in compliance with policies and regulations. They also become responsible for ensuring that all service-level agreements (SLAs) are being met.

These kinds of shadow IT practices lead to inconsistent management practices. Instead of a common operations model, organizations are left with a fractured DevOps situation that makes it more difficult for teams to collaborate. Teams will develop different procedures and use different tools, and this often means each team takes on learning on its own and may not benefit from what others have experienced.

Clearly, a consistent set of policies, practices, and tools across an organization is essential to maintaining an optimal developer experience. It's also important to consider what might be required for an application owner's optimal experience.

The Application Owner's Optimal Experience

Application owners have an obvious stake in ensuring an optimal developer environment. Key considerations from their perspective include:

- Ensuring developers have needed resources
- Standardizing on commonly needed resources and middleware
- Using tools to streamline package management within Kubernetes deployments

KEY RESOURCES FOR DEVELOPERS

Key resources for developers span the development cycle. There should be support for full stack development. UI developers typically work with frameworks for creating complex Web interfaces, while back-end developers are more likely to need tools to help optimize high-performance code.

Tooling should also include support for version control and CI/CD. These tools are becoming more feature-rich and integrated so that as soon as changes are checked into a repository, they can trigger a build, with testing and eventual release to follow.

Service discovery and application catalogs are important for ensuring developers know the kinds of services available in the environment. These tools can foster the sharing of services and reduce redundant code.

STANDARDIZING COMMONLY NEEDED RESOURCES

Application owners should also consider standardizing commonly needed resources and middleware. For example, multiple services may need a relational database back end. There are many high-quality options to choose from, including both open source and commercial products.

While different relational databases have distinct features and capabilities, application owners must ask if the cost of supporting two or more databases is outweighed by the benefit of those specialized features. In many cases, the economics favor standardizing on a single kind of database.

It's also important to make shared components available in a central catalog that are available for developers to easily deploy with a few clicks. This provides the governance that the operations teams need and the self-service agile experience that developers crave.

STREAMLINING PACKAGE MANAGEMENT

Similarly, organizations should standardize on load balancers and monitoring tools. While different load balancers may have different features, the core job of a load balancer isn't likely to vary much among services running in the same environment.

A single, consolidated monitoring tool should be selected as well, with performance metrics collected in a single tool. This allows for more comprehensive analysis of performance monitoring data than if the data were spread across multiple tools.

Logging and distributed tracing tools are also important for understanding the state of your systems, identifying bottlenecks, and understanding the root causes of performance problems.

Service meshes, like Istio, provide additional services on top of Kubernetes (see **Figure 1**). Standardizing on a single service mesh across all deployments of Kubernetes will also improve the overall utility of Kubernetes from a developer and application owner perspective.

For all of the benefits of Kubernetes, there are some challenges to using the platform. Within a single cluster, dozens of packages may be deployed, all of which must be

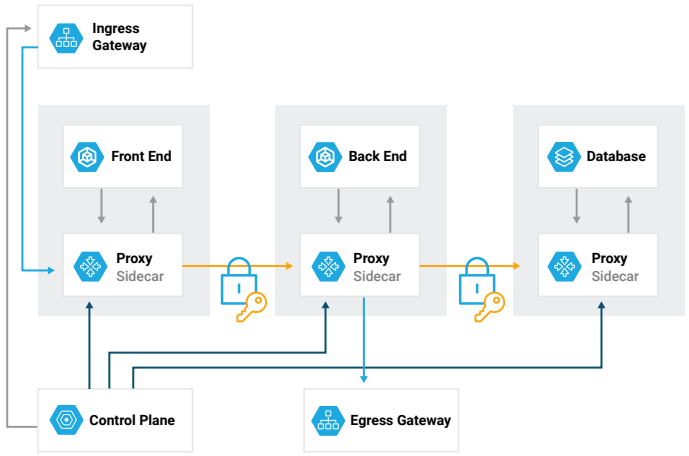

Figure 1: Service mesh traffic overview

monitored and maintained along with other applications. It can be a challenge to keep track of packages and their state in a single cluster, but the workload is multiplied when you include Kubernetes deployments in distributed and edge computing environments.

You need automation to support package management. Fortunately, Helm and Kustomize are two such package managers that can streamline package management.

Platform9 Can Help

The promise of Kubernetes to more efficiently employ computing and storage resources is best realized when you take into account how Kubernetes is used and maintained by developers. Kubernetes is complex, and as responsibility for managing clusters moves from a small number of clusters in a single data center to hundreds or thousands of distributed clusters, there's a risk of not knowing how to run such a distributed platform optimally.

Platform9 has experts that not only understand Kubernetes, but have the real-world experience with large-scale deployment to help you with your Kubernetes initiatives.