DevOps 2.0 is here, and it's time to put end-to-end continuous delivery pipelines behind every project.

Max Martynov

May 08, 2019 · 8 min read

Implementing end-to-end continuous delivery at the enterprise level has always been a challenge. It is a complex problem that requires simultaneous improvements across an organization's structure, culture, application architecture, infrastructure and change management processes.

This challenge is made more difficult by the fact that focusing improvement on only one aspect rarely leads to the desired results, because they are all closely linked and reinforce one another as part of an evolutionarily stable strategy. And attempting to improve all aspects at once is almost impossible at the enterprise level, due to the cumulative level of effort required. Different companies apply different strategies and achieve varying levels of success, but it is rare to find a company that has successfully achieved the right balance between efficiency, productivity and speed.

However, the situation has been changing since the implementation of clouds, containers, microservices and automation across all levels of the software delivery process. Now, in 2019, technologies and best practices have evolved to a point where we can confidently say that the continuous delivery problem has been solved at the enterprise level. Of course, we are not talking about mainframes, but it is true for most brownfield modern applications and definitely the case for all greenfield development.

All the necessary capabilities and techniques are now sufficiently well-known, and most are covered by technologies at the necessary levels of maturity. There remain some occasional gaps, however, they are being quickly addressed. And the big players, like Google, are already providing products and services for this new vision.

The main contributor to solving the problem has been the establishment of well-defined interfaces between the application, infrastructure, platform and change management processes. It has helped to untangle organizational and technological dependencies and implement solutions for individual

problems.

Once the interfaces were established, infrastructure and platform components, which were the biggest bottlenecks, were quickly served with stable and feature-rich technologies from well-known cloud providers, such as: Google, Amazon and Microsoft. Remaining gaps have been filled with open source technologies released by large technology companies, such as: Netflix, Facebook, and LinkedIn, which had already spent large amounts of time and resources battling these problems internally.

Every team has their own developers, QAs, SRE, and other skills to develop, deliver, and support their service

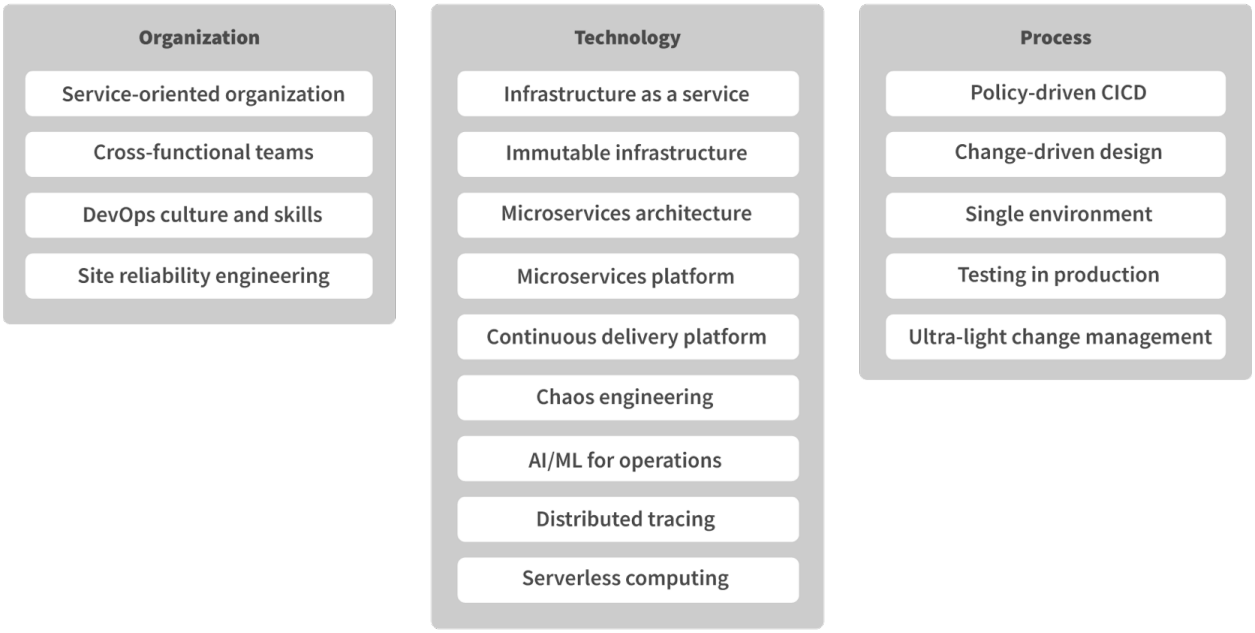
This division allowed for the establishment of service-oriented organizations, where each team develops and manages their own service end-to-end from development to production. Now infrastructure teams can deliver Infrastructure as a Service based on a robust cloud foundation. Platform teams use Infrastructure as a Service and provide microservices management platform capabilities as a service to application teams, closing the gap between raw infrastructure and applications.

Even change management processes can now be implemented in a platform and provided as a service to application teams, with change management policies enforced at the enterprise level without limiting the flexibility of application teams. Finally, application teams can focus on application development and delivery. This can now be achieved without sacrificing efficiency through having human dependencies on infrastructure or change management teams or by implementing missing infrastructure and platform pieces.

New capabilities of DevOps 2.0

We will use the term DevOps 2.0 to describe the enterprise-level capability that solves the continuous delivery problem. Although the term DevOps has been overloaded, it is a well-known

term, and is often used in a similar fashion. DevOps 2.0 brings a number of additional capabilities in the areas of organization, technology and process:



Implementing them brings the state of continuous delivery and DevOps to a completely new level of organizational efficiency, speed and productivity, while retaining visibility and control.

In this next section, we will provide a summary of what each capability means, what value it brings, and how it is supported with an open source or cloud-native technology stack. It is important to note that we are not touching traditional capabilities such as versioning, testing, monitoring, logging, security and others. It is possible to look at the technology advancements in these areas separately.

Organization

In a **service-oriented organization**, every team is working on their own product, which is available to other departments and teams as a service via an API (not self-service portals or emails). This allows organizations to scale indefinitely without a loss of efficiency. Since each team becomes a product team responsible for the requirements, development, testing, release and support of their product end-to-end, it increases the team's agency, leading to overall higher employee satisfaction.

Teams working on a service should be **cross-functional teams** to ensure the ability of the team to deliver the end result with minimal human dependencies on other teams. It doesn't necessarily mean that each engineer in the team should possess all skills across development, testing, infrastructure, deployment, CICD, security, etc. Although excess specialization doesn't lead to good results, healthy division of labor helps with productivity, and the organization's ability to hire and train engineers.

DevOps culture is enabled by a service-oriented organization and cross-functional teams. In many companies with existing DevOps departments, such departments should be built according to the principles of a service-oriented organization. In practical terms, it means that DevOps organization is split in the following way:

1. Microservices platform - this works on top of vanilla Infrastructure as a Service and bridges the gap between application and infrastructure. This is achieved by implementing technology capabilities of service registry and discovery, secret management, deployment, application lifecycle management, service mesh, logging, monitoring, distributed tracing and others.
2. CICD (change management) platform - this implements core tooling for build, code review, continuous integration, continuous deployment, test harness, audit trail, management dashboards and others. The platform is also used to enforce corporate change management policies according to executable policy principles.
3. Specialized deployment and CI engineers embedded into application teams, where application developers lack skills in these areas.

The **site reliability engineering** team replaces or augments traditional production support teams with the mission to convert reactive support efforts to proactive ones, increasing overall system SLOs.

Technology

Infrastructure as a Service or **Cloud**, is a foundation on which applications can be deployed and managed over a well-defined and battle-tested interface. Enterprises tried to create internal IaaS offerings for some time, however, reliability, stability and interfaces were generally subpar. This prevented the establishment of a foundation for application deployment automation and the creation of a truly service-oriented organization. Public cloud vendors changed the game, providing reliable infrastructure services. Large companies such as Google have now moved into private data centers, with offerings such as GKE on premise or Anthos, making reliable infrastructure interfaces possible even on premise.

Immutable infrastructure, especially in the areas of compute and storage, dramatically reduces configuration issues during deployments and configuration drift in environments. It includes creating self-containing containers or VMs with application code and configuration during the build and CI cycle. It follows the "build once, deploy everywhere" principle, with minimal configuration differences between environments.

Immutable networking infrastructure is an advanced technique that includes load balancing, firewall and in some cases subnet configurations. It is possible with modern cloud providers or platforms such as Kubernetes and Istio. Immutable infrastructure requires careful separation of application and environment configuration (properties). Application configuration typically includes memory, CPU and middleware settings that may seriously affect application stability, and are treated as application code from a change management perspective. Environment configuration is dynamic, includes endpoints of upstream dependencies, secrets and feature flags, and is provided to the application by microservices platform interfaces during runtime. It can be treated as data from a change management perspective.

Microservices architecture is a well-known capability that requires well-designed small services that communicate with each other over API and can be delivered by small teams, which in turn follows the two-pizza rule for determining their size.

The **microservices platform** provides a layer of additional services on top of the vanilla Infrastructure as a Service interface. It includes capabilities such as packaging, deployment, service registry, secret management, application lifecycle management, service mesh, etc. Major

technologies include Kubernetes, Istio and Hashicorp stack. You can read more about it in one of our previous blog posts.

The **continuous delivery platform** provides tooling to implement the end-to-end change management process from requirement management, source code repository, CI server, test harness, continuous deployment infrastructure and others. The most common tooling includes JIRA, Git, Jenkins, and Spinnaker. You can read more about it in one of our previous blog posts.

Chaos engineering is both a process and technology capability. From a technology perspective, the chaos engineering platform should include necessary monitoring, logging, and failure induction tooling. The most common tools include Chaos Toolkit, Gremlin and Simian Army.

Distributed tracing and OpenTracing standard is the new way of monitoring end-to-end flows in microservices environments and service mesh. Open source tooling includes Istio, Jaeger, Zipkin, and many others. It is still a new capability, but it is maturing quickly.

Serverless computing further reduces the need to manage infrastructure, middleware, and platform components from an application standpoint. It allows for increases in the efficiency and productivity of application teams for certain types of applications. It is enabled by technologies such as AWS Lambda, Google Functions, and Knative.

AI and ML are becoming necessary components of successful production operations. Production environments generate plenty of logging and monitoring data to do anomaly detection, perform predictive scaling and augment RCA. Change management platforms generate significant data that can be used for the optimization of processes and for determining best practices. These best practices can later be distributed across teams in the organization.

Process

Policy driven CI/CD takes high-level enterprise change management policies but redesigns processes and tools to automatically satisfy those policies. All sign-offs from development leads, test leads, security leads and operations leads are now implemented as executable policies and embedded into the pipeline. For example, a policy with a sign-off from the development lead may be implemented as a required code peer review, static code analysis rules and unit test success rate and coverage.

With such a policy, each build passing these requirements automatically records development lead sign-off in the audit trail with the explanation and proof of how the policy was satisfied. Such policy definition can be created and changed only by the development lead, and it is of course versioned by itself. Since it is executable and automated in a CI process, it doesn't reduce the efficiency of the development team from one side, ensuring full control over change management, and compliance with internal and external audit requirements from another. While such policy-driven CICD is not supported by CI platforms directly, it can be implemented in tools like Jenkins or Spinnaker.

Single environment solves non-production environment creep and increased costs on dev/test environments. With the right microservices architecture, platform, service mesh and CICD process, services don't need a large number of huge integration testing environments. Services can be tested in isolation on small environments during the CICD cycle and can be directly deployed to production for functional and user acceptance testing, canary releases and A/B testing.

At this point, it is important to understand that deployment in the production environment doesn't automatically mean acceptance of user traffic, meaning that the production environment can serve as an integration testing environment for properly designed applications. In the worst case scenario,

during the transformation, the single environment can be split into two: production and staging, with the staging environment used for the last phases of quality assurance.

Testing in production is a technique where significant testing is done directly in the production environment on the services that are deployed in blue-green mode, and are not accepting user traffic. This requires well-designed applications and service mesh capabilities, and in turn, enables single environment capability.

Ultra-light process strives to keep most high-level change management policies in place, but implements them with completely new processes, procedures and tools. It is enabled by all capabilities outlined above, and uncovers the ultimate value of CICD.

Closing notes

We are confident that building DevOps 2.0 capabilities will bring digital organizational efficiency and productivity to a new level, and truly solve the continuous delivery problem. While it doesn't cover certain legacy applications, we found that it works for many modern brownfield applications and all greenfield development.

This article provides a high-level overview. If you are interested in more details:

1. Read our blog post on microservices platform.
2. Check the presentation we've been showing at Dynamic Talks.
3. Get the Continuous Delivery Blueprint book.
4. Reach out to us for a deep dive.

Of course, even when the end state is clear, the transition will still require time, effort, and skills. But the result will be well worth the effort.