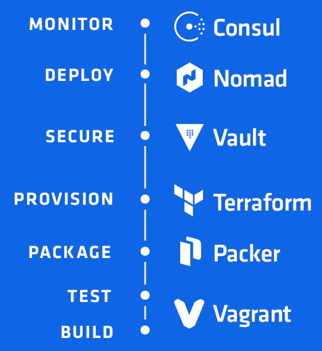
DevOps Defined

Accelerating the Application Delivery Lifecycle



Defined | **Delivered**

The Problem

The definition of DevOps varies from business to business, but the zeitgeist of DevOps is about minimizing the challenges of shipping, rapidly iterating, and securing software applications. HashiCorp defines DevOps as an organizational process tied to the needs of modern applications, with a focus on empowering individuals to improve agility.

> *DevOps is about minimizing the challenges of shipping, rapidly iterating, and securing software applications.*

DevOps primarily involves the people responsible for delivering applications, including developers, operators, and security professionals. These three interdependent roles need tightly coupled tools to coordinate their contributions to application delivery.

DevOps is a movement away from the Waterfall Model of software delivery. In the Waterfall Model, software applications are delivered as a linear sort of waterfall through various groups. Developers receive requirements, and write the application before handing off to Quality Assurance for testing. After the development phase, the application is handed to a release team for packaging and user acceptance testing. When testing is complete, security experts are brought in to ensure compliance and best practices. Eventually, operators deploy the application, and the final stage of the waterfall lands on the monitoring team.

The problem with the traditional Waterfall software delivery model is that it prioritizes minimizing risk instead of maximizing agility. Waterfall restricts individual autonomy, slows feedback loops, and requires many teams and checkpoints for every small change to the application.

DevOps is about allowing the participants in this process -- Operations, Security, Development -- to work in parallel. We do this by deconstructing the essential elements of the application delivery process and providing a tool best suited for each participant and task. The end result is a process that prioritizes agility, time to value, and a more continuous integration and continuous delivery model.

The rise of DevOps is also tied to the rise of hybrid cloud infrastructure, characterized by distributed services and data center resources. Modern applications are Internet-connected and have thin clients such as browsers and mobile apps. Updates can be delivered quickly and there is often no "recall" that requires more disciplined risk management.

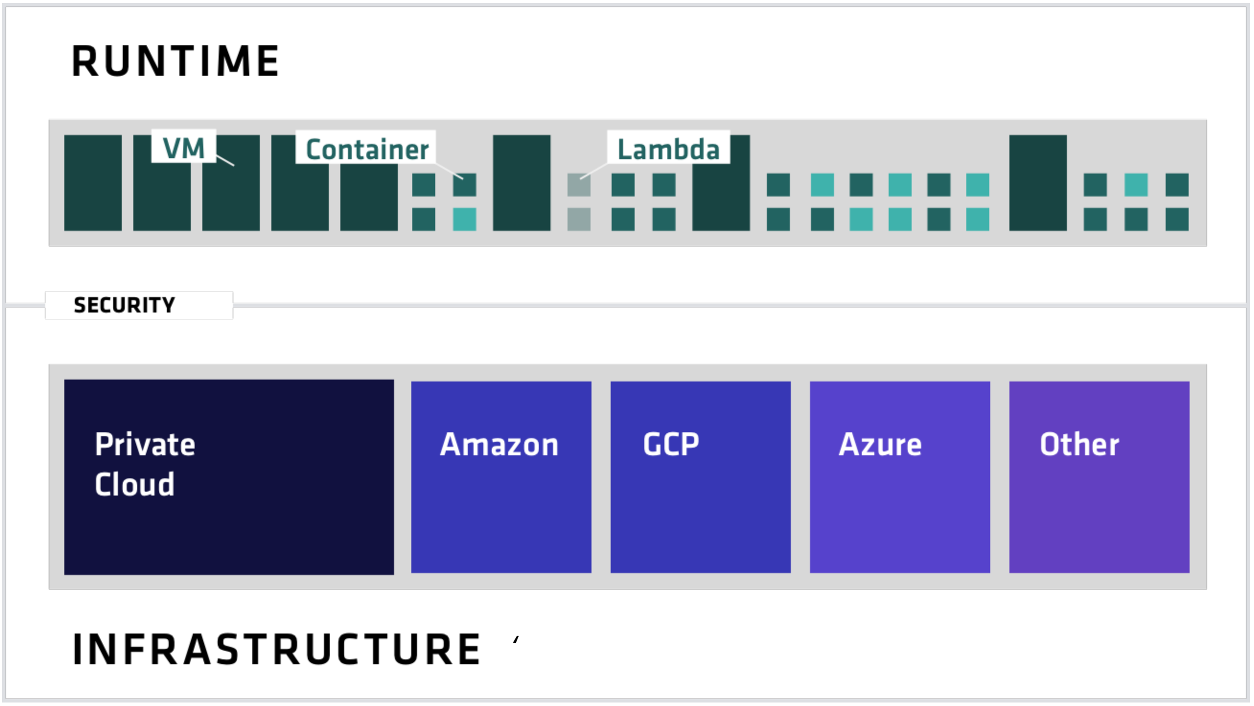


Figure 1. *The rise of DevOps is tied to the rise of hybrid cloud infrastructure, characterized by distributed models for application services and data center resources.*

DevOps done right maximizes the velocity of software delivery. By viewing the entire delivery process holistically, we can solve the bottlenecks that traditionally happen when one role in the process is overloaded; because at the end of the day, software can only be delivered as fast as the slowest team.

DevOps Defined

The Application Delivery Lifecycle

Every organization has slightly different elements in its software delivery process, driven by technology choices, compliance requirements, or other factors. But if you look at the whole forest and not just the trees, there are seven elements to the software delivery lifecycle:

BUILD An application starts with a developer writing code. For a new application the initial version must be written, but for existing applications there is a perpetual cycle of adding new features and functionality, fixing bugs, and improving performance. This element largely involves developers, but operations teams may be responsible for providing the environment and tools developers are using to write code.

TEST While an application is being written and prior to release, it goes through multiple types of testing. The simplest tests, unit testing, is done by developers, and can be layered with integration testing, acceptance testing, end-to-end tests, etc. This is an important part of the application delivery lifecycle, as it provides automated feedback and an important risk management control. This largely involves developers, but also dedicated QA teams and operations teams who may own the testing infrastructure.

PACKAGE Once an application is written and tested, it needs to be packaged for production. The packaging can depend heavily on the technology or target environment. For example, this might be a WAR file for JBoss, or a Docker container for Kubernetes. The underlying application is being packaged from its raw source form into a packaged executable for the target environment. These packages are stored in a registry such as Artifactory.

PROVISION Applications require somewhere to run. Under all the layers of abstraction there is still compute, storage, and networking resources that must be provided. These might be provided directly with bare metal or VMs, or indirectly through a PaaS or Lambda-as-a-Service frameworks. In any case, these resources must be provisioned and configured to the application requirements, updated over time, and finally decommissioned at the end of their utility. Provisioning is usually owned by operations teams and provided to developers.

SECURE Overall security of a system is only as strong as its weakest link. This means security is involved from the beginning of the application delivery process. Security teams help ensure best practices are used during development, assist in modeling network topology, protect the credentials used to provision infrastructure, and grant secrets needed to deploy applications such as database passwords and API tokens. Security typically falls to dedicated roles on a teams, but involves all other teams that are delivering an application.

DEPLOY With the underlying resources provisioned, deployment means taking the packaged application and running it. This can be tightly coupled with provisioning if machines or VMs are specialized to run a single application, or decoupled if a scheduler is used to dynamically place applications on machines.

MONITOR Running applications need to be monitored to ensure they continue to run and are healthy. Services need to talk to each other while avoiding communicating with faulty or degraded instances. This leads to the need for service discovery. Monitoring ranges a wide gamut from coarse grained liveness to detailed logging and telemetry. Monitoring involves developers who want to understand the behavior of their applications, operators who manage the infrastructure, and monitoring and site reliability teams who maintain the broader application.

DevOps Delivered

Provision, secure and run any infrastructure for any application

Designing a high performance organization is similar to designing a high performance application: it must require minimal coordination.

For software this is best captured by Amdahl's law, but if we think of individuals as "serial execution units" productivity is similarly dominated by coordination.

The seven elements of delivering an application cannot be skipped, so the priority must become minimizing the coordination required to perform each step. If each team is empowered to work independently, then coordination can be reduced and individual productivity can be increased; application delivery velocity increases. This is the heart of DevOps, and the tools we choose must prioritize these key functions. To maintain consistent process, those tools must focus on workflows, not technologies, to address the technical heterogeneity that is the reality of most organizations.

HashiCorp provides a suite of tools with DevOps in-mind focused on reducing manual coordination across the elements of application delivery lifecycle.

Vagrant

Vagrant allows developers to quickly setup a development environment on their own, without needing to consult peers or operators. By providing a production-like environment, it also allows developers to easily test their code with a tighter feedback loop. This is one of the goals of Continuous Integration (CI), as it allows developers to be more individually productive by giving them feedback more rapidly.

To learn more about Vagrant visit https://www.vagrantup.com.

Packer

Packer provides a single workflow to package applications for any target environment. By sharing configuration, Packer allows teams to be decoupled and avoid coordination. The coordination is pushed into the artifact registry such as Artifactory or Docker Hub.

To learn more about Packer visit https://www.packer.io.

Terraform

Terraform is a product to provision infrastructure and application resources across any infrastructure using a common workflow. Terraform enables operators to safely and predictably create, change, and improve production infrastructure. It codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.

To learn more about Terraform visit https://www.terraform.io.

Vault

Vault provides a centralized approach to secrets-management across every element of the application delivery lifecycle. Vault provides a highly available and secure way of storing and exposing secrets to applications and end users; for example, encryption keys, API tokens, and database credentials. Vault allows teams to consume the data they need without coordinating with security teams. Security teams can change passwords, rotate credentials, and update policies without coordinating across the organization.

To learn more about Vault visit https://www.vaultproject.io.

Nomad

Nomad is a cluster manager and scheduler. Schedulers allow an organization to decouple concerns even further, and abstract machines away from developers entirely. Instead, developers focus on the applications they want to run, and allow a scheduler to place the application and manage capacity of the machines. Nomad allows operators to provision a fleet of machines, decoupled from developers who are submitting jobs to Nomad. Nomad places the applications on available machines, allowing operators and developers to avoid manual coordination.

To learn more about Nomad visit https://www.nomadproject.io.

Consul

Consul is a service discovery and monitoring tool. Consul allows applications that are running to broadcast their availability, and makes them easily reached by other applications. For example, web servers can use Consul to find their upstream databases or API services. Consul also monitors the health of applications to ensure only healthy instances receive traffic, and it notifies developers or operators of any issues. This allows development teams to avoid coordinating on IP addresses, and pushes discovery into the runtime of the application, so services can be updated independently. This ties into the rise of microservices and service-oriented architectures. Productivity is improved by having independent services updated by separate teams in parallel, without the kind of coordination required by a monolithic code base.

To learn more about Consul visit https://www.consul.io.

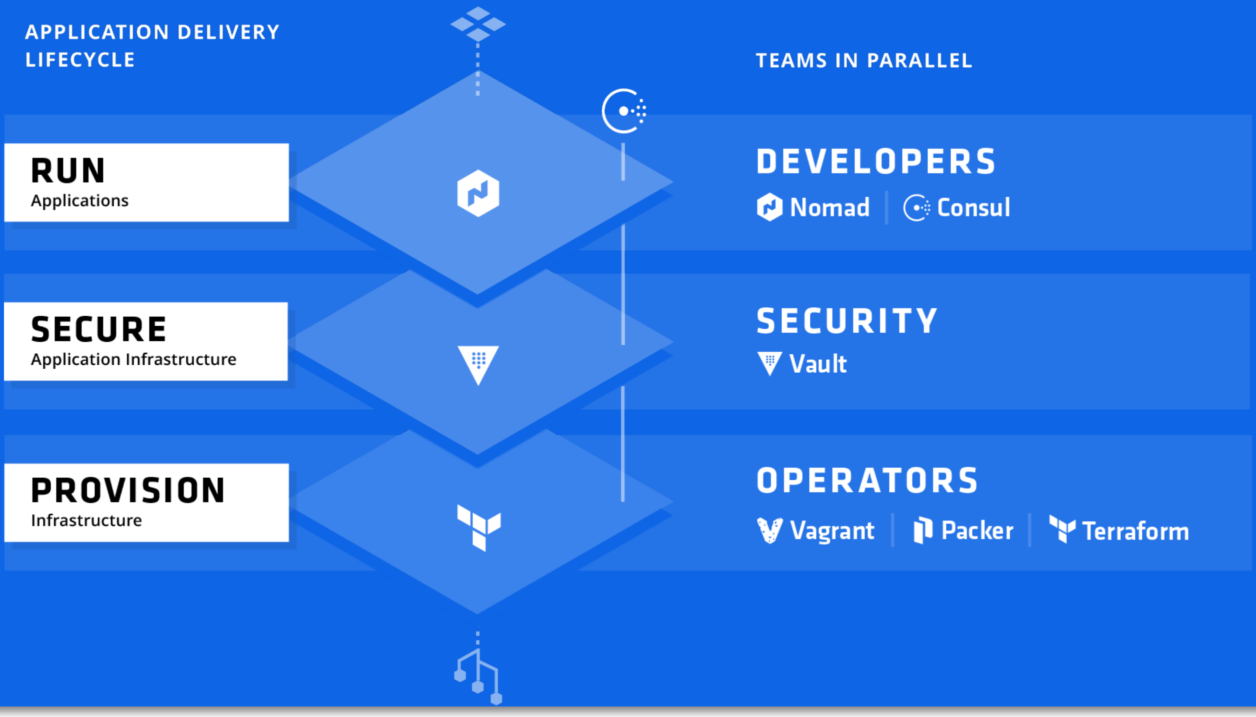
DevOps Done Right

Allowing Operations, Security and Development teams to work in parallel

As every company becomes a software company, the ability to execute a DevOps model allows them to deliver better applications, faster. And hundreds of thousands of software professionals globally are using the HashiCorp DevOps Suite to achieve this.

By providing a tool specifically designed for each of the elements of DevOps we allow the different participants in the software supply chain -- development, operations, and security -- to focus on their primary concern while unblocking their peers. This means turning what was a linear waterfall process into one where all three teams can run in parallel.

It is the parallelization of these workflows that is the essence of DevOps: the ability to Provision, Secure and Run any infrastructure for any application.


APPLICATION DELIVERY LIFECYCLE
TEAMS IN PARALLEL
RUN
Applications
DEVELOPERS
Nomad
Consul
SECURE
Application Infrastructure
SECURITY
Vault
PROVISION
Infrastructure
OPERATORS
Vagrant
Packer
Terraform