HashiCorp

Application Delivery with HashiCorp

Application Delivery

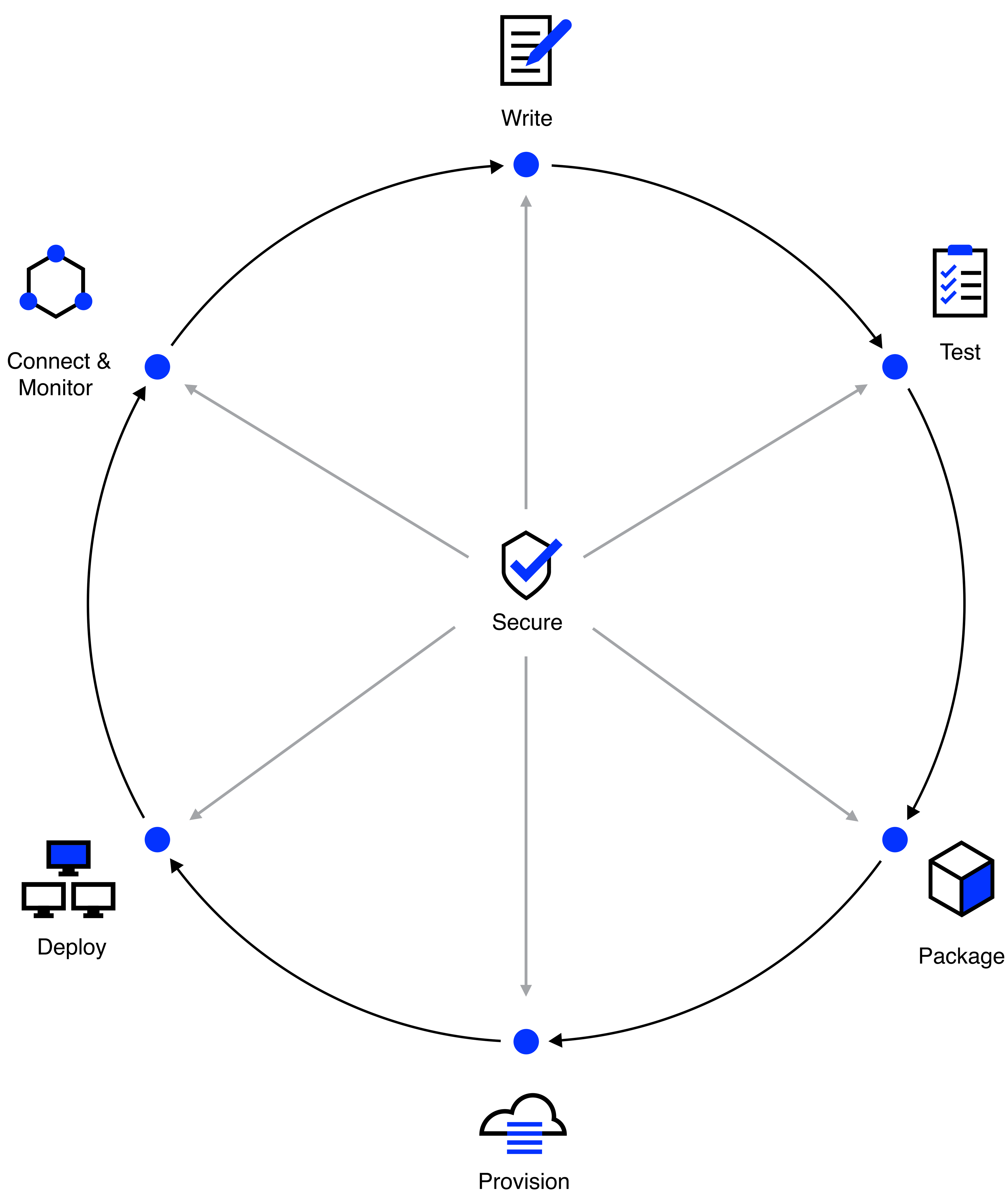
At HashiCorp we are focused on providing the tools that allow organizations to adopt the cloud and automate their infrastructure, with the end goal of accelerating application delivery. This means we have a wide span on concerns, including provisioning, securing, running, and connecting applications. Our approach applies a set of infrastructure management principles to the application delivery lifecycle delivered through a suite of products. In this document, we describe the application delivery lifecycle and the HashiCorp approach to modern application delivery.

01

Application Delivery Lifecycle

Application Delivery Lifecycle

Software is never finished, and instead evolves to add new features, fix bugs, and solve new problems. This means software delivery is a continuous lifecycle and not a one time event. We believe this lifecycle has seven steps which are necessary and sufficient for most organizations:



Application Delivery Lifecycle

Together these steps are both necessary and sufficient for delivering most applications. Depending on the organization additional steps may be required between these anchor points, but often times they can be integrated into these steps. These steps are technology agnostic and reflect the workflow challenges in delivering an application, regardless of the underlying technology choices.

Write

Every application needs to be written and modified, regardless of language, framework, or OS.

Test

Every application should be tested, with various levels of granularity from unit, integration, and acceptance testing.

Package

Source code and configuration management are transformed into a compiled artifact suitable for production environments. This can be a static binary, JAR file, DEB/RPM package, Docker container, or VM image.These artifacts are typically versioned and stored.

Provision

Applications require infrastructure to run on. This can be low level storage, compute, and networking provided by an Infrastructure-as-a-Service (IaaS) provider, or it could be a higher level service abstraction provided by a Platform-as-a-Service(PaaS). In any case, that underlying infrastructure has a lifecycle that requires acquiring and configuring initially, managing throughout the lifecycle, and destroying or releasing when no longer needed.

Deploy

The process of mapping artifacts to underlying infrastructure is deployment. This step can be tightly coupled with provisioning if applications are baked into machine images, but typically is a distinct step which is done more often than provisioning. Configuration management tools often "specialize" a machine after provisioning to deploy an application while schedulers dynamically place applications on infrastructure out of band from provisioning.

Connect & Monitor

Once an application is deployed, we want to ensure it continues running and connect it to other applications. Monitoring includes an entire spectrum including health checking, logging, application performance monitoring, request tracing, and more. Connecting includes service discovery, load balancing, and more. The goal is to ensure applications and infrastructure systems are healthy and performing within a service level objective (SLO), while providing observability tooling to diagnose issues as they arise.

Security

Throughout this entire lifecycle we must be concerned with security. While other steps can be considered in isolation without regard to the entire process, security demands a wholistic approach. If security is bolted on a single step, an attacker will target the other unprotected steps, forcing practitioners to adopt a "weakest link" mentality.

02

Principles of Infrastructure

Infrastructure Management

The continuous evolution of software means application delivery is naturally modeled as a continuous process as well; the infrastructure that supports an application must continuously evolve to meet the changing requirements of the application. For teams building, managing, or supporting applications and infrastructure, we are tasked with two critical responsibilities:

Managing Complexity

As an application gains new features and functionality or inherits constraints such as compliance requirements, it typically becomes more complex. Over time, a simple two tier application has N-tiers of frontends, backends, storage, and middleware that are tightly integrated. This is a natural reflection of the essential complexity of the application. The goal is to minimize accidental complexity and the creation of a Rube Goldberg machine. By minimizing complexity, we make it simpler to reason about the system and reduce the friction of evolving the application.

Managing Risk

As an application becomes increasingly business critical, there is an increased sensitivity to risk. There is no way to entirely eliminate risk, especially when applications and infrastructure are changing and co-evolving. Instead, we can implement various controls and processes to reduce risk. For example, as a simple application matures we should demand more regression testing, automate more tasks, perform canary or blue/green deploys, and invest in more monitoring. By reducing risk, we make it cheaper to experiment and iterate on the application.

While managing complexity and risk are the guiding principles of infrastructure management, there are a number of techniques we can apply to achieve them:

Workflows over Technology

Many tools are focused on solving application lifecycle problems in a technology specific way, such as deployment for JBoss applications. This forces us to adopt unique management tools for each technology. Instead by focusing on the fundamental workflows, and making them technology agnostic, we can unify the management tooling. This reduces the overall complexity by moving technology-specific tooling, and makes it simpler to test and adopt new technologies without requiring new management tooling reducing risk of experimentation or lock in.

Infrastructure as Code

Traditional approaches to infrastructure have relied on point-and-click interfaces or manual operator configuration, however these approaches are prone to human error. Treating infrastructure as code allows us automate execution and apply the best practices of software development, including code review and version control. This technique helps manage complexity by reusing code, providing up-to-date documentation, and allows for modular decomposition of the problem. It reduces risk by allowing code review, versioning, and automated execution.

Policy as Code

This is a closely related approach to Infrastructure as Code, but instead focuses on codifying the business policies and automating enforcement. Traditionally, compliance and security teams would define policies in plain language documents and enforce them manually through a ticketing flow. By turning policies into code, we benefit from version control, code review, code reuse, regression testing, and automation. This allows enforcement to be decoupled from the definition of policy, and to be automated. Policy as Code allows a "sandbox" to be defined to minimize the risks associated with Infrastructure as Code. Additionally, it reduces the burden of compliance, and minimizes the risk of human error or oversight.

Immutability

Often times servers are provisioning and subsequently evolved manually by operators or by configuration management tools. As servers are evolved generation over generation, any variations or errors cause a fragmentation of versions. This can be caused by human operators performing different steps or automated steps failing silently. This means servers are not running version 5, but a continuous set of versions from version 4,…, 4.63,.., 5. Immutability instead pushes for golden images and discrete versions. This means a server is either running version 4 or 5, without an in between state. This reduces complexity because we can more easily reason about a small number of well known versions instead of a distribution of many unknown versions. It reduces risk because we can test and certify versions, perform rollbacks, and avoid transient failures.

Modular and Composable

By decomposing a problem into smaller sub-problems we make it easier to reason about each problem in isolation. By taking a modular approach and composing various tools and systems together, we make it easier to understand how things work and simpler to iterate on. Monolithic approaches tightly couple concerns, such that changing a small part of the system forces considering much larger scope. This increases the complexity of the overall system, and introduces a large amount of systemic risk to making changes.

Service Oriented Architecture

In order to support a composable approach, a service oriented or micro-services architecture can be used. This prescribes autonomous, well scoped, and loosely coupled applications communicating over the network. By allowing each service to encapsulate implementation details behind an API, we can work on problems in isolation and iterate on each service independently. This reduces the complexity of solving sub-problems and reduces the risk of making changes to the application.

What Cloud Changes

The application delivery lifecycle and principles of infrastructure management are fundamental and apply to any technology. However, there are several properties of the cloud which affect the tools and techniques we use:

API driven

The biggest difference between Public and Private Clouds from the previous generation of infrastructure is the API driven nature. Instead of a manual ticket driven process, the entire lifecycle of infrastructure is now managed programmatically. This enables richer automation that runs orders of magnitude faster and more efficiently.

Elasticity

The API abstraction of clouds is an "infinite" resource pool of compute. Instead of Rack-and-Stack with upfront CapEx, resources are provisioned on demand and treated as OpEx. This means instead of provisioning for peak capacity, the API driven nature allows the server fleet to expand and contract to accommodate load and "right size" the capacity. This requires automation of provisioning and application deployment, made simpler by infrastructure as code and declarative models.

Failures

Cloud service providers offer minimal to no SLAs around the failure rate of machines. Instead, customers are expected to architect their applications to tolerate failures. This encourages service oriented architectures to built resiliency against failures at the application layer, and drift correction at infrastructure layer again made simpler by infrastructure as code and declarative models.

No Network Perimeter

The API contract of the cloud means that we lose the simpler "four walls" abstraction of a physical datacenter with fixed ingress and egress points. Any machine can be programmatically setup to ingress or egress traffic to the Internet. Instead of assuming an impregnable network that is trusted, a more practical and secure assumption is that the network has been or will be compromised. Under this model, centralized security middleware becomes less effective as an attacker is assumed to be inside the trusted segments. When assuming zero trust, applications must encrypt data in transit and at rest while enforcing authentication and authorization of clients. This increases the demand for tooling and automation to make this simpler for developers.

Multiple Providers

Lastly there are many viable cloud service providers. Each provider offers a roughly similar abstraction at an IaaS layer, but the non-standard APIs and semantic differences make it increasingly important to adopt tooling that provides a common workflow to avoid increasing the complexity of tooling and process while avoiding the risk of vendor lock in.

03

People and

People and Roles

There is a deep amount of domain knowledge for each step in the application delivery process, making it impractical for an individual to be an expert every area. Instead, there is a natural specialization of knowledge and a process which allows all the domain experts to collaborate together. Typically the following groups are involved:

Developers

The developers are knowledgable in programming languages, frameworks, and application design. Their productivity is limited by the feedback loop between writing code and testing, which are their primary concerns. They can be further empowered with self-service deployments and observability tooling to diagnose application issues.

Operators

The operators are knowledgeable about cloud service providers, infrastructure automation, and networking. They support developers and the infrastructure for applications. Typically they are responsible for system stability and uptime, and focus on maturing the infrastructure to increase the mean time to failure (MTTF) and reduce mean time to recovery (MTTR).

Security Analysts

The security team is knowledgeable about threat modeling, vulnerability management, secret management, and privileged access. They act as consultants to developers and operators, and ensure compliance targets are met and risk is appropriately managed.

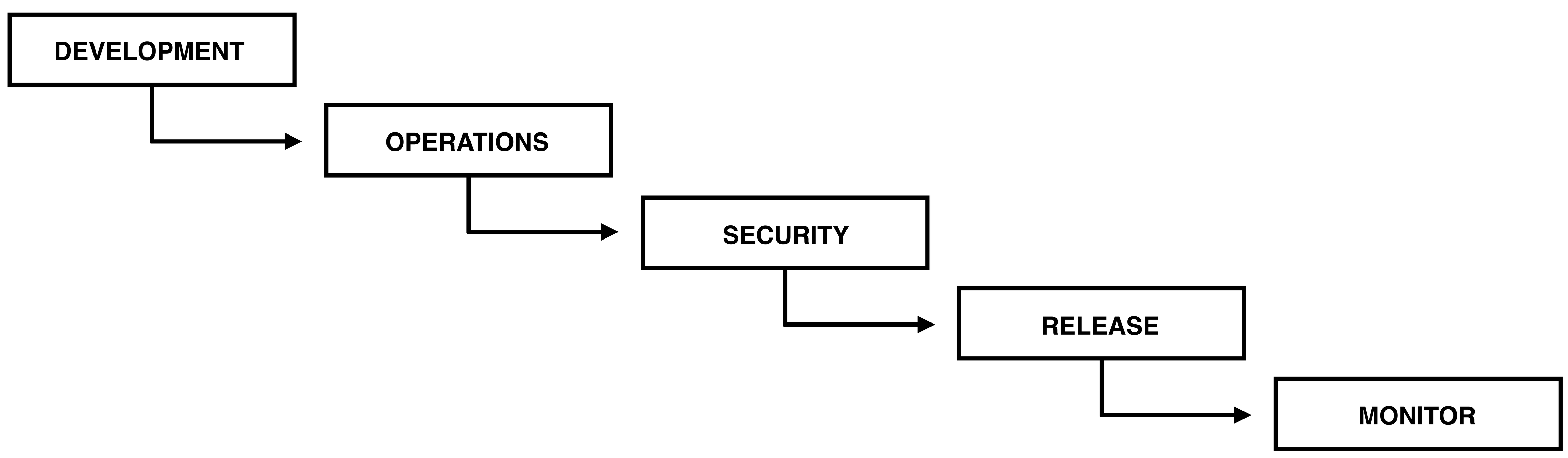
Each of these roles have a different set of knowledge and expertise, and there are many ways in which these teams can work together. The Waterfall methodology is the most common, while the DevOps approach is being broadly adopted.

04

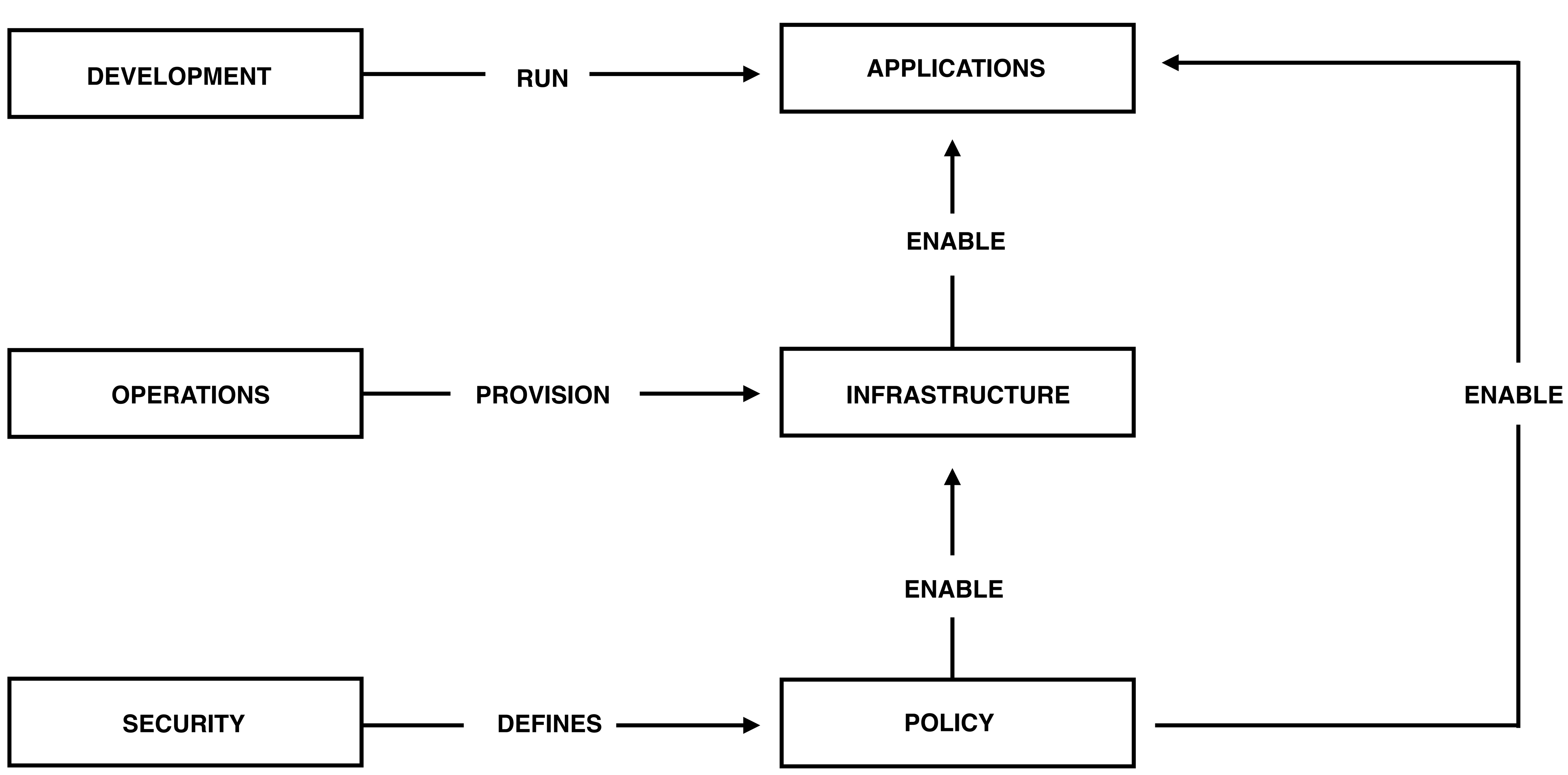
Pipeline View

Waterfall vs DevOps Process

In many traditional software organizations, Waterfall is the dominant model used to deliver applications. This approach prioritizes managing risk, sequentially flowing work between various groups. This tends to be very slow and reflects the challenges of delivering desktop applications that could not be easily updated.

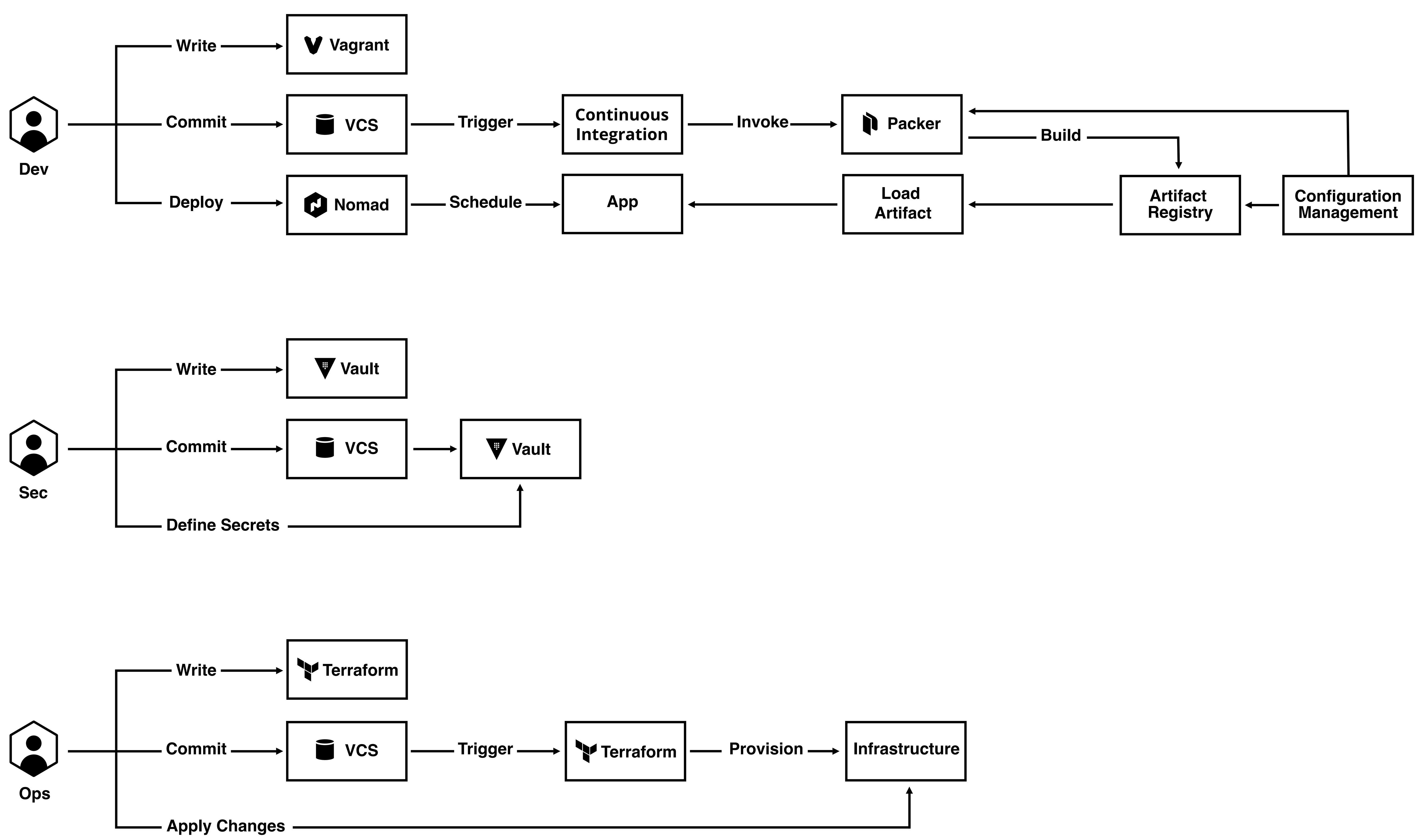


Instead, organizations adopting a DevOps approach create "APIs" between teams, such that the details of each role can be encapsulated, and each group can work independently. This prioritizes agility, allowing things to be done in parallel. This reflects the lower cost of updating software, especially for online applications, which don't need stringent controls over risk.

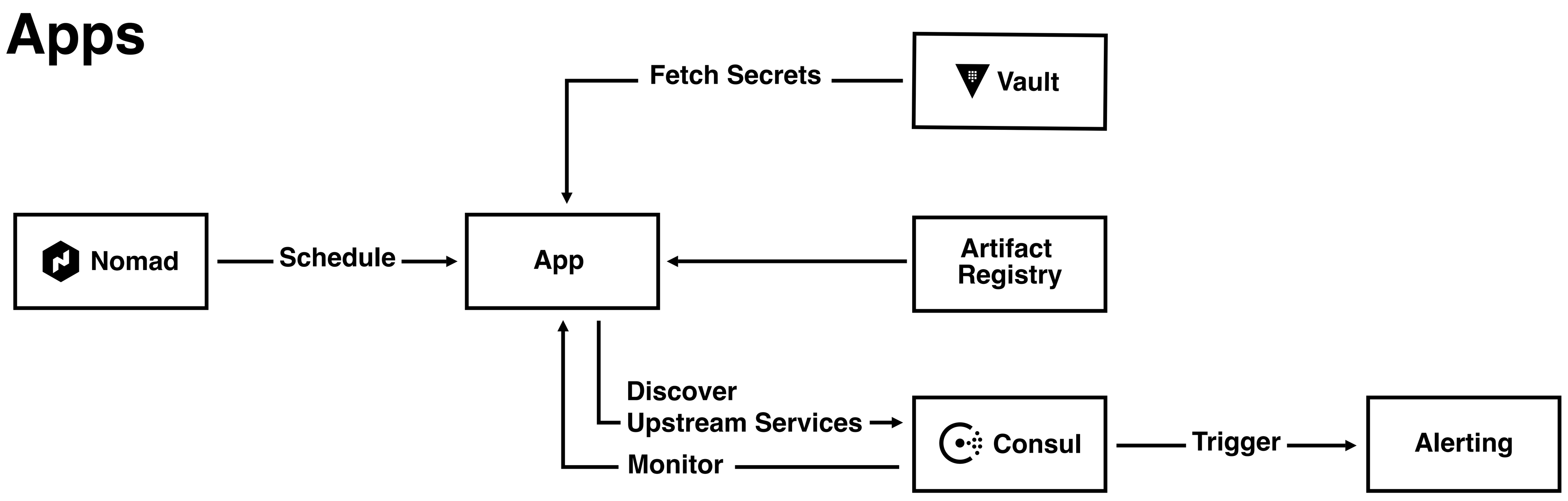


Pipeline View

If we integrate all the people and application delivery steps into a pipeline, we get the following:



Apps



Where HashiCorp Fits

In the pipeline view there are a number of HashiCorp products, each of which serves a specific function:

Vagrant

Vagrant provides a development environment that closely mimics production. This dev/prod parity avoids the "it worked on my machine" class of bugs and allows developers to have a fast feedback loop between development and testing. By codifying the setup of a Vagrant environment new developers can quickly and reliably be on boarded.

Packer

Packer is used to build artifacts, ranging from Docker containers, to AWS AMI's, to VMDK's. It is used to take source code, configuration management, and other provisioning information to build artifacts. These artifacts are usually immutable and versioned.

Terraform

Terraform provides a consistent way to provision and manage resources across hundreds of providers and thousands of resource types. This includes low level storage, compute, and network from cloud service providers, to higher level services like DNS, Content Distribution Networks (CDN). It uses an Infrastructure as Code approach to manage infrastructure and lets operators take a modular and composable approach to managing complex fleets. Terraform Enterprise provides centralized collaboration, coordination, and governance, similar to GitHub for developers.

Nomad

Nomad is a cluster manager and application scheduler. It pools together the resources of many machines and dynamically schedules applications based on declarative job files. Job files give developers an infrastructure as code way to deploy applications abstracted from hardware, while decoupling operators who are managing the underlying fleet.

Consul

Consul provides a toolkit to support service oriented architectures. Applications broadcast their availability and register health checks for monitoring. Applications can discover their upstream services via DNS or by querying Consul with a RESTful HTTP API. Consul provides load balancing and uses health checks to route around failures. Consul provides a Key/Value store which can be used for application configuration and high availability via leader election.

Vault

Vault provides a centralized service for brokering access to credentials and secrets. Security teams can manage policies, delegate access, publish secrets, and audit access. Developers, operators, and applications can access the secret material they need in a secure fashion. Vault also provides key management and cryptographic offload to encrypt PII or other sensitive data. Higher level features include brokering SSH access, dynamic credential generation, and PKI.

Where Other Tooling Fits

Aside from the HashiCorp tools, there are other pieces in the pipeline view where existing tools and technologies are leveraged:

Version Control

Version Control systems come in many different flavors, including Git, Mercurial, SVN, CVS, Perforce and more. These systems provide a versioned history of configuration. Paired with an Infrastructure as Code approach to management, they provide a granular view at how infrastructure has been evolved.

Continuous Integration

There are many common CI systems, including Travis, Jenkins, Hudson, and Bamboo. These systems allow code to be tested, running a battery of unit, integration, and acceptance testing. They also can be used to compile source code and invoke tools like Packer which build production worthy artifacts.

Configuration Management

Configuration management tools like Puppet, Chef, and Ansible provide an Infrastructure as Code approach to setting up machines. These tools are invoked by Packer to configure artifacts and make them production worthy.

Artifact Registry

CI systems and Packer produce production worthy artifacts. These artifacts need to be versioned and stored so that they can be deployed. Feature rich registries like Artifactory can be used, or simpler solutions like an AWS S3 bucket can work depending on the needs of an organization.

Deployment Portal

Often times a deployment portal is used as a wrapper around various systems to shield developers from the underlying complexity. These portals often integrate with CMDB's, artifact registries, and deployment tools like Nomad. They provide self-service ways for developers to deploy new versions, rollback, and scale up/down applications.

Monitoring

Monitoring is a rich space, that includes centralized logging (Splunk, ELK), telemetry (Datadog, InfluxDB), Application Performance Monitoring (NewRelic, AppDynamics), tracing (ZipKin), and more. Applications usually integrate with these systems and stream data to them. These systems are used to diagnose any issues and trigger alerting systems when service level objectives (SLO) are compromised.

Alerting

Alerting systems are usually triggered off monitoring systems when a service level objective (SLO) is compromised and systems like Consul which provide a consistent view of where applications are running and their current health. Services like PagerDuty are used to get developers and/or operators to react and respond.

05

Where to Start

Where to Start

While HashiCorp has many products, getting started is easy because of the modular approach which allows for incremental adoption. We recommend starting with a well scoped project, both to minimize risks and define a clear success criteria. Below are some recommended projects:

Manage Secrets with Vault

Many organizations suffer from secret sprawl, where privileged material like credentials, API tokens, and TLS certificates are stored in many different systems in plaintext including source code, shell scripts, or configuration management. Instead, secrets should be managed in Vault, where they are encrypted in transit and at rest, with central authentication, authorization, and auditing. Organizations can start by standing up a Vault cluster, moving existing secrets, and integrating applications.

Enable Service Discovery with Consul

As micro-services or Service Oriented Architectures (SOA) are adopted, services need the ability to discover and route to their upstreams. For example, web servers needs to communicate with backend API servers. Consul provides a toolkit of features to enable SOA. Organizations can start by standing up a Consul cluster, registering a few services, and using DNS or consul-template to begin integration with downstream services.

Provision Cloud Infrastructure with Terraform

Cloud adoption provides an opportunity to experiment with new tools without changing existing processes. Terraform supports all the major Cloud Service Providers, and there exists rich documentation and examples on using it to provision cloud resources. Organizations can start by provisioning cloud resources for greenfield projects with Terraform. As more comfort is gained, existing applications can be brought under management and the scope of Terraform usage increased.

Build Images with Packer

HashiCorp tools generally push for an immutable model of management, allow it's not a prerequisite. Packer allows for machine images to be easily created across dozens of targets including container and cloud VM images. Building immutable images with Packer is a good starting point to leverage existing configuration management and provisioning tools while adopting a more immutable approach, reducing operational complexity and risk of provisioning time failures.

Run a Container with Nomad

Containers are a convenient way to package applications regardless of language or framework, and provide a standard unit to ship around. Nomad provides a simple way to schedule a single container, all the way up to a million. Organizations can start playing with container schedulers by setting up a Nomad cluster and running Docker containers. This can be extended using Consul for service discovery and load balancing, and Vault for managing and distributing secrets.

Identifying the Value

We have discussed the various challenges in application delivery, along with the principles HashiCorp applies in solving them, but often the value in our approach is implicit. There are several tangible and intangible values, including:

Cloud Adoption

HashiCorp provides a product suite which enables organizations to adopt a single cloud or multiple clouds, both public and private. The workflow-centric view allows the platform specific differences to be accommodated without many cloud-specific workflows.

Infrastructure Automation

Applying an Infrastructure as Code approach across our product suite enables automation of the entire application delivery process. This increases the agility of all teams involved, reduces human errors, and improves security.

Empowering Developer, Operator, and Security Teams

Delineating the application delivery challenge into the sub-problems and using tools instead of tickets to coordinate between teams provides individuals more autonomy via self-service for developers and a decoupling of concerns for operators and security teams.

Technology Flexibility

A workflow-centric approach to application delivery allows heterogeneous technologies to be used easily. This simplifies hybrid cloud adoption, but also enables simpler experimentation and adoption of new tools and services without changing workflows.

Modern Security

Integrating security into each step of the application delivery process and placing zero trust in the network is required for the security challenges of today. HashiCorp Vault provides a security foundation and integrates with other HashiCorp and industry products to provide a wholistic security solution.

Reduced Complexity and Risk

Solving application delivery in a principled way is the difference between working and working well. Empowering individuals to be more effective and organizations to be more productive, avoiding failures and security breaches, and enabling a focus on core competencies.

Simplify Compliance

Adopting a Policy as Code approach can remove compliance as a bottleneck in the application delivery process. Instead of filing tickets and manually enforcing business policies, compliance checks can be codified and automatically enforced. This provides a "sandbox" that developers and operators can work in without needing to wait for a slow approval process.

HashiCorp

06

Conclusion

Conclusion

HashiCorp approaches infrastructure with a principled but pragmatic approach. Software and Infrastructure both evolve to solve new challenges and incorporate the latest best practices. For organizations adopting cloud, HashiCorp provides the products necessary to make the transition from traditional data centers in an incremental way without rewriting applications or adopting complex platforms. For organizations adopting DevOps, HashiCorp provides the products to automate infrastructure and empower developers, operators, and security teams.

Our open source tools are used by millions of users in every geography and industry sector. Our Enterprise products are focused on solving the organizational challenges of the Global 10K, including collaboration, governance, and compliance and customers include many of the Fortune 500 such as Capital One, Comcast, Salesforce, Verizon, and SAP.

HashiCorp