harness

eBook

8 Deployment Pattern Structures to Transform your CI/CD

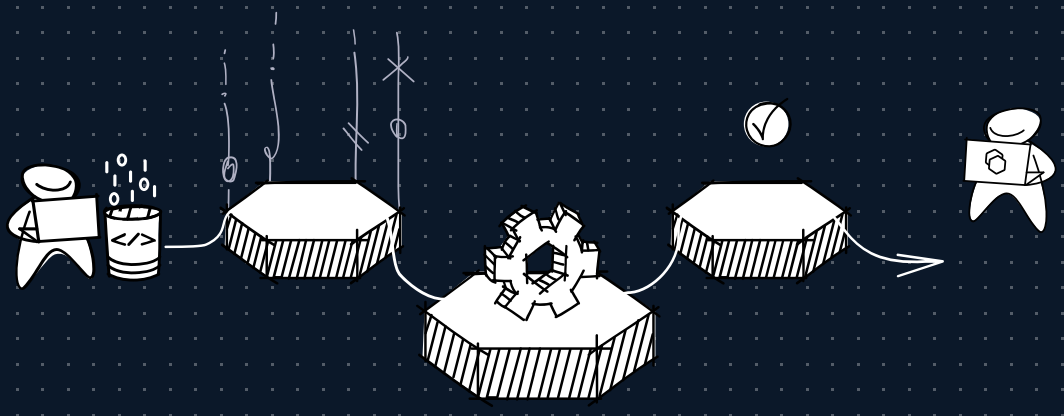
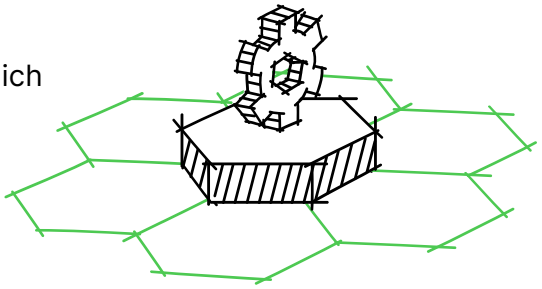


Table of Contents

Introduction

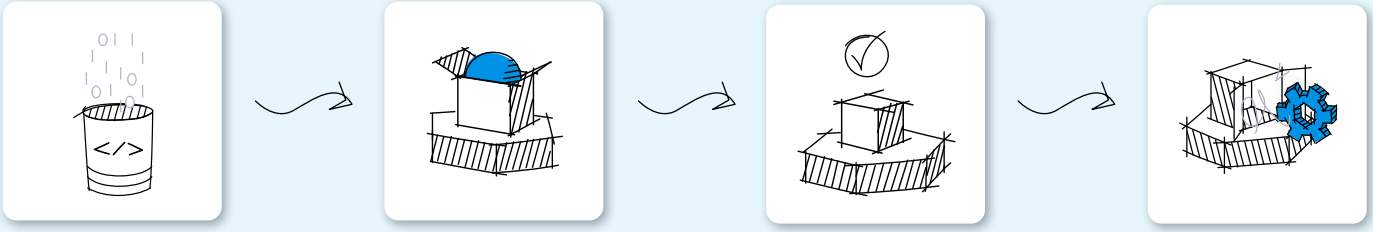
Today, CI/CD deployment patterns are crucial for getting your ideas to production safely and quickly. There is a lot to unpack in your patterns, as they mimic your software development lifecycle (SDLC) and include opinions and governance that are appropriate for your organization and industry.

At Harness, we have the privilege of seeing hundreds of different organizations approach their deployment strategy in distinct ways, which allows us to highlight a few noteworthy trends. In this eBook, we share details of the most common pipeline patterns we see, the process around determining which might be the best fit for your organization, and how they can modernize your DevOps initiatives.



What is a Deployment Pattern?

A deployment pattern is typically made up of one or more steps where source code is turned into a build that is tested and deployed across one or more environments until it eventually reaches the end-user in production.

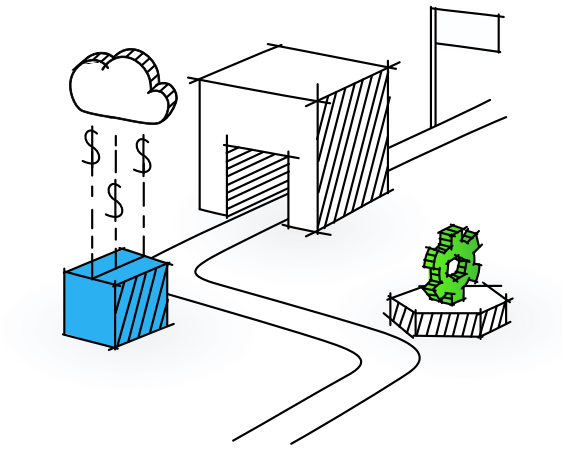


Why use Deployment Patterns?

As systems become more complex, distributed, and modular, our expectation is that they will always perform well. In software delivery, having a safe, fast, and repeatable way of getting your changes into production is critical.

For example, many organizations use traditional shell scripts to deploy their applications. These scripts serve as a simplistic pipeline for deploying a new build or version to a target server or environment.

However, in today's complex, distributed environments, these scripts would need to be modified and executed dozens to hundreds of times over common changes that need constant updating, such as server names, hard-coded passwords or IP addresses, and confidential data. That's not to mention the manual rework that would be required to reconcile those changes.



What's The Difference Between a Good and a Great Deployment Pattern?

Simply put, good deployment patterns mean fast and repeatable pipelines, great patterns mean fast, safe, and repeatable pipelines.

According to the book "Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations," elite performers have a lead time of less than 1 hour, and a change failure rate of less than 15% for production deployments. Therefore, a great pattern will complete in under an hour, and catch 95% of anomalies and regressions, before code reaches an end-user.

If your code takes longer than an hour to reach production, or if more than 2 out of 10 deployments fail, you might want to reconsider your pipeline design and strategy.

The structure of deployment patterns tends to follow what they are most driven by. Ultimately, the driving goal has an impact on the pipeline structure itself. Thus, in order to determine which structures work best for your unique organization, it's important to consider the most common forces that help determine those decisions.

What Are You Optimizing Your Deployments For?

Let's start with the definition of Continuous Delivery from Jez Humble, of the DevOps Research Association (DORA), and coauthor of the book Accelerate:

"Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way."

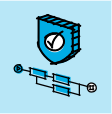
Safely means pipelines enforce quality, testing and a mechanism to deal with failure. Quickly implies pipelines promote code without human intervention, in an automated fashion, and finally, sustainable means pipelines can consistently achieve all this with low effort and resources.

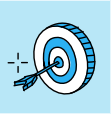
In the real world, it's rare for any customer to build safe, fast and sustainable pipelines. For example, a pipeline that simply picks up an artifact from a repo and deploys it to a Kubernetes cluster can take less than 20-30 seconds.

However, what happens if that artifact has bugs or impacts end users?

Here are some considerations to think about when designing pipelines:

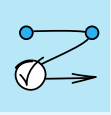

Quality & test coverage/ automation

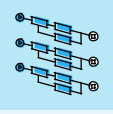

Security (e.g. vulnerability scanning)


Target environment configuration (e.g. ephemeral compute)


Variables and secrets


Release strategy (blue/green/ canary)


Approvals

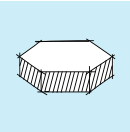

Auditing & compliance


Rollback & failure strategies


Maintaining SLAs, SLOs, SLIs

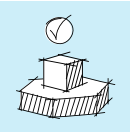
Pipelines can be as unique as organizations; though as our industry marches towards standardization, patterns start to appear.

Driving Forces of Deployment Pattern Selection



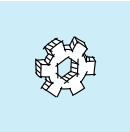
Environments

As systems grow more distributed, the number of locations a service needs to be deployed to increases. If your main goal is to deploy to multiple environments/locations, the pipelines will tend to be more deployment centric favoring the orchestration of all the environments a service has to traverse through.



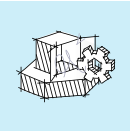
Tests

Test orchestration is a popular use of patterns. If you need to chain together several different testing methodologies, a natural home for the automation progressing the testing is in your deployment pattern. These patterns will tend to have longer stages as progression occurs between environments. The rigor of the testing increases, resulting in a longer time per stage as the pattern gets closer to production.



Services

With the rise of microservices, deployments tend to include more than one service. If the deployment pattern is used for service orchestration, several services in parallel or sequentially need to be deployed. These patterns tend to be used for the orchestration of several services ensuring uniformity across their deployments.



Outcome

Eventually, the feature has to match the expectation. Patterns that focus on outcomes tend to have longer final stages and can continue to run when the deployment is over. An outcome can mean ensuring SLAs/SLOs/SLIs and if breached, the pattern is a conduit to restore/MTTR. Regression or a change in outcome can happen hours or days after a deployment.

People

Before there were deployment patterns, people were highly involved with progressing deployments. If the deployment process is still fairly manual and requires lots of approval gates (e.g. driven by people), these pipelines tend to be long-running and used as a place to keep track of human workflow. Modern patterns try to avoid this practice.

8 Deployment Patterns to Transform your CI/CD

Outcome Centric

Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
High	Medium	Low	Low

Modern workloads and modern infrastructures make it easier to flip the switch to promote changes. With a canary release, changes can safely be introduced throughout environments and introduced to the customer in a safe manner. If you are unfamiliar with a canary release, it introduces changes as a percent of traffic or infrastructure and as changes are vetted continues to promote the changes until the stable version is completely replaced by the canary. With the Outcome Centric Pattern, once the goals are met, the changes are ready to be deployed.

What is Good About This Pattern?

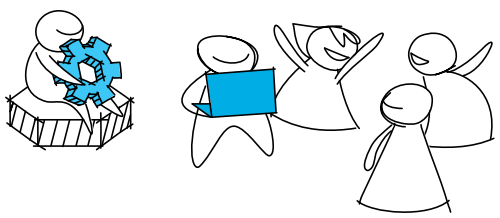
A very pragmatic approach to delivering changes to production, there is less emphasis on what environment to deploy do vs did the changes make sense. Canary releases coupled with more stringent or harder to pass a set of tests like a soak test instill confidence that a change meets expectations.

Where Do You See This Pattern?

This pattern does require teams to have maturity in infrastructure and modern quality engineering. Where the applications are the bread and butter of an organization, such as eCommerce, customers tend to exhibit this pattern. The adage of your customers don't care how you did it [infrastructure] but what you did [application changes] shines bright with eCommerce firms.

Room for Improvement with This Pattern

Depending on who the internal customer is for this pattern, the testing phase (e.g. the main gatekeeper to production) might be viewed as a black box to development teams.



Button Push

Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
Low to Medium	Low	Low to Medium	Low to Medium

There is a duality when embarking on a pipeline journey: either the pipeline is completely automated or completely manual. The Button Push Pattern allows for human approval at the finish line rather than entirely removing the human element. This method allows for one more validation either for compliance, regulatory or confidence building reasons.

What is Good About This Pattern?

Blending human and systemic expertise takes the burden off of the engineering teams to get a release candidate very close to production. Then depending on regulatory, compliance or organizational requirements, a human can review all of the facts and findings that were systematically produced. If you are facing resistance in your pipeline automation journey, the Button Push Pattern would be a good middle ground as automation continues to build confidence.

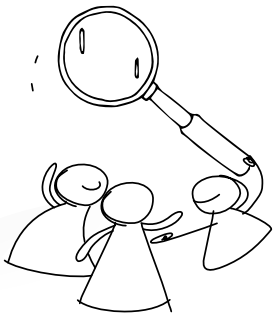
Where Do You See This Pattern?

We see this pattern primarily in organizations where there is some sort of compliance or regulatory requirement for a responsible party to sign off. As organizations try to limit the number of manual approvals but can't fully remove them, the Button Push Pattern is a good choice.

Room for Improvement with This Pattern

Providing all of the information needed to make a decision and keep in an auditable format is challenging for certain CI/CD platforms that do not have out of box capabilities for manual steps. Usually adding a manual step into an automated process is hard. Once the team develops that pattern, expanding the number of manual approvals might be a pitfall or an anti-pattern that might lean towards the Approval Pattern.

If you are facing resistance in your pipeline automation journey, the Button Push Pattern would be a good middle ground as automation continues to build confidence.



Test Automation

Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
Medium	Low	Medium	Low

Test automation can take on a life of its own. Modern test frameworks mimic application development that have orchestration needs and packages to be deployed along with the application itself. Modifying test coverage to cover application changes is an expectation with automated tests. Having a pipeline dedicated to test automation can be a prudent choice.

What is Good About This Pattern?

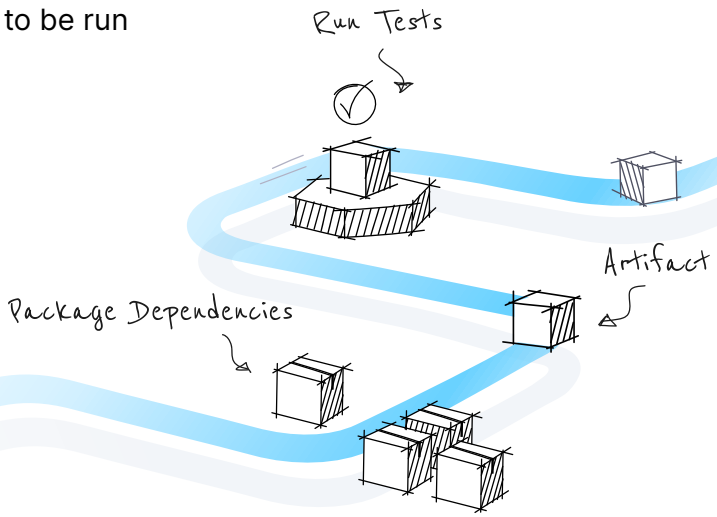
With organizations putting a heavier emphasis on automated testing, getting the testing right requires iteration similar to the application development. The Test Automation Pattern treats testing/quality assurance assets like application assets being able to deploy and run test suites against application artifacts. The pipeline can be run independently so if developers and/or quality assurance engineers want to give feedback early, this is possible with this pattern since is designed to be run multiple times.

Where Do You See This Pattern?

This pattern can potentially be Conway's Law at play. We see this with customers who are building out their test automation practices or tend to have a lot of iteration in their test suites. Several customers representing different verticals had variations of this pattern where test automation was orchestrated by dedicated pipelines.

Room for Improvement with This Pattern

As with any purpose-specific (excluding complete path to production) pipeline, Test Automation represents the ideal steps towards Continuous Delivery. Dependant on the pipeline's designer, placing quality assurance as a separate pipeline for applications traversing through the pattern, may be creating a path of least resistance.



Multiservice

Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
Medium to High	Medium	Medium	Low to Medium

When thinking of a traditional pipeline, the notion of one artifact per execution of a pipeline comes to mind, though there are valid instances where more than one artifact needs to be deployed during a pipeline. For example, deploying multiple services together for a platform change or different types of assets e.g static assets alongside application assets, the Multiservice Pattern's main goal is getting the artifacts deployed.

What is Good About This Pattern?

Multiservice allows the ability to subject multiple services to the same rigor during a release. As enterprise standards permeate the organization and new services are created, a base level of confidence for all deployed services in the pipeline equates to a safer release. As applications and services become more granular, adding additional services to the pipeline is standard to the design of the pattern.

Where Do You See This Pattern?

Organizations that are building platforms e.g have more than one artifact at a time for a release. Previous renditions of the pattern would be executing the same pipeline over and over, just changing the artifact and/or destination for the deployment.

Room for Improvement with This Pattern

Orchestration of multiple service deployments can be complex, especially during a partial failure scenario. Each service should be independent of the others and not tightly coupled.

However, if one service depended on an outcome of another service, there could be a delay in a new feature implementation, as the pipeline needs to be able to have logic or the underlying system needs to determine what has failed vs what has succeeded in the past.

As enterprise standards permeate the organization and new services are created, a base level of confidence for all deployed services in the pipeline equates to a safer release.

Multiservice Environment

Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
Medium	Medium to High	High	Low

The more distributed your applications grow, the more artifacts and environments you will have to orchestrate. Taking into account both of these concerns, the Multiservice Environment Pattern can help drive changes across multiple environments with multiple artifacts.

What is Good About This Pattern?

What was in place before this pattern could be one of the more complex orchestrations ensuring parity between environments and the multiple artifacts that have to be deployed. This pattern grew out of the need to automate a complex deployment scenario. Not every artifact and service is created equally.

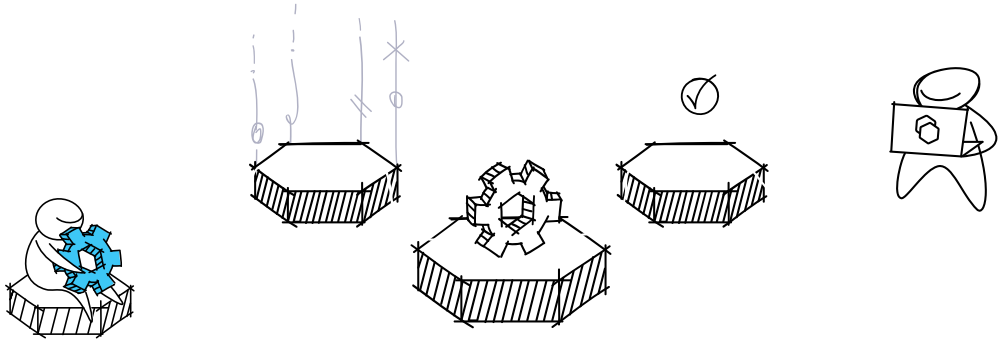
Interacting with different application infrastructure providers, such as a cache or a database, requires a different strategy to deploy and adds to the number of disparate environments/infrastructure. The Multiservice Environment pattern drives both of those concerns.

Where Do You See This Pattern?

This pattern is common among organizations that have complex topologies and are dealing with the significant load to warrant the complex topologies. For example, eCommerce and Travel and Leisure providers dealing with web-scale traffic on a constant basis.

Room for Improvement with This Pattern

This pattern more often than not is broken up into separate pipelines encompassing pre-prod and then prod. When crossing potentially multiple skill domains in a pipeline (e.g. application and cache changes), the full effect of all the changes needs to be vetted out. By design this could be a soak test or what was observed was production-specific pipelines once lower environments were deemed fit.



Semi Approval

Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
Low to Medium	Medium to High	Medium	Low

Sometimes it is hard to remove the human element completely from your pipeline. With the Semi Approval pattern, there is a good mix of automation and human approval (when necessary). A mix of the new and traditional approaches are represented.

What is Good About This Pattern?

For organizations under regulatory/compliance obligations or who are risk-averse, this is a step towards full automation. Automating even pieces of the promotional steps might represent a fundamental shift in the organization. Being pragmatic on where to tackle automation first, automated promotions/approvals will happen in lower environments. This pattern can be a catalyst for change.

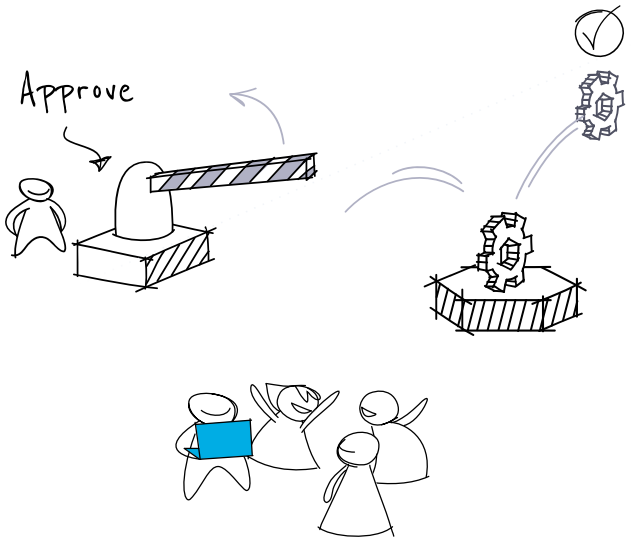
> Automating even pieces of the promotional steps might represent a fundamental shift in the organization.

Where do You See This Pattern?

In financial services firms and regulated industries, these patterns exist to address the drive for innovation and digital transformation. Organizations that also have high-risk changes can run a subset of their applications through the pattern.

Room for Improvement with This Pattern

If there are too many approval gates or there is a lag with an approval gate, a bottleneck will occur. The Full Approval Pattern takes the Semi Approval Pattern and adds a manual approval to every stage. As a more evolutionary pattern, improvements will be made as confidence in the pipeline continues and the need for manual approvals lessens.



Full Approval Pattern

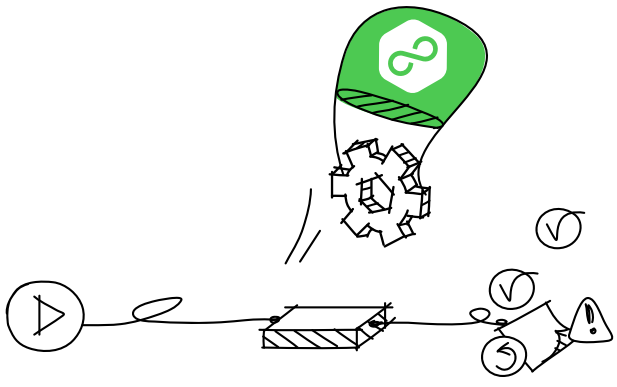
Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
Low	High	Medium	Low

Remember when you played Telephone as a kid? You would pass a secret message from person to person, and at the end of the line, the phrase would be infuenced by all of the people it passed through. That could be a pattern in your pipeline. With the Full Approval Pattern, most if not all promotion steps require a manual approval.

What is Good About This Pattern?

For certain organizations that are conservative and risk-averse, the Full Approval Pattern mimics the current SDLC. Pushback to adopt this pattern would be low since most likely was happening before a pipeline in disjointed steps in several platforms. Having this pattern would help track where bottlenecks do occur and if there was some sort of future evolution to occur, the areas for which improvements could be made would be very apparent.



Where do You See This Pattern?

This pattern most often supports highly regulated industries such as critical infra-structure, intelligence/military, and sensitive financial and healthcare organizations. When there is a need to bring together disparate workforces e.g multiple contracting or consulting organizations to deliver a project, confidence automation might not be readily available until future renditions of the project.

Room for Improvement with This Pattern

More often than not, the manual approval steps are signs of manual testing. Especially if this is your first foray into getting an item into production, automation will come as learnings about all of the steps that a production deployment takes is flushed out. This lift and shift of your existing manual process being tracked by a workflow or pipeline is the first step into automation. After the steps are all complete, the pipeline is complete. Ownership of what happens after deployment follows the organizational structure.

GitOps

Pattern Score

Agility:	Administration:	Complexity of Pattern:	Risk of Change Failure:
High	Medium	Medium	Medium

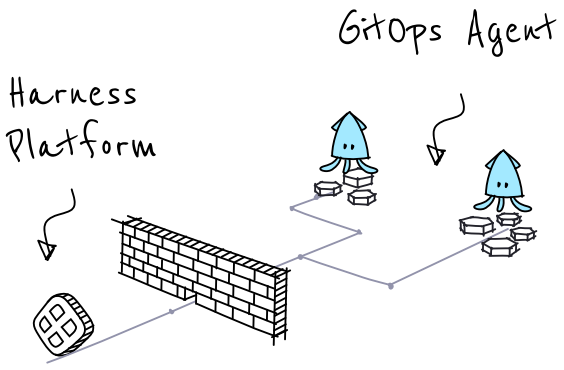
In a GitOps model, the system's desired configuration is stored in a version control system, such as Git. Instead of making system changes via a UI or CLI, an engineer makes changes to the configuration files that represent the desired state using Git as a single source of truth. A difference between the desired state stored in Git and the system's actual state indicates that not all changes have been deployed. These changes can be reviewed and approved through standard version control mechanisms such as pull requests, code reviews, and merges to master. When changes have been approved and merged to the main branch, an operator software process is responsible for changing the system's current state to the desired state based on the configuration stored in Git.

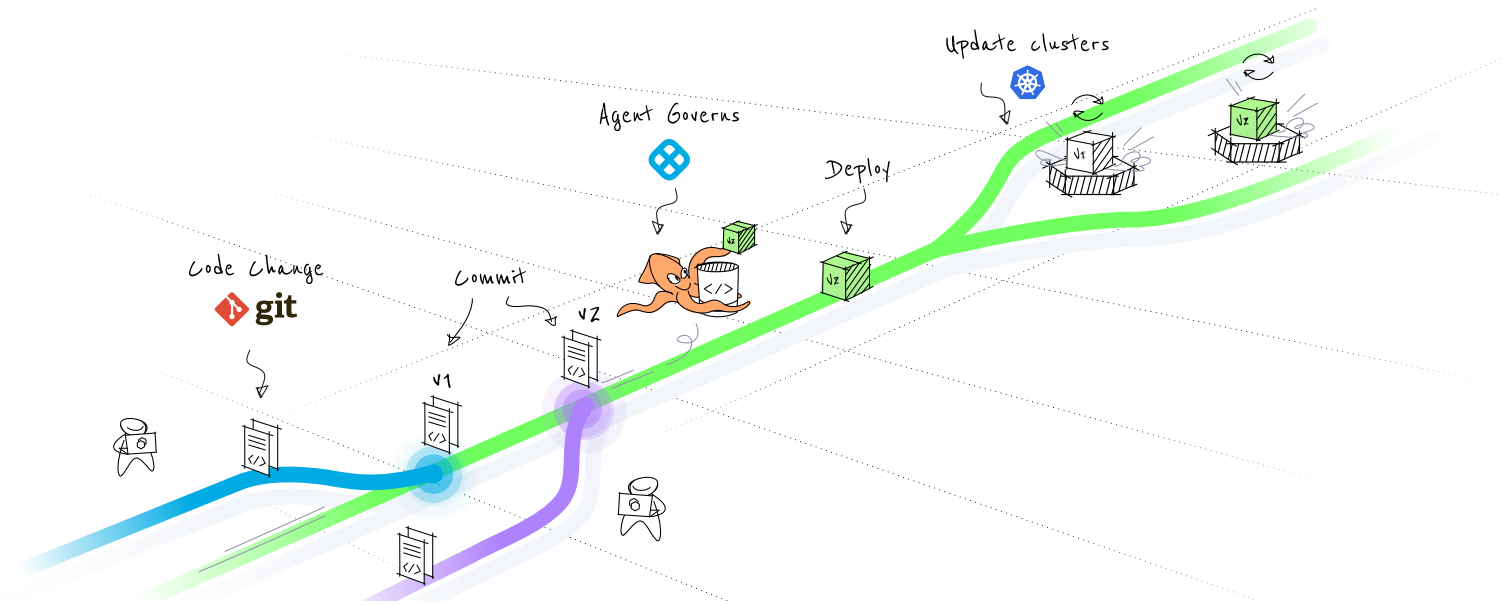
GitOps is the fastest pattern for developers to deploy code changes to environments.

What is Good About This Pattern?

GitOps is the fastest pattern for developers to deploy code changes to environments. Developers favor GitOps because it gives them the freedom to deploy software without having to interface with a traditional CD UI and they can avoid extremely bureaucratic deployment pipelines. Most developers are familiar with Git, configuration-as-code, and infrastructure as code meaning they aren't expected to learn an entirely new CD system.

Developers favor GitOps because it gives them the freedom to deploy software without having to interface with a traditional CD UI and they can avoid extremely bureaucratic deployment pipelines.





Where Do You See This Pattern?

This pattern is ideal for non-production environments with high rates of change and low chances of failure, although these are not strict limitations. GitOps is often used for testing environments or for applications that aren't business critical.

Room for Improvement with This Pattern

GitOps covers a subset of the software lifecycle mainly focused on infrastructure automation. This is important because GitOps tools are sometimes marketed as the one-size-fits-all solution that will solve all release problems, and this is simply not true. First of all, GitOps requires that your deployment artifacts are already there. This means that tasks such as "Compiling code, running unit/integration tests, security scanning and static analysis" ...are not a concern of GitOps tools and are assumed to already be in place.

GitOps also doesn't address promotion of releases between environments. GitOps uses a pull request approval model where developers make changes, create PRs, then an approver may accept the PR. Generally this works great until we need to include business approvals. Business personnel may not be familiar with Git or PRs. If a team needs business approvals, they have to build a process over GitOps. This drastically increases lead time. GitOps is dependent on a pure PR - approval based model - in which the PR approver is responsible for any reviews. After a PR is approved there is no way to enforce enterprise policy rules. Enterprises need ways to control what end-users can do on the cluster and ways to ensure the clusters are in compliance with company policies. These policies may be there to meet governance and legal requirements or to enforce best practices and organizational conventions.

Conclusion

At Harness, we have the opportunity to see how hundreds of organizations implement software delivery pipelines. Transcending organizations, the patterns covered in this ebook serve as approaches to getting ideas into production, and they continue to evolve, balancing agility and regulatory constraints. No one pattern can rule them all since each organization and industry vertical they represent are different. Because pipelines bring together disparate skills and applications are the culmination of the work of many teams, pipelines can be highly unique and specific to the organization.

Teams and organizations can have variations or aggregations of multiple patterns. Fine-tuning the processes around manual approvals and full automation, these patterns support different levels of agility, confidence, and opinion. Though your specific goal might not 100% match one of the patterns we've covered, these models can act as a foundation for pipeline development in your organization.

harness

About Harness

Harness, the Modern Software Delivery Platform™, provides a simple, safe and secure way for engineering and DevOps teams to rapidly release applications into production. Harness uses machine learning to detect the quality of deployments and automatically roll back failed ones, saving time and reducing the need for custom scripting and manual oversight, giving engineers their weekends back. Harness Inc. is based in San Francisco.

Learn more at harness.io.