STATE OF GITOPS REPORT

Learn how hundreds of practitioners apply GitOps concepts in the real world

2025

Octopus Deploy

Codefresh

Contents

Executive summary

The State of GitOps report investigates the adoption, practices, and benefits of a GitOps approach. Our research is based on data from 660 survey responses and interviews with a panel of experts and practitioners.

The goal of our research is to understand what good GitOps looks like. To do this, we explored different adoption styles, analyzed whether GitOps delivered the expected benefits, and found specific techniques and practices contributing to success.

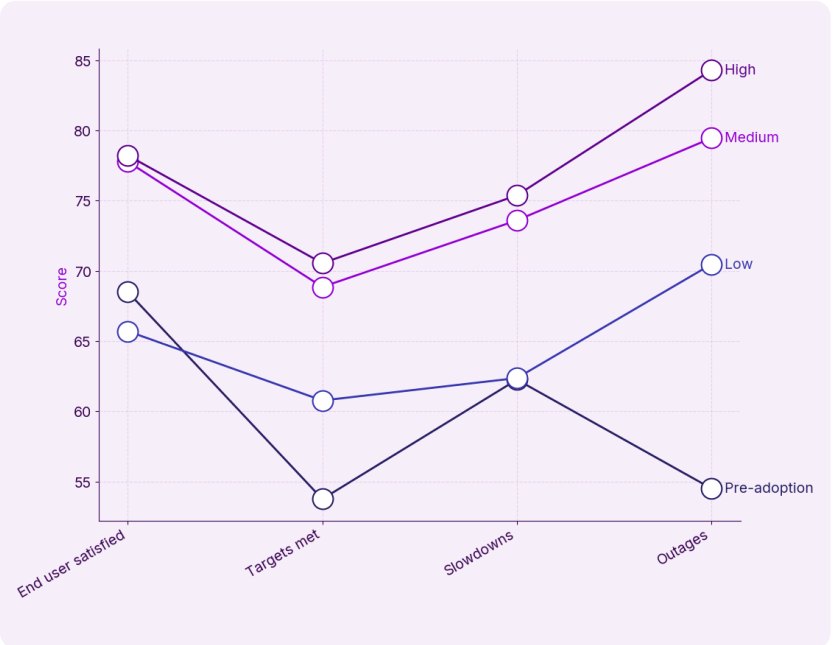
Our survey gathered samples from GitOps practitioners and technical professionals across multiple industries and geographical locations. We collected data on the level of GitOps adoption, the specific techniques and practices in use, and their effectiveness across several outcome perspectives. In addition to our analysis, we interviewed GitOps experts to provide their insights and commentary on the findings.

This is the first report to look at GitOps in depth, and we're excited to share what we've learned.

1. Better software delivery

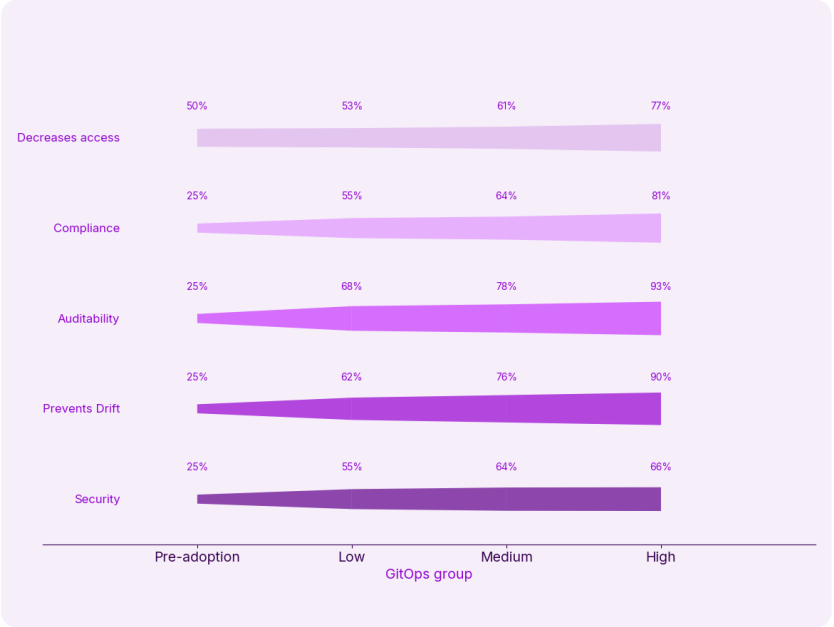
High-performing GitOps teams were more likely to exhibit high software delivery performance as measured by the DORA 4 key metrics, change failure rate, time to restore, deployment frequency, and lead time for changes.

2. Increased reliability



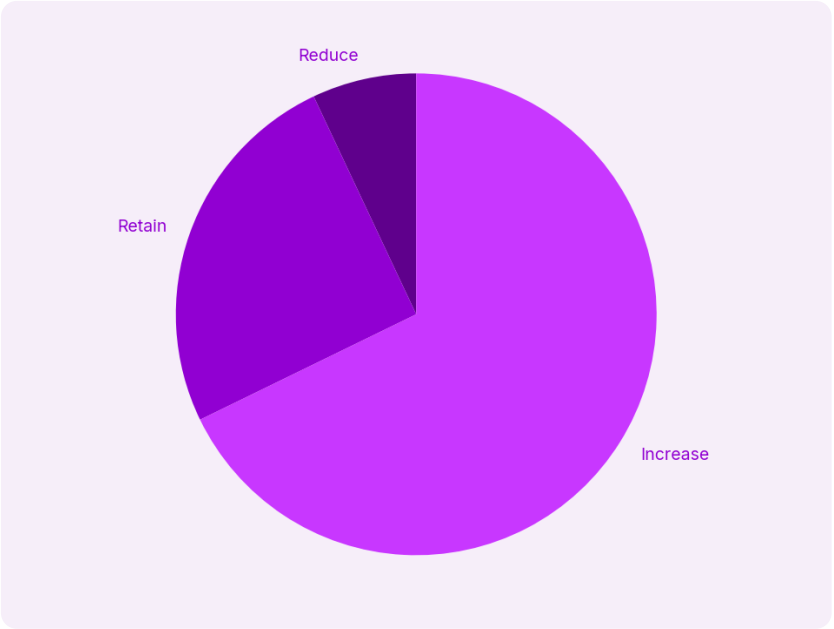
High-performing GitOps teams also had the best reliability record, as measured by user satisfaction, meeting uptime targets, and avoiding slowdowns and outages.

3. Security and compliance



We tested 5 direct GitOps security and compliance benefits. We found a clear link between GitOps maturity and benefit attainment. Teams who adopt the most GitOps practices were most likely to receive these benefits.

4. Adoption is increasing



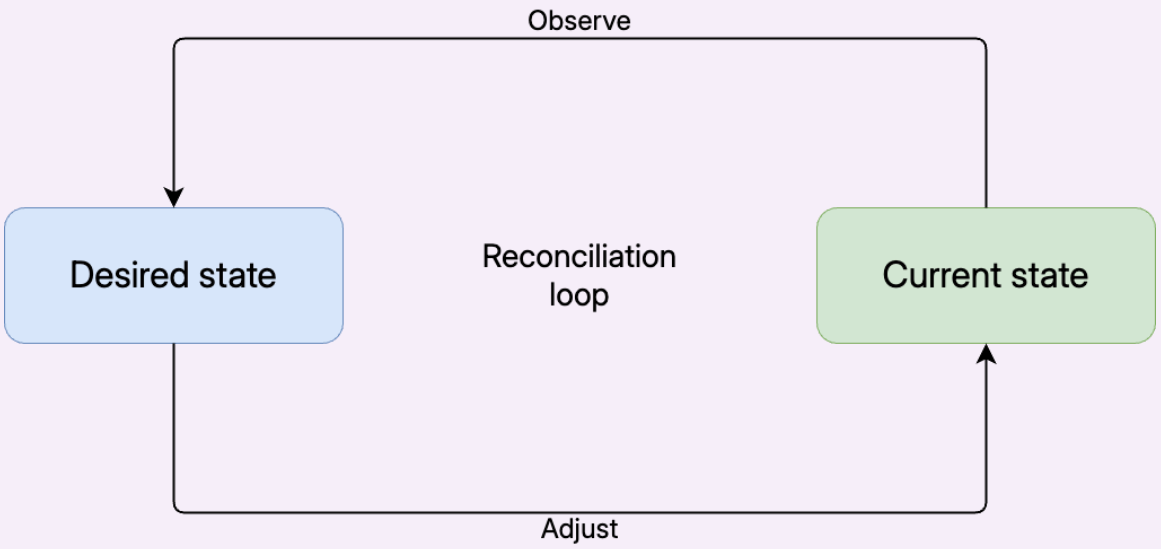
Most organizations (93%) plan to continue or increase their GitOps adoption. Organizations that have yet to fully adopt GitOps practices are more likely to reduce or stop their rollout.

What is GitOps?

To understand the findings, it's helpful to understand how we built our definition of GitOps and its practices.

GitOps draws on ideas from DevOps and infrastructure as code (IaC). By storing definitions of infrastructure and applications in version control, teams can use familiar tools to make, review, roll back, and audit those changes. Just like DevOps, there are cultural and technical elements to GitOps. Teams agree to flow all changes through declarative state files and decide on the collaboration around changes. Once a team makes a change, high levels of automation ensure they are applied.

At the core of GitOps is a reconciliation loop that observes the system's current state, compares it to the desired state, and makes adjustments to bring the current state into alignment.



The GitOps reconciliation loop

GitOps principles

The **OpenGitOps project**[4] defines 4 GitOps principles in version 1.0.0.

1. Declarative
A system managed by GitOps must have its desired state expressed declaratively.

2. Versioned and Immutable
Desired state is stored in a way that enforces immutability, versioning, and retains a complete version history.

3. Pulled Automatically
Software agents automatically pull the desired state declarations from the source.

4. Continuously Reconciled
Software agents continuously observe the actual system state and attempt to apply the desired state.

You can **learn more about GitOps on the Codefresh website**[5].

The State of GitOps report

We created the State of GitOps report to learn how practitioners apply GitOps concepts in the real world. Understanding different levels and styles of adoption and how outcomes are affected was central to the research. We wanted to test the validity of principles and practices like those found in OpenGitOps to build a model that describes what *good GitOps* looks like.

Our pre-survey interviews with practitioners surfaced a common collection of GitOps benefits, and it's essential to understand whether those benefits are likely and under what conditions.

Most importantly, we're curious about how everything works in software delivery, and existing research doesn't cover GitOps practices.

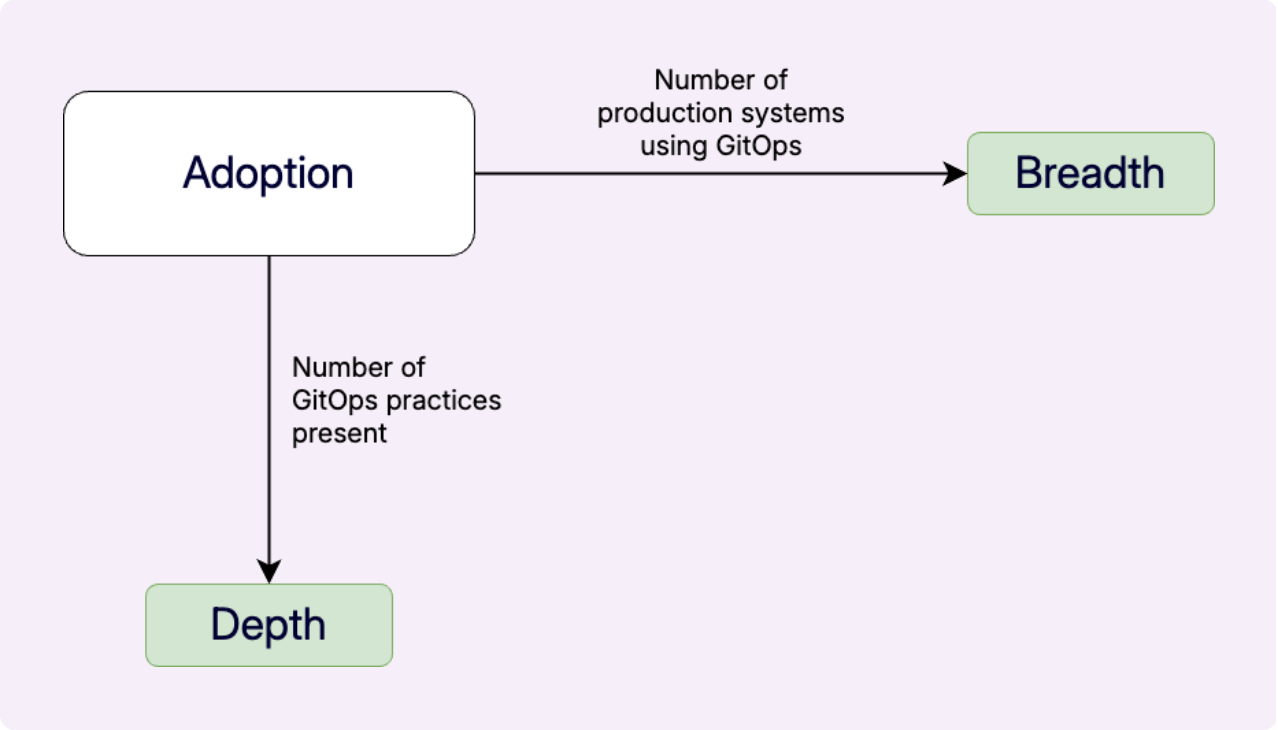
The State of GitOps report is a chance to baseline the practice and position it alongside the other capabilities teams need to create great software and sustain their efforts over the long term.

GitOps adoption

To truly understand the extent to which GitOps has been put to use, we needed to go beyond the number of organizations using it. We know interest in GitOps is high as organizations consider using it to solve common problems around security, consistency, and operating at scale. Many organizations will tentatively experiment with GitOps, while others have rolled it out broadly.

To capture the difference between tentative and embedded GitOps, we assessed the breadth of adoption within each organization. This highlights the journey from evaluation to full rollout. Almost one-third of organizations use GitOps for less than 20% of their production systems.

Later in the report, we cover the depth of adoption by looking at specific GitOps practices. Combining breadth and depth of adoption shows how many organizations move past proof of concept to deeply embedded GitOps practice.



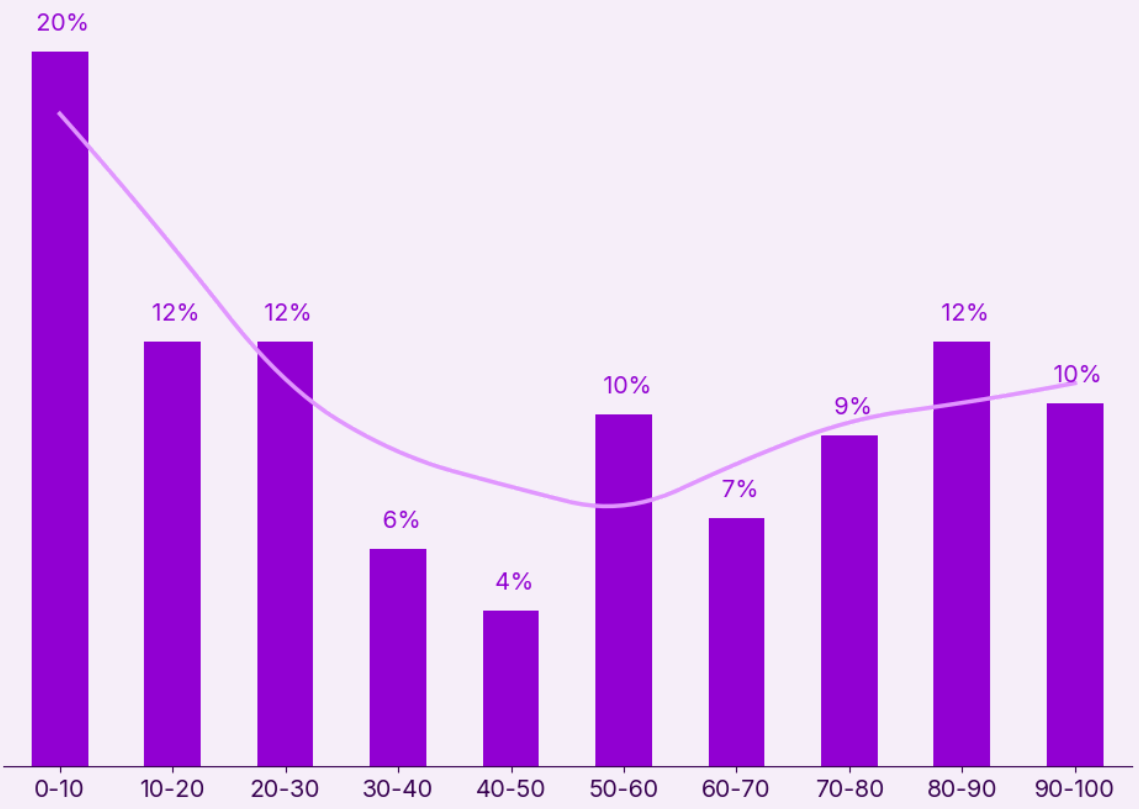
Breadth

The extent of adoption across production systems within an organization, indicating whether GitOps is an early experiment or an embedded way of working.

Depth

How many practices have been implemented, indicating the level of maturity with GitOps.

Production use

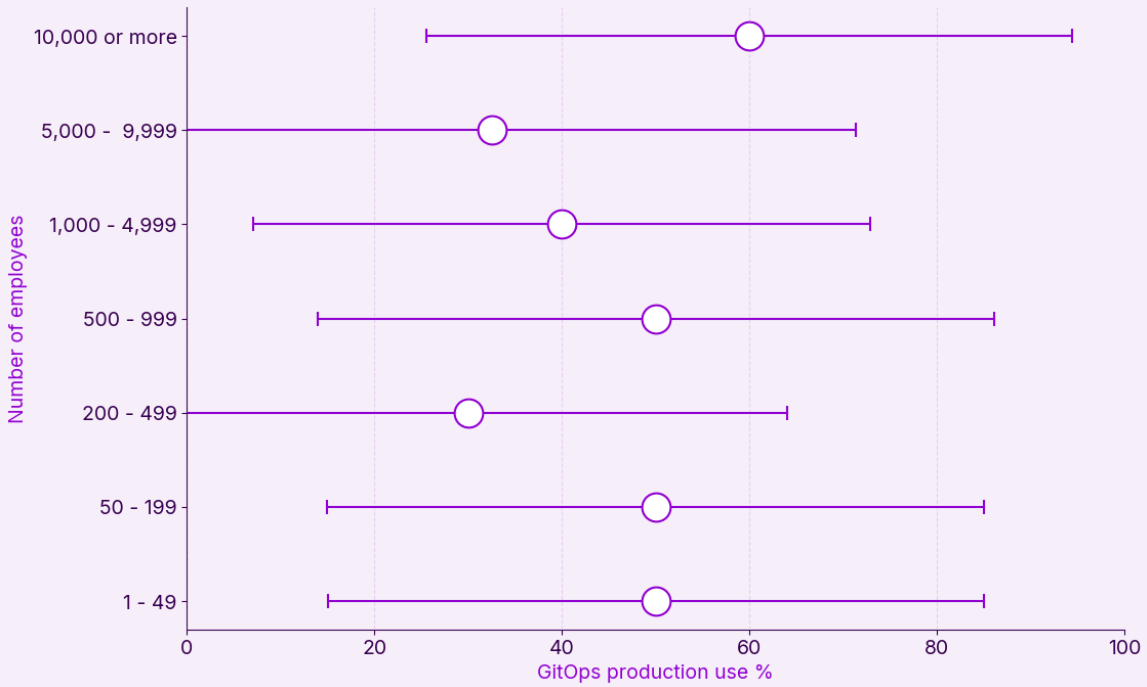


Organizations shown by percentage of production use (bottom axis)

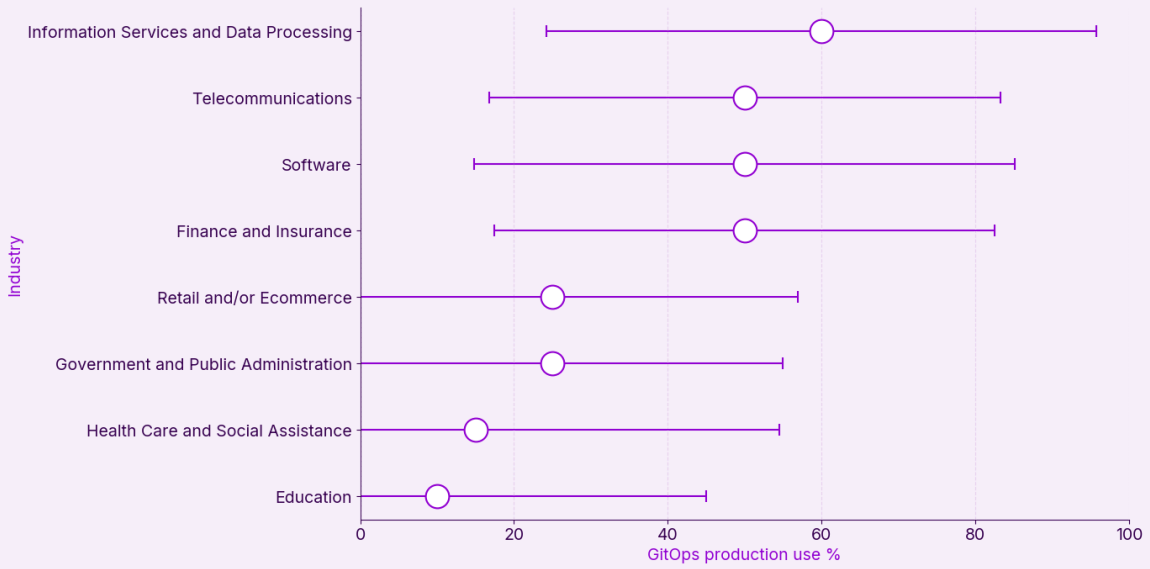
Looking at GitOps adoption as a percentage of production applications indicates how deeply embedded the practice is within an organization. Some organizations use GitOps for practically all production systems, while others are just starting their journey.

There is an interesting dip at the lower-middle range of 30-50%. This could indicate that organizations get stuck below the 30% mark or that they can accelerate past the middle. As 93% of organizations plan on continuing or increasing GitOps adoption, this hop likely indicates that once you have a working pilot of GitOps you can roll it out quickly to other applications and services with a similar profile.

We looked into whether adoption varied by organization size and industry. Each segment had high variability, with industry being a slightly better predictor of adoption. This likely indicates existing industry preferences for being early or late adopters of new technology.

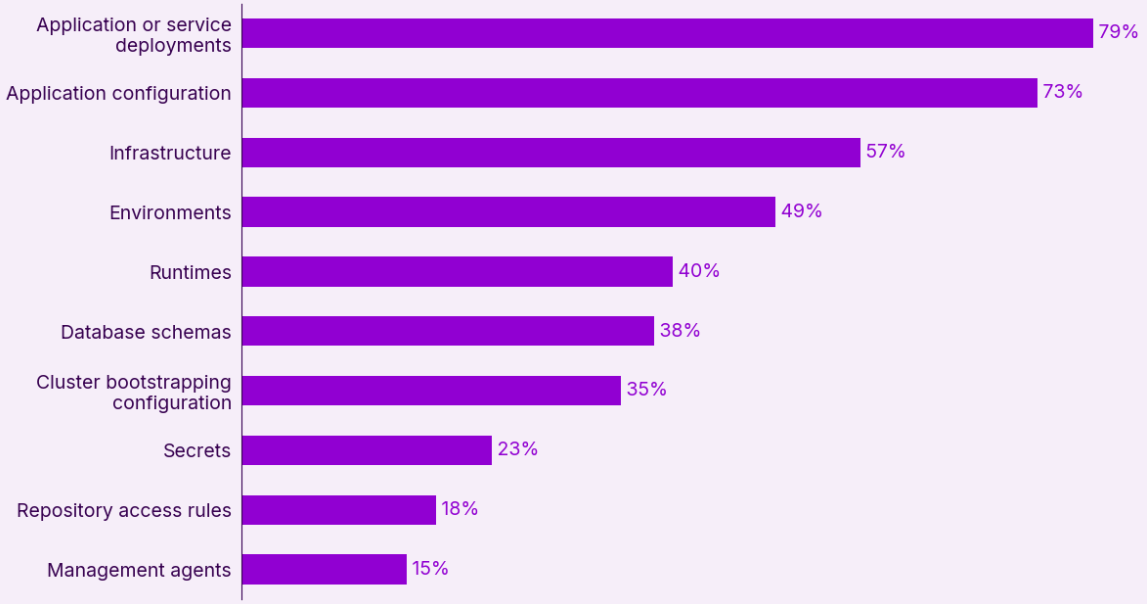


The % of production systems using GitOps by organization size



GitOps production use by industry (top and bottom 4)

What GitOps is used for



***Question**: What parts of the application's configuration are stored in version-controlled files?*

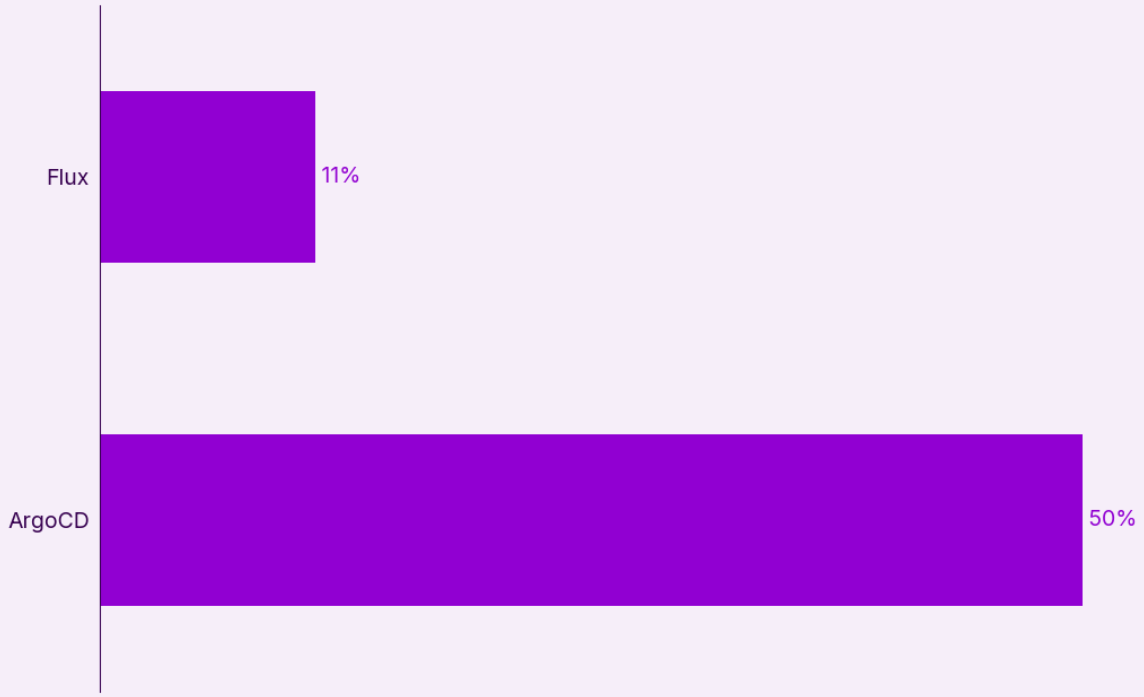
Organizations most often use GitOps for application or service deployments (79%), application configurations (73%), and infrastructure (57%). At the lower end of the scale, secrets (23%), repository access rules (18%), and management agent configurations (15%) are the least common uses.

There has been widespread discussion of storing secrets in version control using techniques like **sealed secrets**[1]. In terms of adoption, this is not the preferred way to handle secrets. Continuous Delivery tools and dedicated vaults provide a secure way to handle secrets and make it easy to update them in one place.

> Although GitOps is strongly associated with Kubernetes, 26% of organizations are applying GitOps to technology stacks that don't include it. This challenges the popular belief that GitOps is only for Kubernetes.

In GitOps, more is more. More of the practices, for more of the uses, for more of your production workload gets the best results. You can read more about this effect in the GitOps benefits section.

GitOps tools

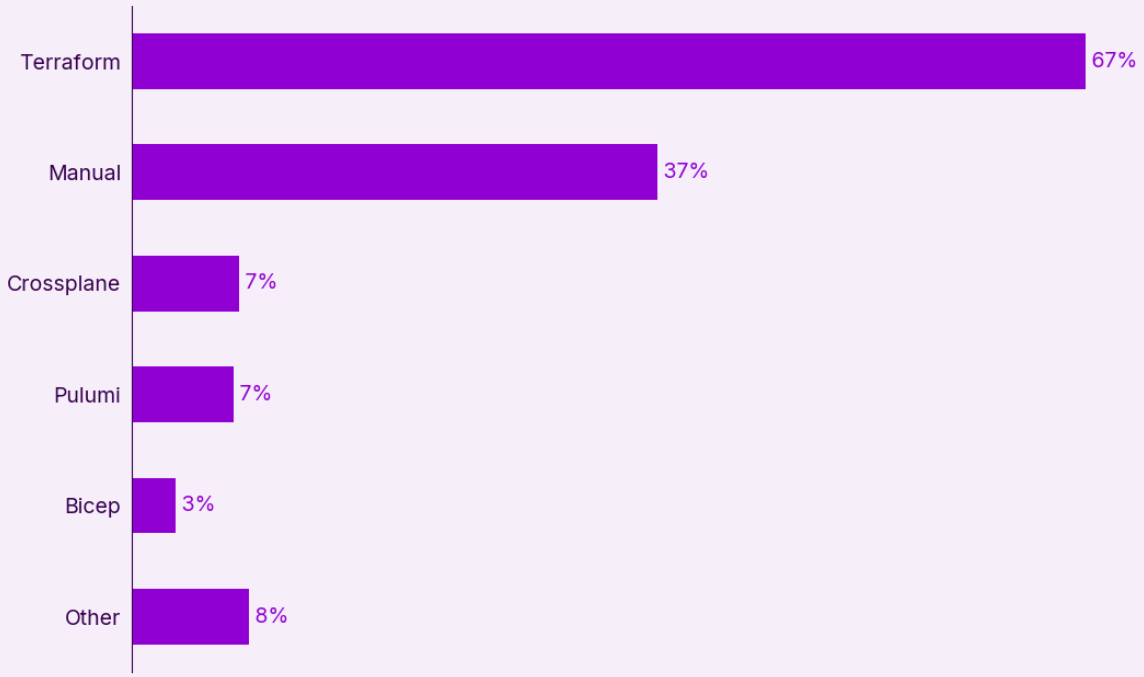


Adoption of open-source GitOps tools

We collected the tools organizations are using as part of their GitOps adoption. Respondents could also add tools not available in our initial list. The tools were a mix of open-source and commercial products, including dedicated GitOps tools and related tools for platforms, orchestration, and pipelines. From this list we extracted all open-source GitOps tools to compare their adoption.

ArgoCD (50%) is the most popular open-source GitOps tool, with second place going to Flux (11%).

Infrastructure automation tools



***Question**: How do you create infrastructure?*

We asked which tools people were using for infrastructure automation. We allowed multiple selections. Most organizations (76%) use a single tool, while 20% used 2 tools and 4% used between 3 and 5 tools. We assessed GitOps adoption independently of specific tools and the presence of a tool doesn't necessarily mean GitOps principles have been applied.

For infrastructure automation, Terraform has a clear lead over everything else (67%). The main competitor to Terraform is manual infrastructure management (37%). After Crossplane (8%), Pulumi (7%), and Bicep (3%), no other tool had more than 3% adoption. In future reports, we hope to show the trend of adoption for these tools. It's also worth noting that tools become available at different times. For example, the first public release of Terraform was published on **June 30, 2015**[3] while for Crossplane it was **December 4, 2018**[2].

A lack of infrastructure automation will likely be a problem when working at scale, potentially preventing a whole category of architectural choices due to the prohibitive cost of management. Even at a small scale, automation can prevent configuration drift and ease the setup of additional resources.

GitOps maturity

The **OpenGitOps project**[4] provides a broadly agreed definition of GitOps. It uses technology-agnostic language to define 4 principles you can use to understand the concept.

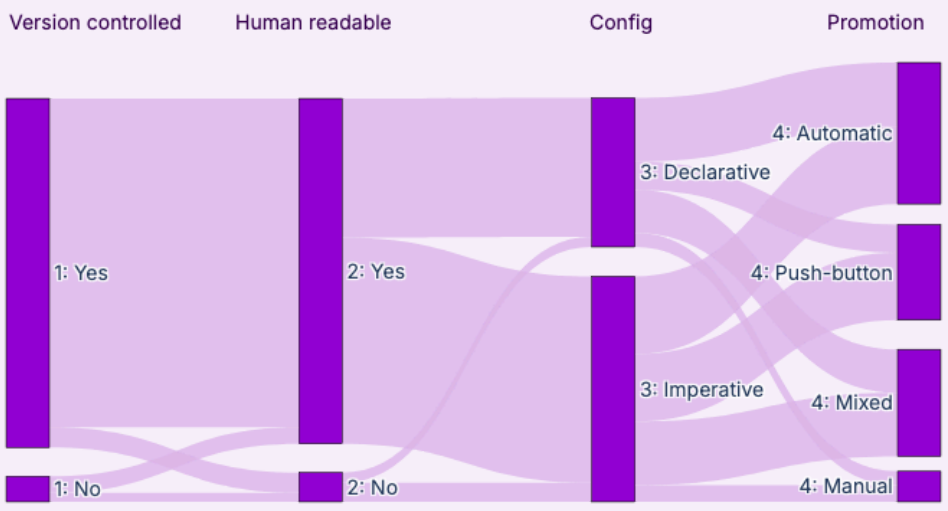
While many organizations might say they are using GitOps, each implementation would look different. To better understand this, we asked questions to determine how closely organizations aligned with our understanding of the techniques required for GitOps to work.

Practice does vary, with a strong pattern emerging for partial adoptions that use version-controlled and human-readable configuration files without the automatic pull and continuous reconciliation capabilities. As organizations start their journey with GitOps, it's understandable that their focus starts with creating their configuration files in version control. This is a pragmatic starting point and prerequisite for the more advanced aspects of GitOps.

As we encouraged responses from people at all stages of the adoption journey, we expected various adoption stories, including from organizations that don't claim to be doing GitOps.

> We use Sankey diagrams to visualize the flow of responses between different GitOps practices. The height of the bars indicate the proportion of responses giving each answer, and the links show how responses map to the next practice.

GitOps practice adoption



The GitOps practice landscape

The GitOps practice landscape shows the volume of answers for 4 questions relating to GitOps practices. The journey along the top of this map is considered the most consistent with GitOps principles. Organizations use version control and human readable formats in most cases, but other practices are more fragmented.

We examine each step between the practices below.

Version controlled and human readable

93% use version control and 92% use human readable formats

Storing configuration in version control is an intrinsic part of the GitOps definition. Without it, we wouldn't expect to find an organization claiming to practice GitOps.

Human-readable formats make it easier to manage changes and approvals, review change history, and understand the system's desired state. To be human-readable, configuration is defined in concise text files using a format such as JSON or YAML. Some formats, such as Hashicorp Configuration Language (HCL), introduce more complex elements that aren't present in serialization formats, but this would still be described as human-readable.

There is high adoption for storing application configuration files in version control (93%) and human-readable formats (92%). The mapping from version control to human-readable formats is less than 100%, which suggests a small proportion of organizations are using a non-readable format for GitOps.

Non-readable formats might include generated configuration files created by capturing the current state. These may use less readable formats or be considered unreadable due to large file sizes. It may also include files created by tools that use binary formats that depend on the tool's user interface to view.

Human readable and declarative

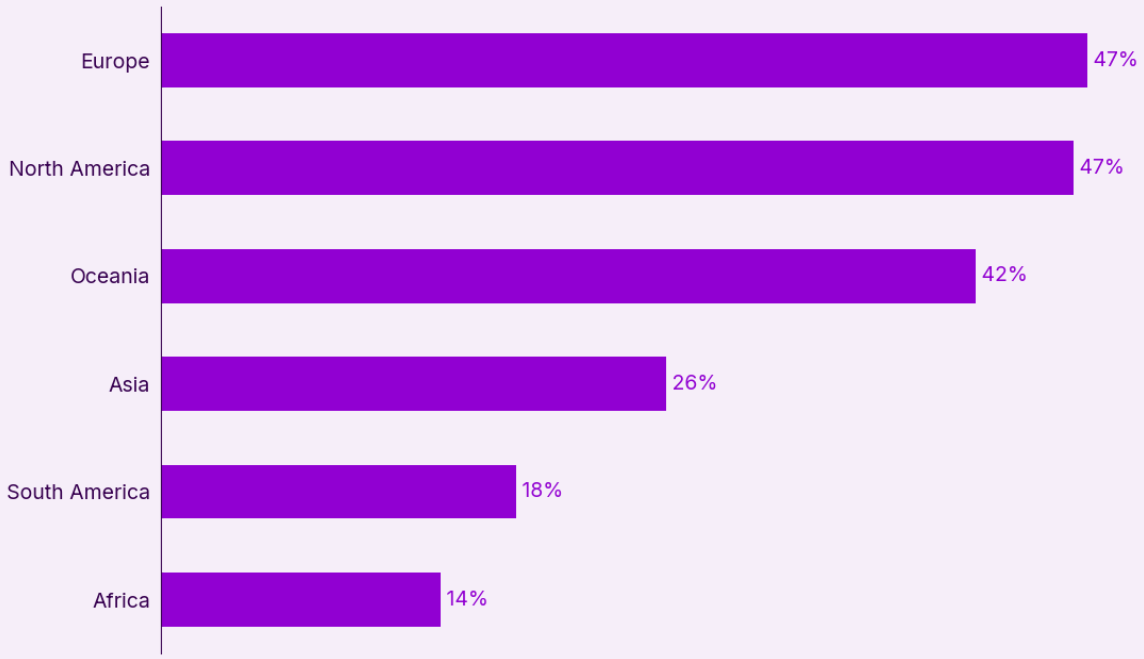
40% use declarative configuration

While the OpenGitOps principles state that configuration should be declarative, more organizations (60%) use imperative configuration. Imperative configuration describes a sequence of steps that must be performed to bring the system into the correct state. In contrast, declarative configuration specifies the target state, not the steps to reach it.

We found that in many cases, declarative tools were being used as part of imperative orchestrations. For example, you might call Terraform (declarative) from a pipeline that executes steps that call tools, scripts, and APIs in sequence (imperative).

Traditional checklists may be an influence on a preference for imperative configuration as they typically detail a sequence of steps. It's possible organizations are blocked by infrastructure complexity or aren't ready to trust tooling to manage the gap between actual and desired state. This is an area we would like to explore further.

This is one of the largest deviations from textbook GitOps we've found in the report. There were large differences between regions, with the proportion of declarative configuration having a high of 47% and a low of 14%.



Regional adoption of declarative configuration

Declarative and automatically applied

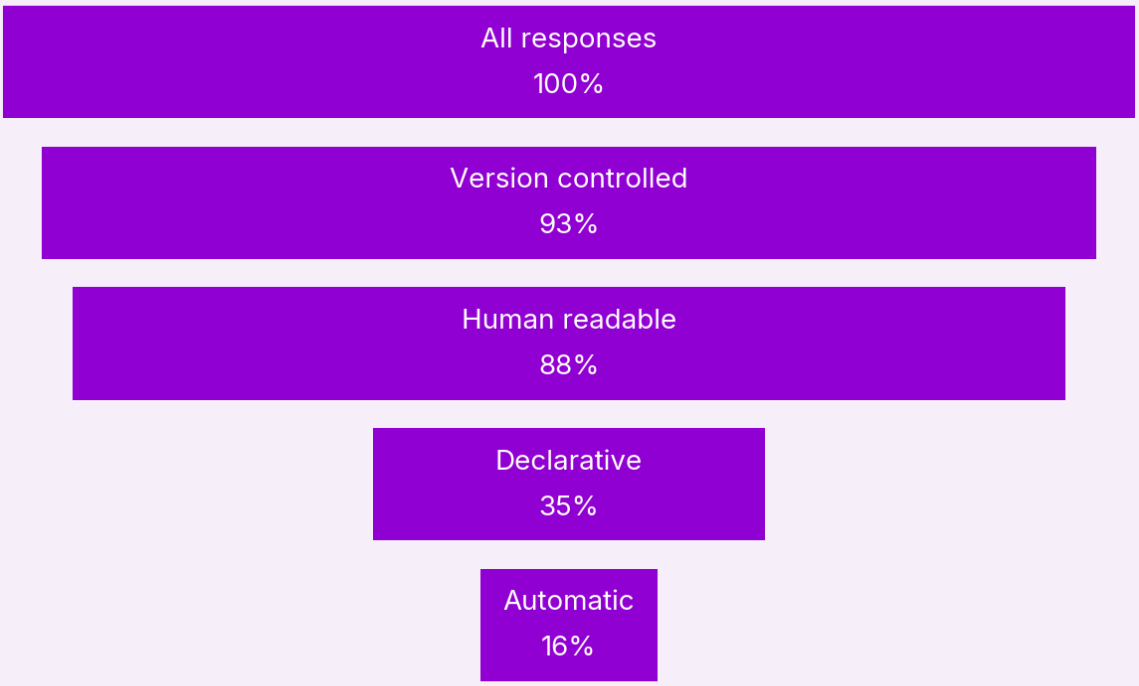
38% use automatic promotion

The promotion of GitOps changes is another area that differs from what we expected. The reconciliation loop at the heart of GitOps should have the version-controlled definition pulled regularly, with the desired state being applied automatically. While the largest group of responses followed this pattern (38%), the majority used a different approach, such as manually applying the changes made in version control (25%), making changes entirely manually (8%), or a mix of approaches (29%).

Our set of questions about how changes are reconciled indicates the presence of tools that work as an inverse alternative to GitOps where the changes are made to infrastructure and synced back to version control. While this isn't GitOps, it creates a similar audit trail and history of changes.

A click-first versioned configuration will be unlikely to bring all of the GitOps benefits, such as security and reduced access. These changes would be reviewed after the fact, which may increase risk and reduce reliability.

GitOps adoption funnel



GitOps adoption funnel

The GitOps adoption funnel provides a view of practice adoption by ranking them in order of popularity and disqualifying responses when they diverge from the strict definition of GitOps. This helps us understand where practice deviates from our assumed GitOps definition based on OpenGitOps principles.

Each stage of the funnel requires the current and all previous criteria to be met. For example, an organization with version control, human readable formats, and automatic reconciliation but not using declarative configuration would be represented only in the first three funnel stages as they are eliminated in the fourth stage.

The funnel shows high adoption rates for version-controlled and human-readable configurations. It then splits in half for declarative configuration and again for automatic reconciliation.

The table compares all respondents with practice adoption against those who proceed through the funnel. This allows a comparison between organizations using the practice and those using the practice alongside preceding practices.

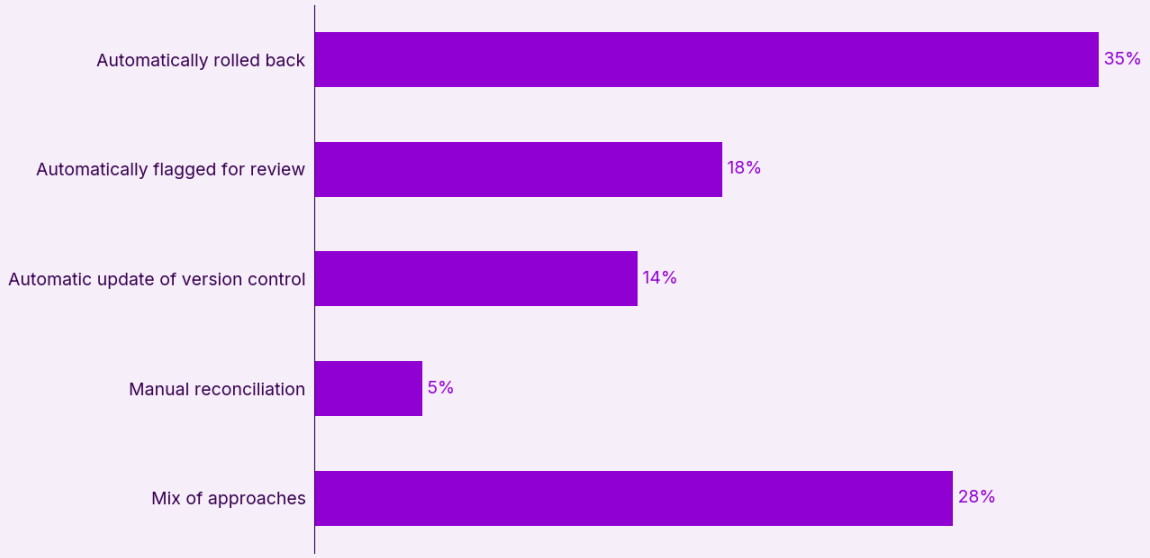
Stage	All responses	Funnel survivors
Version controlled	93%	93%
Human readable	92%	88%
Declarative	40%	35%
Automatic	38%	16%

Funnel survivors are shown as a % of all respondents.

Having a defined target state is a necessary precondition of automatic reconciliation. Declarative configuration, even if it's controlled by imperative orchestration, is the best known way to prevent configuration drift. Our working theory is that organizations with imperative configuration who achieve continuous reconciliation are using at least one declarative tool within their process.

Our findings indicate that missing GitOps practices prevent organizations from achieving the expected benefits of the approach. We expand on the impact of partial GitOps adoption in the GitOps benefits section of the report.

Reconciliation approaches



***Question**: If a change was manually applied to the running application, what would happen?*

When application state is modified manually, the GitOps approach is to roll back the manual change automatically. While this is the most common approach (35%), flagging changes for manual review is another popular way to handle the differences (18%).

Organizations may make temporary changes using ClickOps (manually changing settings using remote access, ad-hoc commands, or via management portals), such as increasing a replica count during an incident. They would want to retain this manual change until the incident is over. This likely reflects a weakness in their cycle time for GitOps changes, as it should be possible to perform the change using the standard GitOps approach with a follow-up change after the incident to return the number of replicas to the original value.

A bonus of using GitOps is that the immediate change *and* the rollback change can be prepared at the same time, minimizing the chance of forgetting to revert to the desired state after the incident. The immediate change can be approved, while the rollback is held back until the appropriate time.

It may be convenient to make a manual change and use the reconciliation process to flag the synchronization issue, as the reviewer can reject it to revert the change later. One problem with this approach is it leaves gaps in the history of changes in version control. It would also require another process to capture how an incident was handled.

As we learned from the field of database change management, the best way to build great habits is to use the blunt force of automated reconciliation. Developers who are used to changing the development environment database to add tables or columns find it hard to break the habit. Overwriting manual changes makes it easier to do the right thing and make the changes via version control. In the same way, teams using GitOps should enable automatic reconciliation.

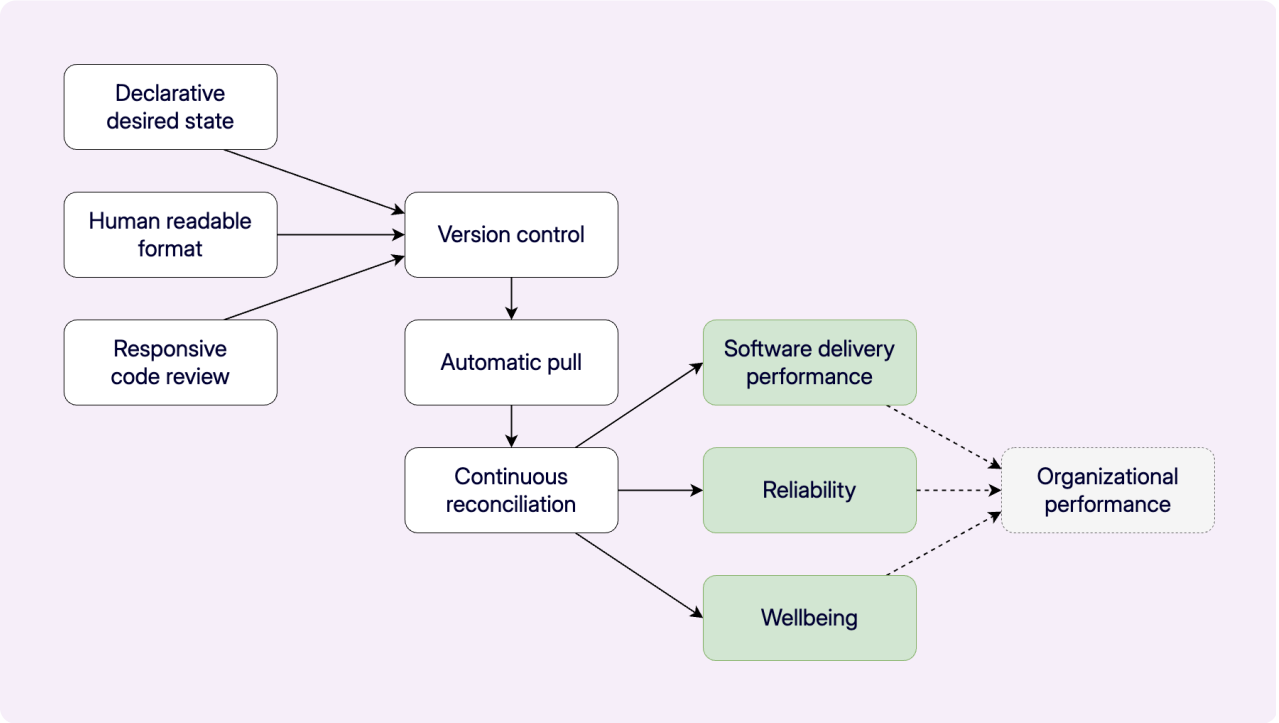
While some infrastructure teams may be concerned about changes being automatically reconciled, the benefits of returning the system to the correct state and eliminating configuration drift should outweigh these concerns. A period of running the reconciliation process to flag drift, allowing the infrastructure team to see what would be automatically corrected, can demonstrate these benefits.

DevOps first

Organizations that have been on a DevOps journey will have development and operations teams that share goals and collaborate closely. Without this, it's common for infrastructure teams to block the installation of reconciliation agents, which limits GitOps performance.

Equally, organizations with low deployment frequency often form slow code review habits. When GitOps code reviews face long waits and multiple approval steps, changes will likely be made outside the process for urgent changes and incident response.

The GitOps model



The GitOps model sets out the set of 6 practices that we found to be necessary for successful GitOps adoption. We tested these practices against established measurement systems for software delivery performance, reliability, and wellbeing. We selected these outcomes because the **DORA State of DevOps research**[6] has found they predict successful outcomes for the organization. Additionally, we tested the practices against the expected benefits of a GitOps approach.

The layout of the 6 practices reflects the logical ordering of these practices in relation to each other.

1. Declarative desired state

Declarative configuration describes the desired state without specifying the steps and sequence required to reach it. This keeps the target condition separate from the implementation details, such as API calls, commands, or scripts. It also removes problems caused by assumptions on the start state that may result in the end state not matching what is needed.

2. Human readable format

A human-readable format makes it easy to review a configuration change and understand how it will change the system's state. Ideally, it is also concise and has a strong signal-to-noise ratio. The YAML format is the most common example of a human-readable GitOps format.

3. Responsive code review

When configuration changes require a code review, this should take place as close to real time as possible. Slow code review delays changes and increases the chances of ClickOps changes being made for time-sensitive updates.

4. Version control

Storing desired state configuration in version control ensures it is versioned, immutable, and has a history of all changes. This helps diagnose issues and provides an easy rollback mechanism when required.

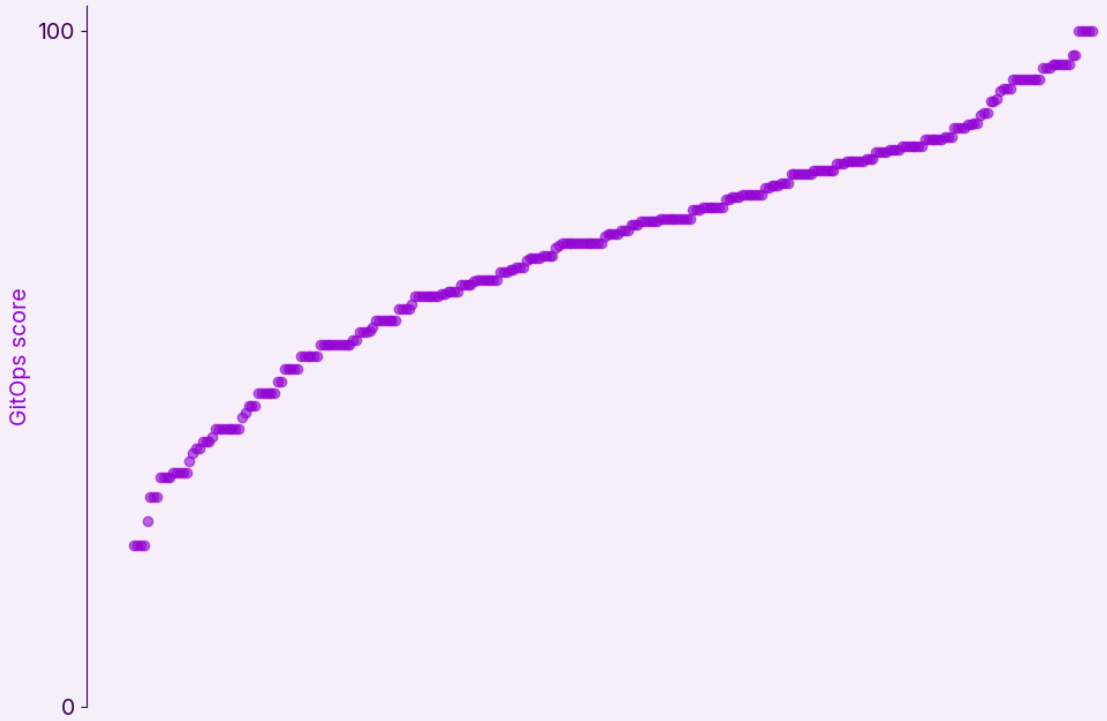
5. Automatic pull

A software agent should automatically pull configuration from version control. This means the configuration doesn't need to be aware of where it must be applied, either in terms of location or the number of destinations.

6. Continuous reconciliation

The configuration defined in version control reflects the system's intended state, so this state should be checked and brought into line automatically. This is usually done by a software agent.

GitOps scores



Scores achieved by individual responses

The model also presents an opportunity to create a GitOps score based on the 6 practices. A score is calculated for each practice and represented as an aggregated GitOps score normalized to a 100-point scale. For example, automatically rolling back manual configuration changes receives more points than detecting them and flagging them for review.

Responses achieved scores in the range of 23 to 100. We used this score to compare GitOps performance with similar scoring scales for outcomes.

We don't want to use GitOps scores to be judgemental. They reflect how closely a particular adoption meets the definition of GitOps described in the model. You know your context better than us. It's for you to decide whether changes or improvements are needed. GitOps scores can help you assess your current state while the model provides improvement ideas.

Policies within your organization may limit your score. If you feel policies are a limiting factor, you can work on building the trust required to influence them over time.

GitOps benefits

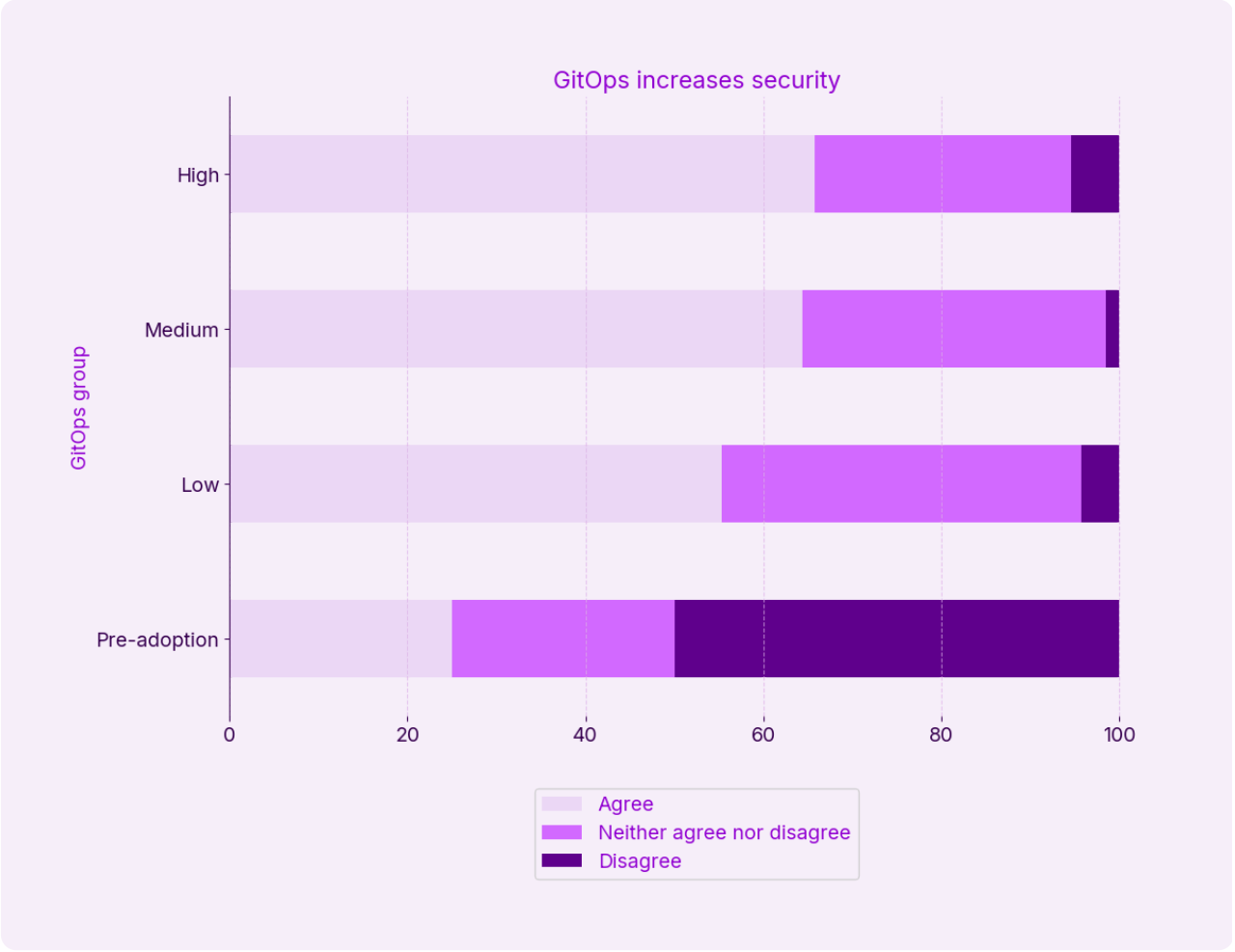
We looked at 5 potential benefits of GitOps. This approach may offer further benefits, which we are likely to explore in the future. While many organizations report finding these benefits, we found an interesting link between GitOps scores and benefit attainment.

By grouping responses by score, we found those with higher scores were more likely to report they were getting the benefits. For example, those scoring 75-100 were 2.7x more likely to say GitOps increased security than those scoring 0-25.

Organizations with higher GitOps scores are more likely to obtain the benefits of GitOps. Organizations that are missing practices from the GitOps model are less likely to find benefits accrue.

GitOps score	Group	Description
75-100	High performers	Have adopted most or all GitOps practices to a high degree
50-75	Medium performers	Have adopted several GitOps practices, some to a high degree
25-50	Low performers	They have adopted some GitOps practices to a lesser extent
0-25	Pre-adoption	Not yet using GitOps or early in adoption

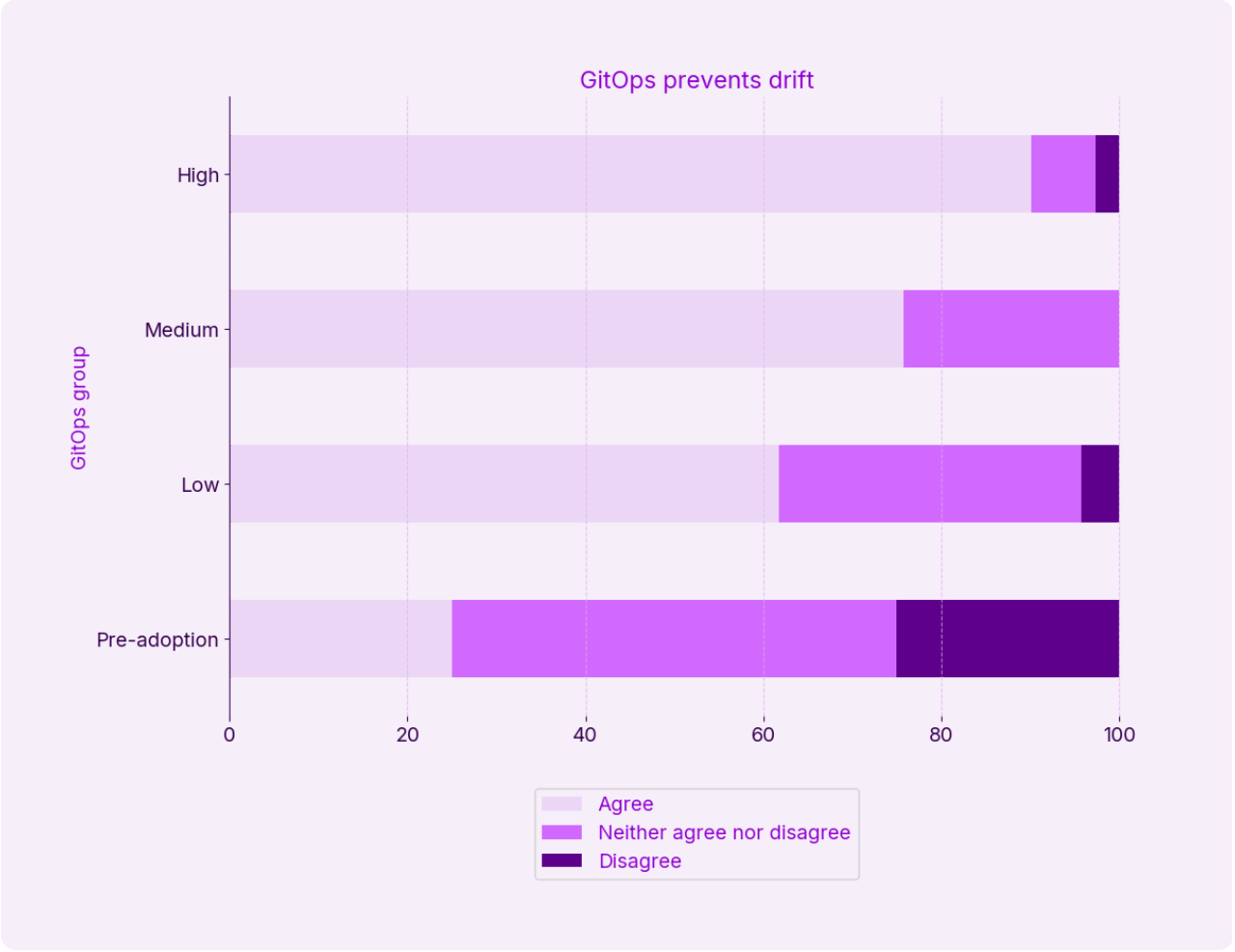
GitOps increases security



Overall, 63% of respondents agree that GitOps increases security. For high performers, this rises to 66%.

Using **version control** tools with **responsive code review** means all changes are audited and checked as part of the process. **Automatic pull** eliminates the need to expose an access endpoint for configuration changes, which removes this attack vector. These 3 practices should result in increased security.

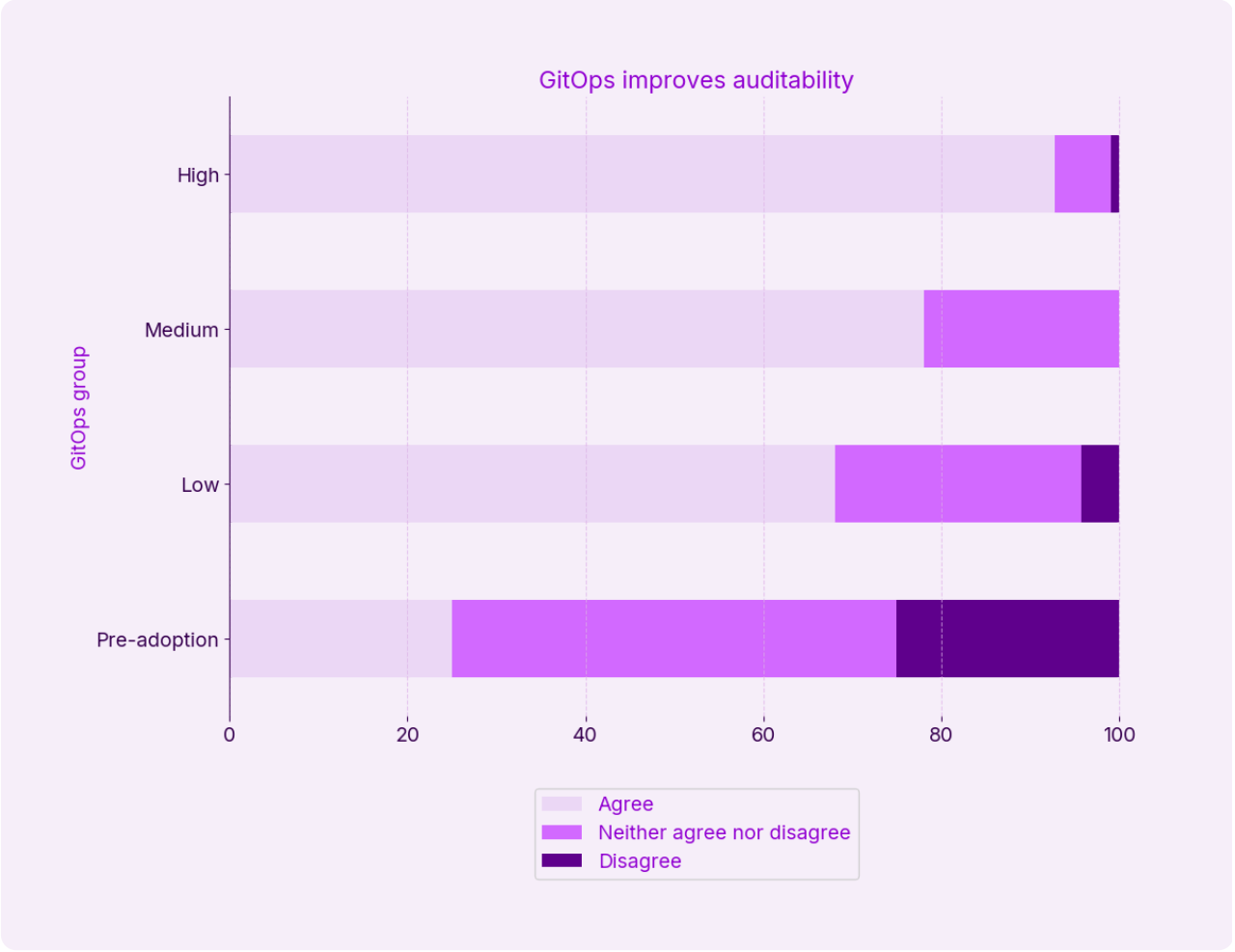
GitOps prevents configuration drift



Overall, 78% of respondents agree that GitOps prevents configuration drift. For high performers, this increases to 90%.

Automatic pull and **continuous reconciliation** should automatically correct changes made outside of the GitOps process. Organizations missing these practices will likely still experience configuration drift.

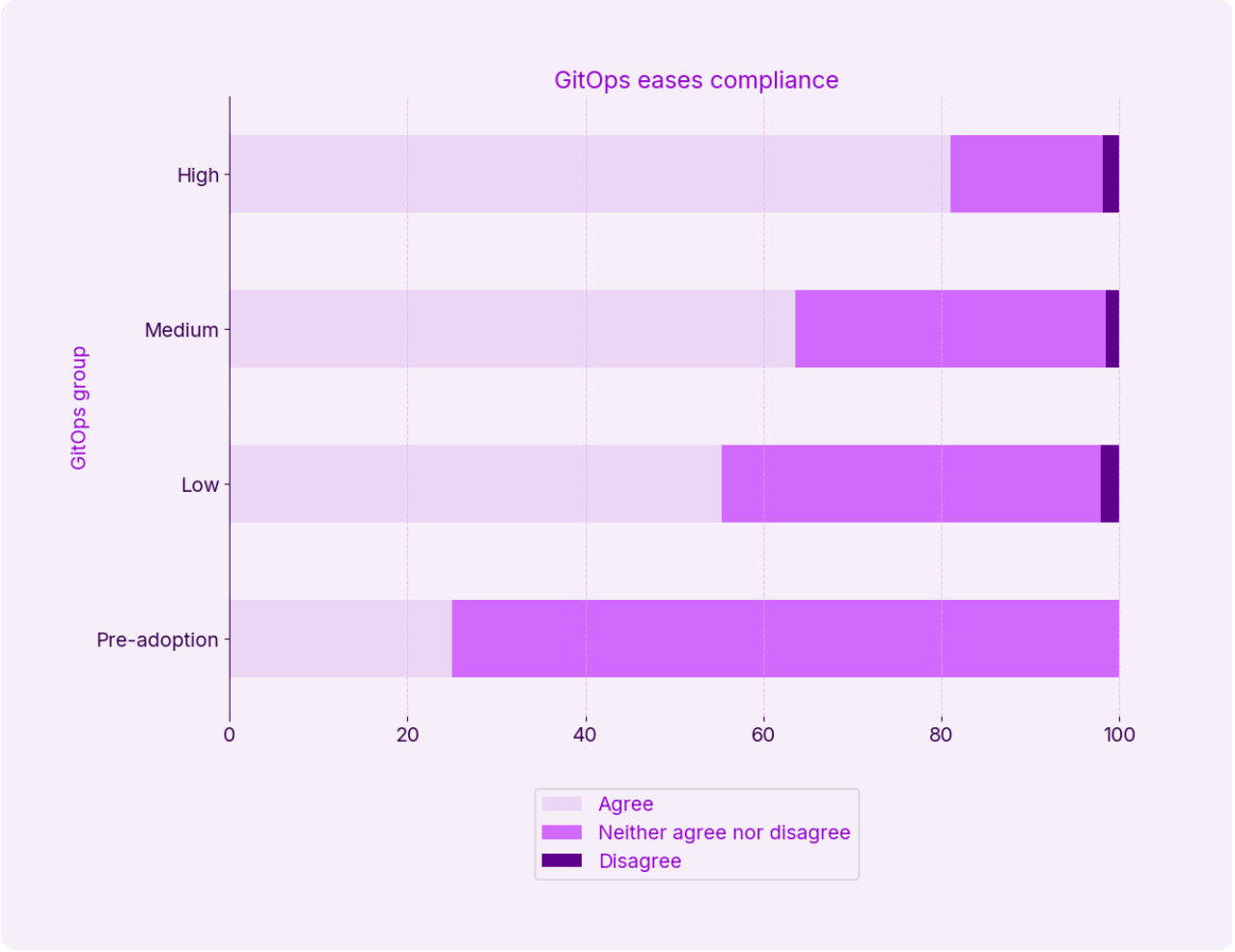
GitOps improves auditability



Overall, 81% of respondents agree that GitOps improves auditability. For high performers, this increases to 93%.

The use of **version control** with **human-readable formats** should make it easy to see a complete history of changes with a clear diff, dates and times, authorship, and commit messages. However, organizations that lack **responsive code review** and **continuous reconciliation** may be hampered because changes made outside of GitOps won't be audited.

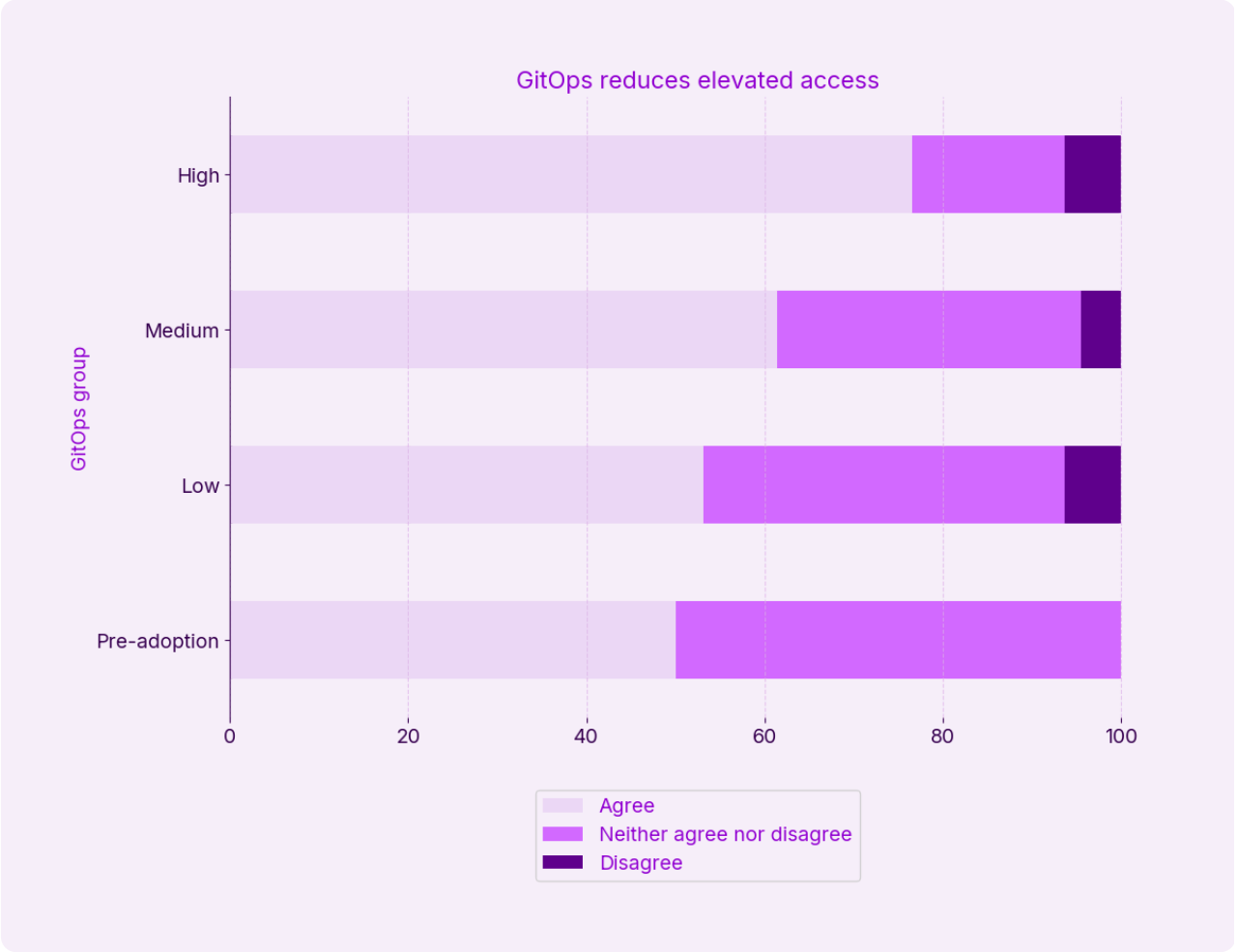
GitOps eases compliance



Overall, 68% of respondents agree that GitOps eases compliance. For high performers, this increases to 81%.

Using **version control** makes it easy to evidence separation of duties and **responsive code review** ensures work doesn't get blocked and encourages smaller batches of change. Practices such as policy-as-code, policy engines, and pre-commit hooks may also ease compliance for GitOps teams. The good practices a team would apply to version control for application code are transferable to GitOps.

GitOps reduces elevated access



Overall, 66% of respondents agree that GitOps reduces elevated access. For high performers, this increases to 77%.

Making all changes through **version control** with **automatic pull** and **continuous reconciliation** should mean fewer people need direct access to infrastructure. Those with high GitOps adoption could remove administrative access except in break-glass (emergency) scenarios.

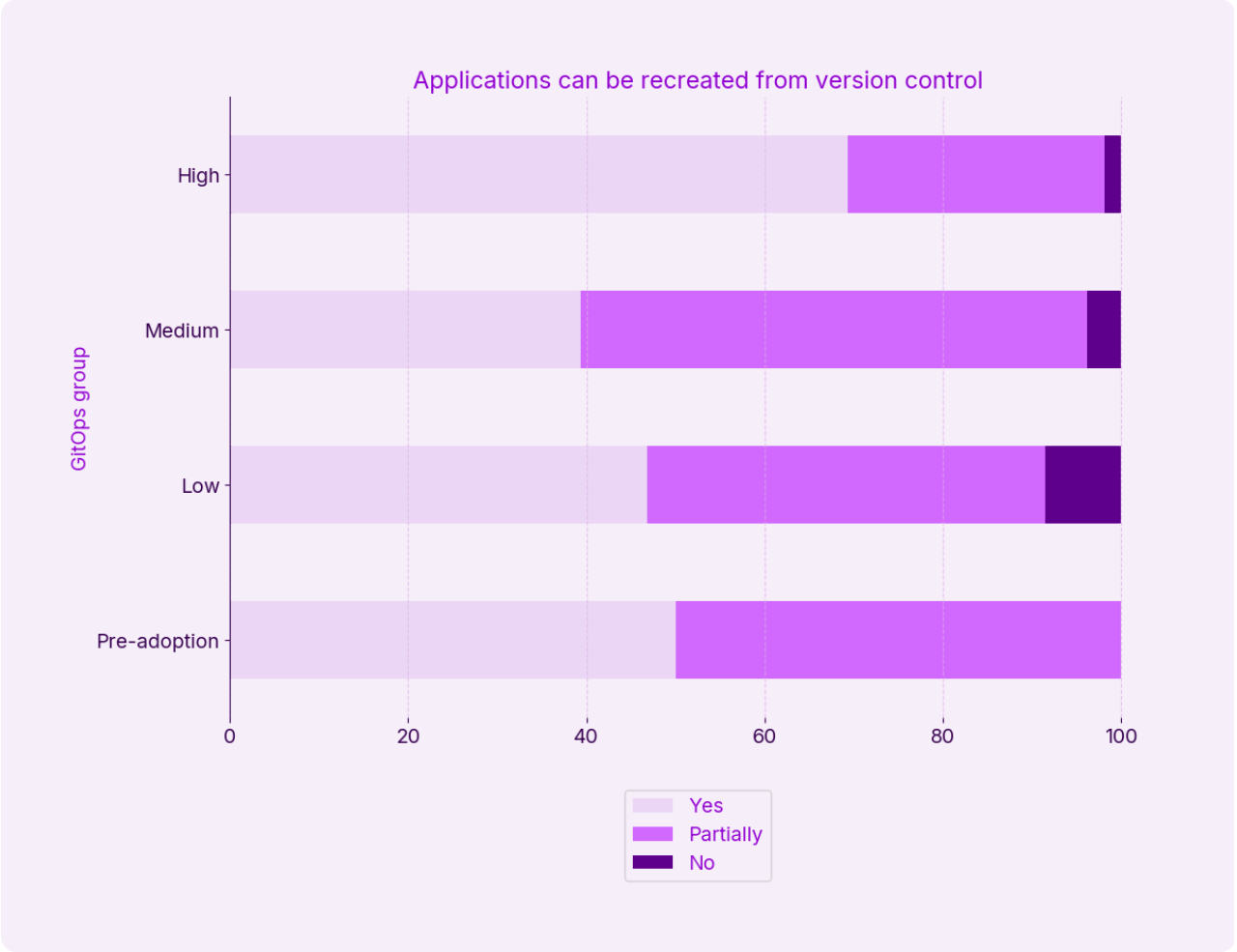
At the highest performance levels, there is a small increase in disagreement that GitOps reduces elevated access. This may reflect a change in perception about the version control repository, which is now the interface to infrastructure changes. This highlights the importance of secure repository access, branching policies, required reviewers, and approvals for changes to sensitive environments.

GitOps benefits score board

Based on the responses to the GitOps benefits questions, we can create a scoreboard for the areas in which GitOps has the most significant impact.

Benefit	All responses	High performers
Auditability	81%	93%
Drift prevention	78%	90%
Compliance	68%	81%
Reduced access	66%	77%
Increased security	63%	66%

Bonus: Recreatable applications



In addition to the direct benefits, we asked whether respondents could recreate their applications based on the configuration in version control. This is relevant to disaster recovery, easy cloud migration, and using ephemeral environments for testing. Contrary to what we found across the other benefits, confidence in the ability to recreate applications diminishes initially and then increases dramatically for the higher-scoring group.

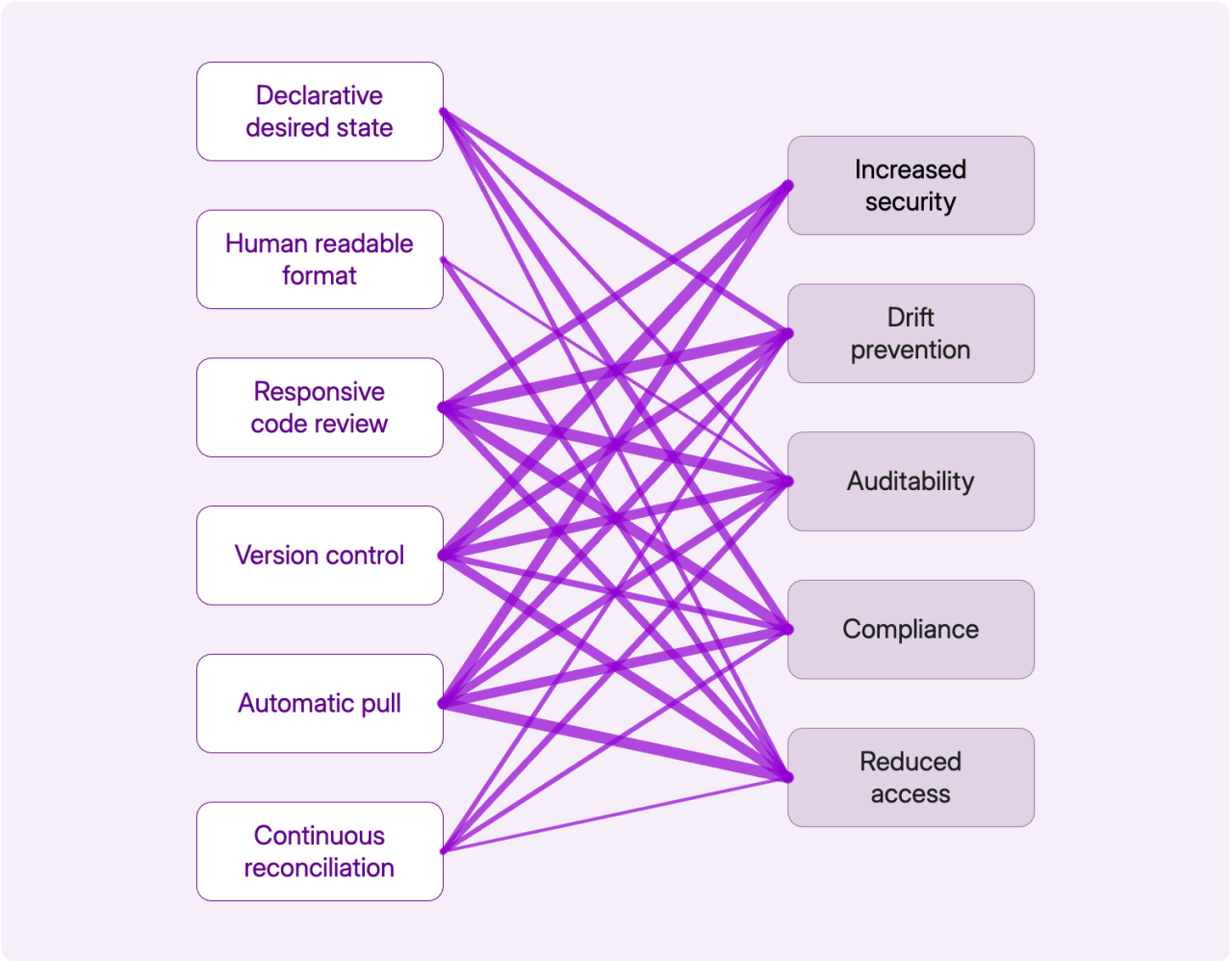
This may represent the learning curve that occurs as you make progress. For each item you bring under the control of GitOps, you'll likely discover more items you didn't originally consider.

Partial recreation may indicate situations where initial infrastructure must be provisioned before GitOps automation can run. This would be necessary if you must obtain new secrets or depend on baseline infrastructure that isn't automated. It also represents situations where additional changes are needed further out from the application and its immediate infrastructure, such as DNS record changes. In the event of disaster recovery, stateful parts of the system, like databases, may need to be restored.

Overall, 52% of respondents agree that GitOps lets them recreate their application from version control. For high performers, this increases to 70%.

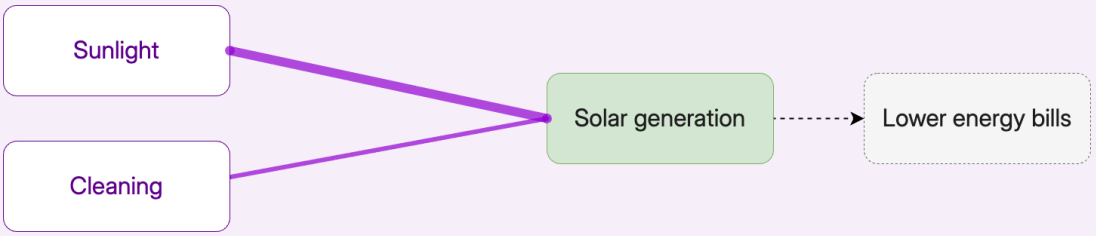
There may be techniques from DevOps and Continuous Delivery that complement GitOps in making applications recreatable. For example, test data management provides a consistent baseline for automated tests when you spin up a test environment. As DevOps inspired GitOps, it can be useful to reflect on previous research on the techniques and practices found in DevOps research, like DORA's **Accelerate State of DevOps report**[7].

How GitOps practices relate to key benefits



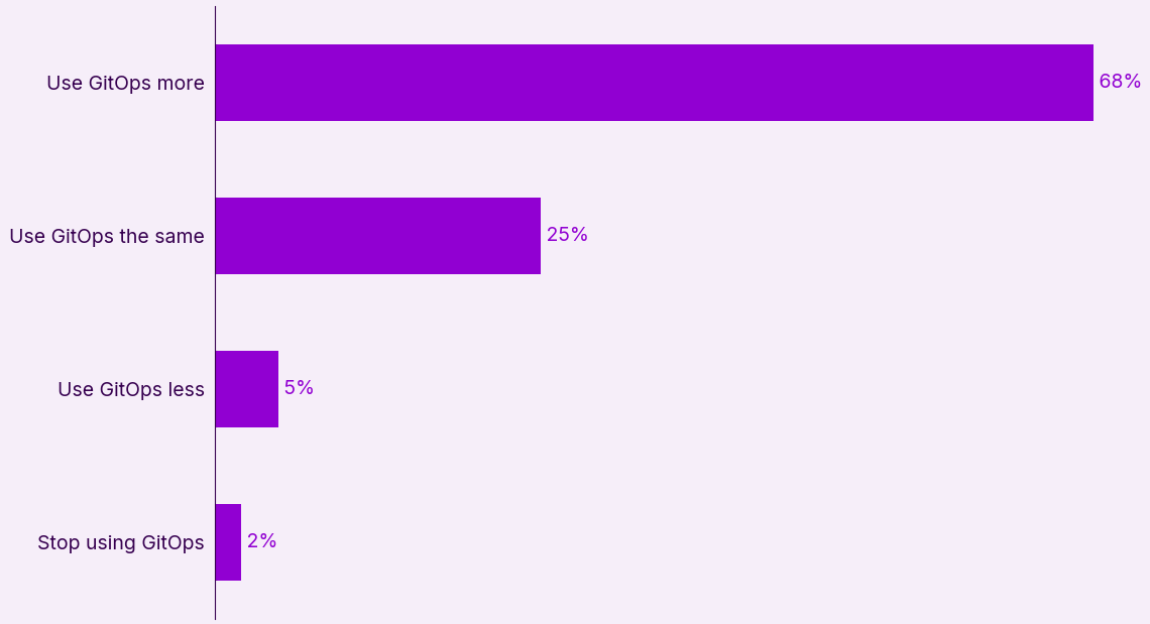
We can go a step further in explaining the relationships between GitOps and these benefits by examining the strength of relationships between the GitOps practices and expected benefit. You can use this diagram by selecting your desired outcome and adopting practices linked to it. For example, *increased security* is likely obtained from **version control**, **automatic pull**, and **responsive code review**.

In this section we use strength-relationship diagrams like the one below.



When a line is present, it indicates we found a positive relationship between the practice and the outcome. The stronger the relationship, the thicker the line.You can find more information on these diagrams in the *method* section at the end of the report.

Future GitOps plans



***Question**: What does your current organization's future GitOps adoption look like?*

Most respondents (93%) plan to use GitOps the same amount or more. Only 7% plan to reduce their use. There is a clear link between partial GitOps adoption and plans to reduce GitOps use. Organizations adopting GitOps should use the GitOps model in this report to guide their adoption and ensure they progress to a point where they can see a return on their investment.

The lack of automatic reconciliation (enabled by **automatic pull** and **continuous reconciliation**) is the leading reason for disillusionment with GitOps. Teams that store configuration in version control and automatically apply it are the least likely to stop using GitOps. Teams that manually apply changes made in version control are three times more likely to stop using GitOps.

GitOps and DevOps

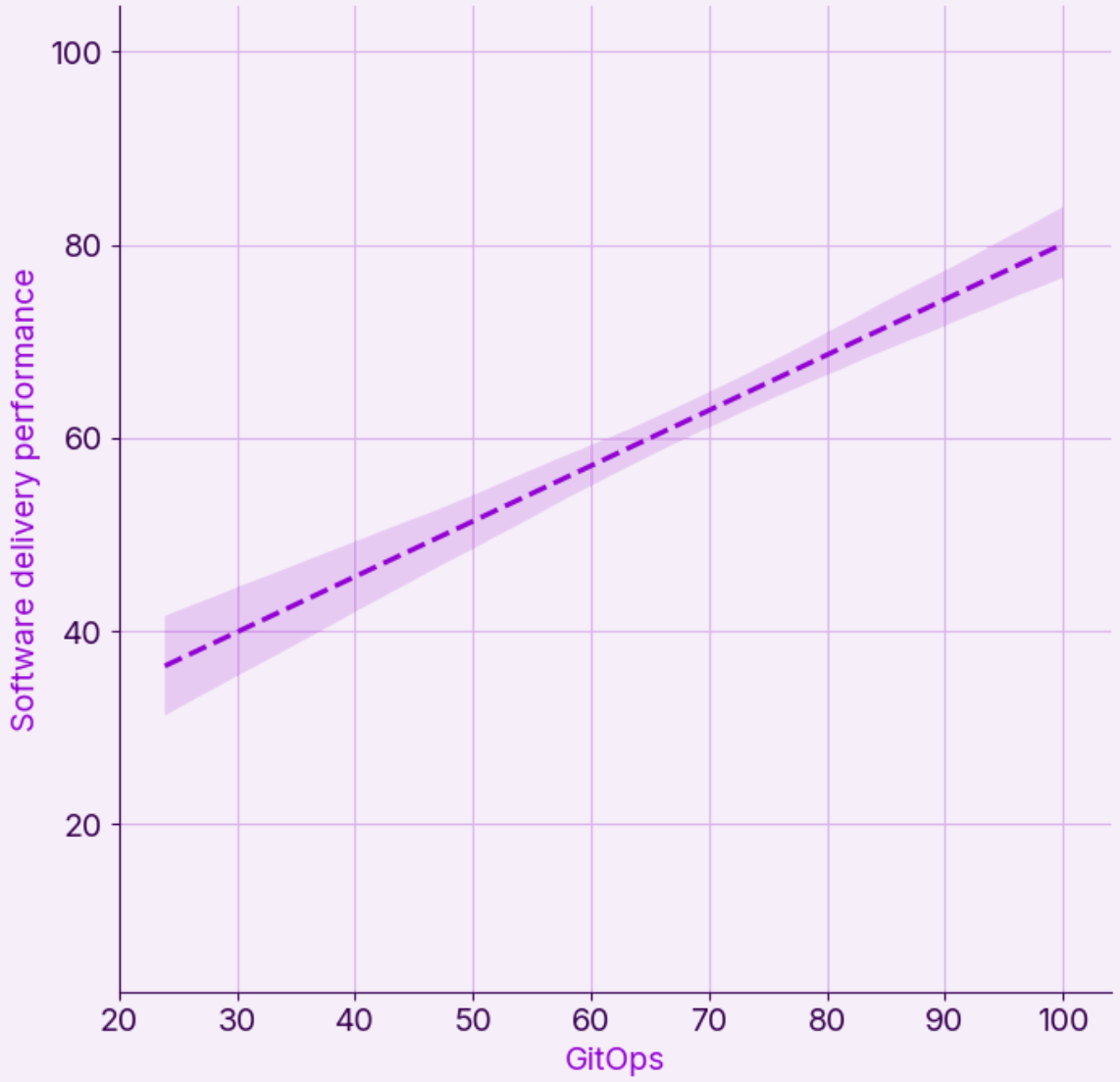
GitOps was originally inspired by DevOps and infrastructure as code (IaC). Though GitOps has different goals from DevOps, it would likely share a strong relationship with many DevOps outcomes besides the GitOps-specific benefits.

We investigated the following DevOps outcomes using established survey questions to make the comparison as direct as possible:

Outcome	Measurement
Software delivery performance	Assessed by 4 questions on the throughput and stability of deployment pipelines.
Reliability	Assessed by 4 questions on reliability targets, slowdowns, and user satisfaction.
Wellbeing	Assessed by 8 questions on work sentiment, attitudes to work, and burnout.

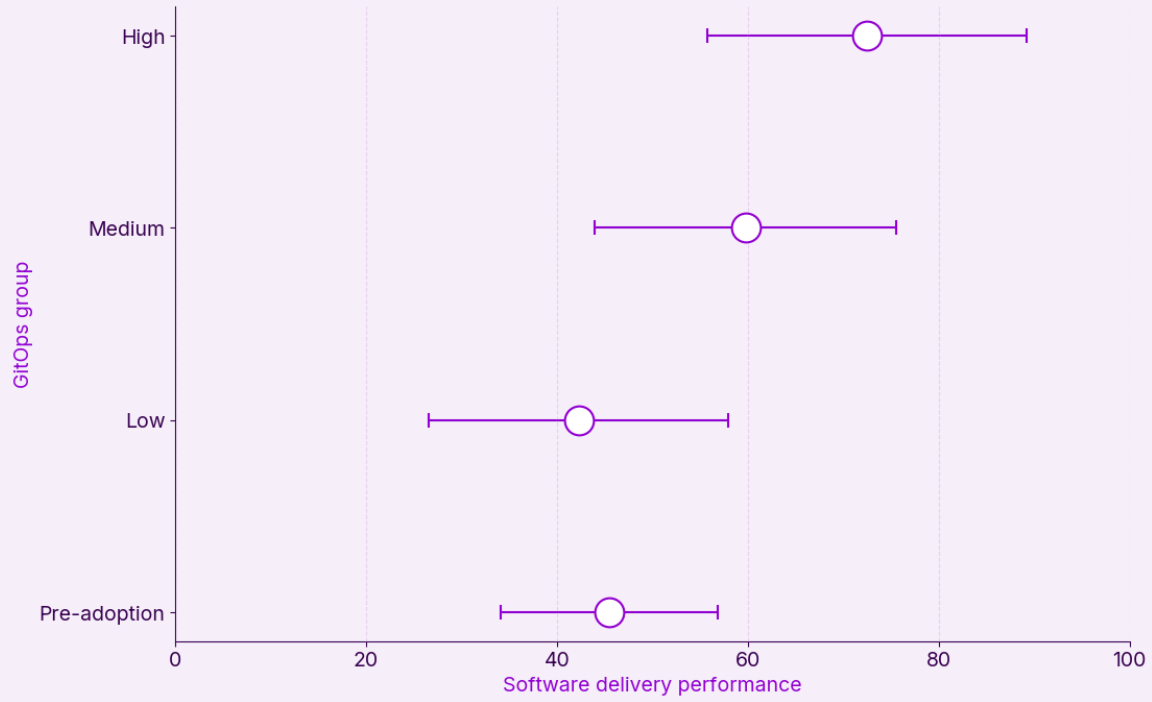
DORA's **State of DevOps research**[7] provides more information on these measurements and their relationship to organizational performance.

Software delivery performance



GitOps and software delivery performance

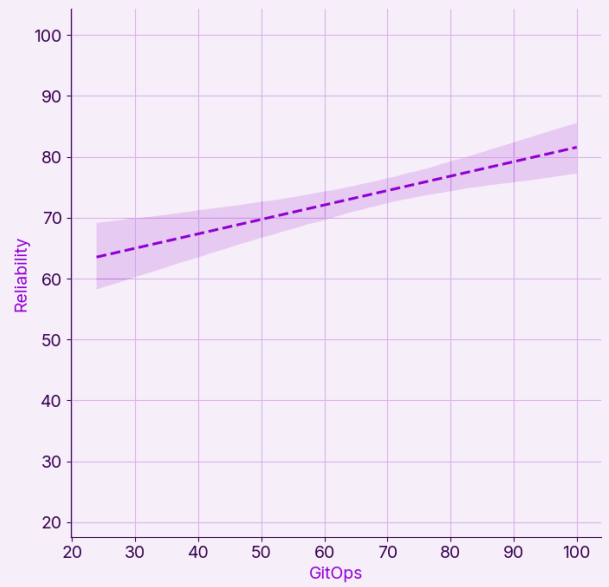
One of our cornerstone findings is the strong relationship between GitOps scores and software delivery performance. Teams with higher GitOps scores also perform better against the DORA 4 key metrics. That means higher GitOps performance levels are associated with increased deployment frequency, shorter lead time for changes, lower change failure rates, and faster recovery time after a failed deployment.



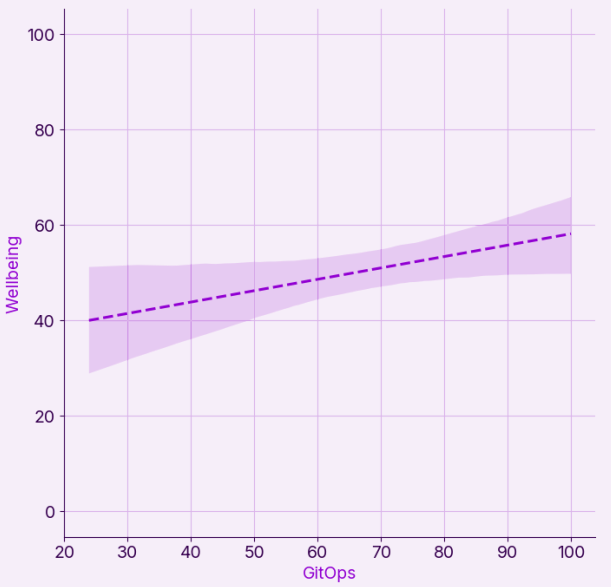
Software delivery performance by GitOps score group.

Teams with a GitOps score of 25-50 fail to outperform those with a score of 0-25, suggesting a critical mass of practices is required to unlock this outcome. Software delivery performance increases for those scoring 50-75 and again for those scoring 75-100.

The drop in performance while organizations adopt new practices, often called a j-curve, is becoming well-established in the software industry. As teams adopt new practices and learn how to use them well, there's an initial drop in performance against outcomes. By persevering, teams recover and then exceed previous performance levels.



Gitops and reliability



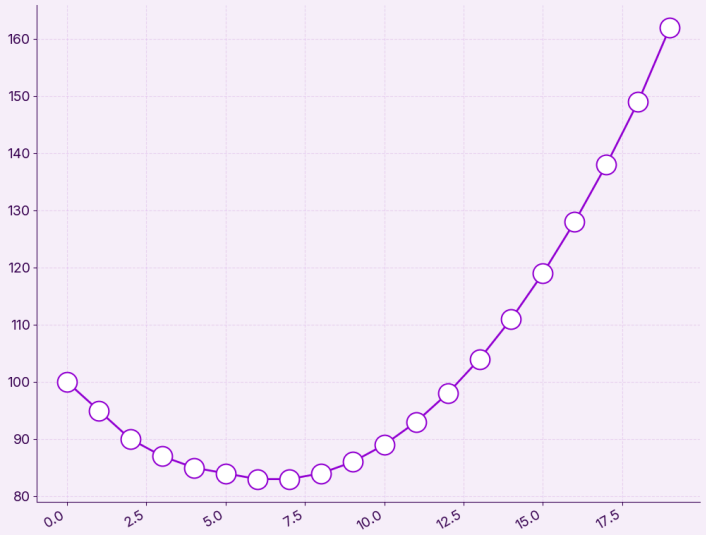
Gitops and wellbeing

The relationship between GitOps scores and reliability is less steep. Established practices, such as architecture, infrastructure choices, observability, and blameless incident review, have a direct effect on reliability. To improve reliability, you'd start with these. However, GitOps provides additional gains for teams once they have the fundamentals in place.

Similarly, wellbeing is influenced by organizational culture, processes, and interactions. We tested the effect of GitOps on wellbeing to determine whether the discipline and control of a GitOps approach negatively impacted wellbeing. The small but positive relationship between GitOps scores and wellbeing suggests that GitOps is associated with improved wellbeing.

J-curves

A productivity J-curve describes what happens when you adopt a new tool or system to improve your productivity. The name comes from the shape of the curve when graphed - it looks like the letter J.

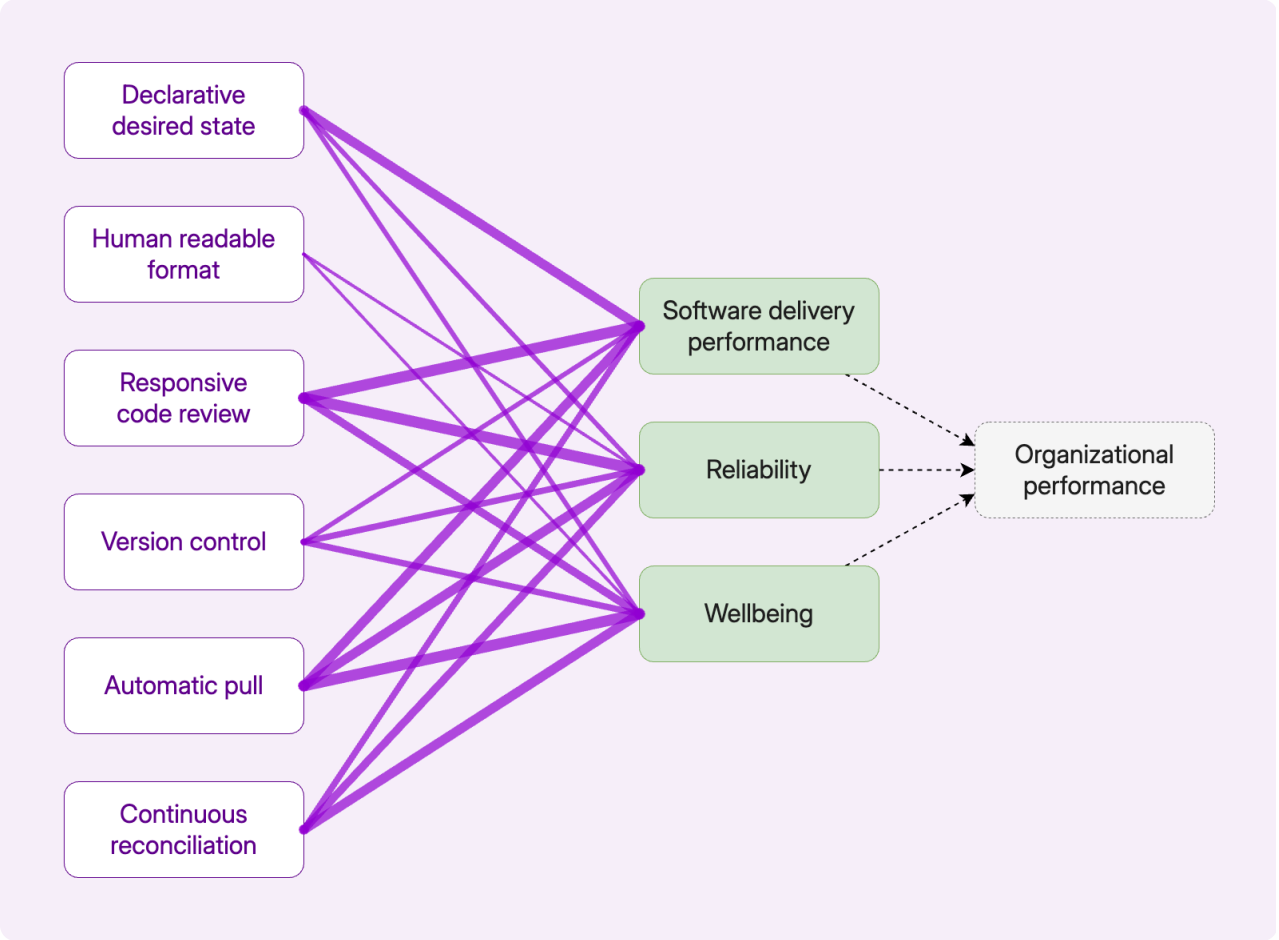


Here's what happens:

1. **Initial Drop**: When you first start using the new tool, your productivity actually goes down for a while. This happens because you need time to learn the new system, adjust your habits, and work through problems.
2. **Recovery**: After some time, you start getting comfortable with the new tool, and your productivity begins to improve, returning to your original level.
3. **Net Gains**: Eventually, you become proficient with the new tool and your productivity exceeds your original level - sometimes by a significant amount.

The key insight is that there's always a temporary dip before the benefits kick in. Many people give up during this dip because they think the new tool is making things worse, when really they just haven't reached the upward part of the J-curve yet.

Mapping practices to outcomes



We can further understand the relationships between GitOps and software delivery by examining the strength of relationships between the GitOps practices and the three software delivery outcomes. The thickness of the lines in the strength relationship diagram indicates how much impact each practice has on the outcomes.

Conclusions

There is growing interest in GitOps and a strong desire to increase adoption over the next 12 months.

Our research has explored the adoption of practices considered foundational to successful GitOps. We found that partial GitOps adoption is less successful in terms of the expected benefits and in relation to established software delivery outcomes. In particular, organizations missing **automatic pull** and **continuous reconciliation** are the most likely to consider moving away from GitOps.

The 6 practices of GitOps complement and amplify each other. That means you'll only get the full benefits of GitOps by adopting all of them.

1. Declarative desired state
2. Human readable format
3. Responsive code review
4. Version control
5. Automatic pull
6. Continuous reconciliation

GitOps benefits each depend on between 3 and 6 practices, while the DevOps outcomes depend on 5 or more.

This builds on the existing set of capabilities we know are required for high-performing teams, including cultural qualities. Managing software at scale is complex, but GitOps certainly helps.

In practice, we wouldn't expect a team to adopt all 6 practices at the same time. A process of continuous improvement that allows teams to establish a practice and increase their skills before adding more new techniques will likely be more successful than a big bang approach. This also allows team to handle situational challenges, such as working out whether they can establish a declarative desired state for an existing application.

You're likely to encounter the j-curve as you adopt GitOps. That means you'll find an initial drop in performance as you adopt, embed, and perfect each technique. The reward for this work is an eventual higher level of performance that pays it all back with interest.

Improving GitOps outcomes

To help you improve your GitOps outcomes, you'll find below a summary of each practice and the benefits most associated with it. You can use information along with the model and relationship diagrams in the report to inform your continuous improvement process.

GitOps practice: Declarative desired state

Declarative configuration provides a verifiable target state. In contrast, Imperative configuration requires you to run through steps and understand how they would change baseline state, which means you need to hold the whole process in your mental model to understand the end result. This also leaves room for assumptions and omissions to result in a state that isn't what you imagined. Declarative configuration describes the end state, which is simpler to understand. It pushes the burden of achieving that state onto automation tools.

With declarative desired state you get:

- A documented target state.
- Easier changes and reviews.
- A readable change history and audit trail.
- Transfer of reconciliation burden to tools.

GitOps practice: Human readable format

The change history and audit trail provided by version control tools loses most value if the configuration files aren't human readable. Instead of having simple diffs to compare, you may end up having to load different versions of the configuration into the original tool for comparison. This may limit the features available for comparing versions, or even make it a spot-the-difference exercise.

With human readable formats you get:

- Easier changes that follow the developer-style workflow.
- Easier reviews as the differences can be viewed within code review tools.
- Change history and audit trails from version control.
- Self-documenting target state.

GitOps practice: Responsive code review

Fast code reviews keep the flow of changes moving. Slow GitOps reviews delay feedback and encourage feature teams to work in larger batches, which is where many software delivery problems emerge. The slower your GitOps change reviews are, the more likely you'll be to make changes outside of GitOps. Those changes won't be reviewed or appear in the change history. Routinely making changes to the state without using GitOps also pushes you to stop continuous reconciliation.

With responsive code reviews you get:

- Smaller batches, which have less risk.
- The ability to make version control the primary interface for system changes.
- Better throughput as people are no longer blocked waiting for a review.

GitOps practice: Version control

Many people say version control is a fundamental principal of GitOps, though the preceding practices are crucial to making the use of version control effective. Version control supplies the tools for access control, storage, retrieval, review, history, and audit trail for the desired state and its changes. Version control was selected as the tooling is already well-established in software delivery.

Using version control gives you:

- A source of truth for the desired state of the system.
- Familiar tools, controls, and audit trails.
- A complete history of changes and mechanisms to rollback changes.

GitOps practice: Automatic pull

Using an agent to pull configuration from version control means you don't need to open up access for remote command execution. Instead of a central orchestrator making calls to commands or APIs, the agent collects the configuration regularly from its source.

Having an agent automatically pulling desired state means you:

- Increase security as you don't need to expose endpoints.
- Don't need to co-ordinate the destinations for configuration.

GitOps practice: Continuous reconciliation

Continuous reconciliation ensures the system is kept in the desired state. It's perhaps the best way to encourage people to make all changes through version control as manual ClickOps changes are quickly identified and corrected. While it's valid to implement continuous reconciliation with manual intervention by raising synchronization issues for a human to review, this leaves the system operating in the wrong state for longer periods and encourages direct hotfix changes that don't get captured in the history and audit trail in version control.

Enabling continuous reconciliation gives you:

- Confidence that the system remains in the intended state.
- Motivation for all changes to be made through version control.

Avoiding GitOps trip hazards

The second way to improve GitOps outcomes is to avoid common trip hazards. A number of these are listed below and can be used to identify negative patterns in your GitOps adoption.

Trip hazard: Resource deletion

With a high level of automation, a simple commit could trigger the deletion of critical production resources. This scenario becomes more dangerous when configuration changes aren't properly validated or developers lack awareness of the downstream effects of their commits.

Fortunately, several protective measures can mitigate this risk.

1. **Resource protection policies** create a safety net that prevents automated deletion of tagged critical resources, even if configuration files suggest otherwise.
2. **Dry-runs** let you preview the proposed changes before applying them to production environments, catching potential issues early.
3. **Approval workflows** for production changes mean multiple team members review modifications before they are applied to critical systems.

These practices, combined with testing in lower environments and proper documentation of protected resources, create multiple layers of defense against accidental deletions.

Trip hazard: Leaking secrets

Storing sensitive data in version control can result in a dangerous security vulnerability. When credentials, API keys, encryption keys, or other secrets are committed to repositories in plain text, they become easily discoverable by anyone with repository access.

Even if a repository is private today, historical secrets remain vulnerable through repository history if you later expand access. This risk extends beyond immediate threats, as leaked credentials can remain exploitable if not properly rotated or revoked after exposure.

You should adopt a stringent "no secrets in manifests" policy to safeguard sensitive information. Instead of injecting secrets directly into configuration files, use specialized secret management tools that provide encryption, access controls, and audit capabilities. You can integrate these tools with GitOps workflows through secure references or operators that retrieve secrets during deployment without exposing them in version control.

You can also use pre-commit hooks and automated scanning tools to detect credential patterns helps catch accidental secret commits before they reach the repository.

By separating configuration from sensitive data and establishing clear processes for secret management, you can maintain the automation benefits of GitOps while significantly reducing risk.

Tip hazard: Overloading version control

In high-frequency GitOps environments, infrastructure automation can inadvertently become a victim of its own efficiency. As deployment pipelines continuously poll version control systems for changes, you may unexpectedly encounter API rate limits or resource exhaustion issues, especially in large-scale implementations with numerous clusters or environments.

These limitations can manifest as failed synchronizations, delayed deployments, or complete workflow disruptions. Such resource constraints become particularly problematic during critical deployment windows or incident response scenarios when system reliability is paramount.

You should implement more efficient change detection mechanisms to prevent version control systems from becoming operational bottlenecks.

Replacing frequent polling with webhook-triggered actions means deployments only happen when relevant changes occur, dramatically reducing API calls. Where polling remains necessary, decreasing the frequency to an appropriate cadence based on actual deployment needs can significantly reduce system load.

Implementing exponential back-off will reduce load during temporary resource constraints so they don't cascade into sustained outages

Trip hazard: Deployment pipeline gaps

When moving to GitOps, keep in mind the good practices you have built over time and transfer them to your new practices.

1. **Environment progression** doesn't just help you test application changes, it means your GitOps configuration is tested just as often as the configuration is applied to development, testing, and staging environments before production.
2. **Robust test automation** helps you catch issues early and can be used to validate new deployments. This increases your confidence in the application and its configuration.
3. **Dedicated change windows** mean you can be on standby to provide immediate support if a problem arises.
4. **Immutable infrastructure** eliminates drift and ensures environments remain aligned with their declarative definitions.

By preserving these established deployment practices within a GitOps context, organizations can benefit from automation while maintaining the safeguards that protect business-critical systems.

Trip hazard: Process gaps

With GitOps, the version control repository undergoes a fundamental transformation from simple code storage to becoming the authoritative control plane for production infrastructure. This creates a unique security and governance challenge, as repository access directly translates to production change capabilities.

1. **Robust role-based access control** (RBAC) for Git repositories ensures that only authorized personnel can change production-affecting configurations with appropriate approval gates for sensitive areas.
2. **Automated drift detection and alerting** provides continuous verification that deployed environments match their declared states, immediately flagging modifications made outside the GitOps workflow.
3. **Regular disaster recovery** exercises focusing specifically on repository loss or corruption scenarios ensure that teams can quickly restore the control plane if version control systems experience failure.

By treating repositories with the same security rigor traditionally reserved for production control systems, organizations create a GitOps implementation that maintains proper separation of duties while preserving automation benefits.

Mutually supportive practices

As you can see GitOps is a collection of mutually supportive practices. When you remove one of the practices, it leaves gaps in the effectiveness of the others. In addition to the practices recommended by GitOps expert practitioners, we discovered the importance of responsive code reviews for GitOps changes. This is the minimal definition for successful GitOps as we see it today.

As we continue our research, we may find other techniques that complement GitOps. We may find these are also fundamentally important, or that they play a more contextual or situational role.

The State of GitOps Report investigated the adoption, practices, and benefits of a GitOps approach. Now, it's up to you to use this knowledge to improve your unique situation. We encourage you to try the concepts and share what you learn.

You can join our mailing list to receive the latest research updates as we continue our research. Visit **oc.to/research-updates** to join.

Contributors

Steve Fenton

Principal DevEx Researcher

A long-time software delivery practitioner and research lead for the State of GitOps report

Bob Walker

Field CTO

A software engineer and architect with over two decades of industry and deployment automation experience

Dan Garfield

VP GitOps and Open Source

Codefresh Co-Founder

Argo Maintainer

Co-Creator Open GitOps

Kostis Kapelonis

Senior Developer Evangelist

A software engineer and technical writer with deep GitOps experience and a member of the Argo team

Joanna Wyganowska

VP of Marketing

20 years of experience running product marketing for software startups and software giants alike

Matthew Casperson

Principal Solutions Engineer

A deeply knowledgeable technologist and author who works on the leading edge of technology

Idan Arbel

Principal Product Manager

A seasoned product management professional with deep knowledge of GitOps

Matt Allford

Developer Advocate

A technologist and content creator who combines 15+ years of deep technical experience with a passion for community education

Advisors and field experts

Ivan Krnic (**CROZ**)
Denis Jajcevic (**CROZ**)
Pim Polderman (**Formatic**)
Shiran Ginige (**Kodez**)
David Stenglein (**Missing Mass**)
Vlad Cantiru (**Arctiq**)
Sivamuthu Kumar (**CEI**)
Ian Crosby (**Aptum**)

Method

We performed preliminary interviews to inform our survey design and understand what factors and outcomes were essential to GitOps.

We promoted the survey organically to collect responses from a broad audience. The survey included questions to validate we obtained responses beyond our own audience, as we wanted to get a broad sample with experience of different tools and approaches.

To link to well-established software industry measures, we included questions from **DORA's research**[7] alongside questions we designed to understand GitOps. We limited the survey to 30 questions to keep completion times as low as possible while covering crucial topics.

Responses were cleaned, for example, to de-duplicate user-entered data like "backstage" and "back stage". We developed a scoring mechanism to convert responses into normalized scales for GitOps performance, software delivery performance, reliability, and wellbeing. This made it easier to compare different performance relationships. We also analyzed direct relationships to ensure we didn't miss important links between practices and outcomes.

For some areas of study, we filtered the dataset. For example, the wellbeing questions were optional. We only included responses with complete wellbeing answer sets when looking at wellbeing relationships.

Analysis

We used Python for data analysis and notebooks to organize and annotate the different threads of investigation. To find the presence and strength of relationships, we used **Pearson correlation coefficients**[8] to measure linear correlation. Software delivery is complex and requires a large number of practices for success. That means the strength of individual practices is often low, while combinations amplify each other to create a larger effect.

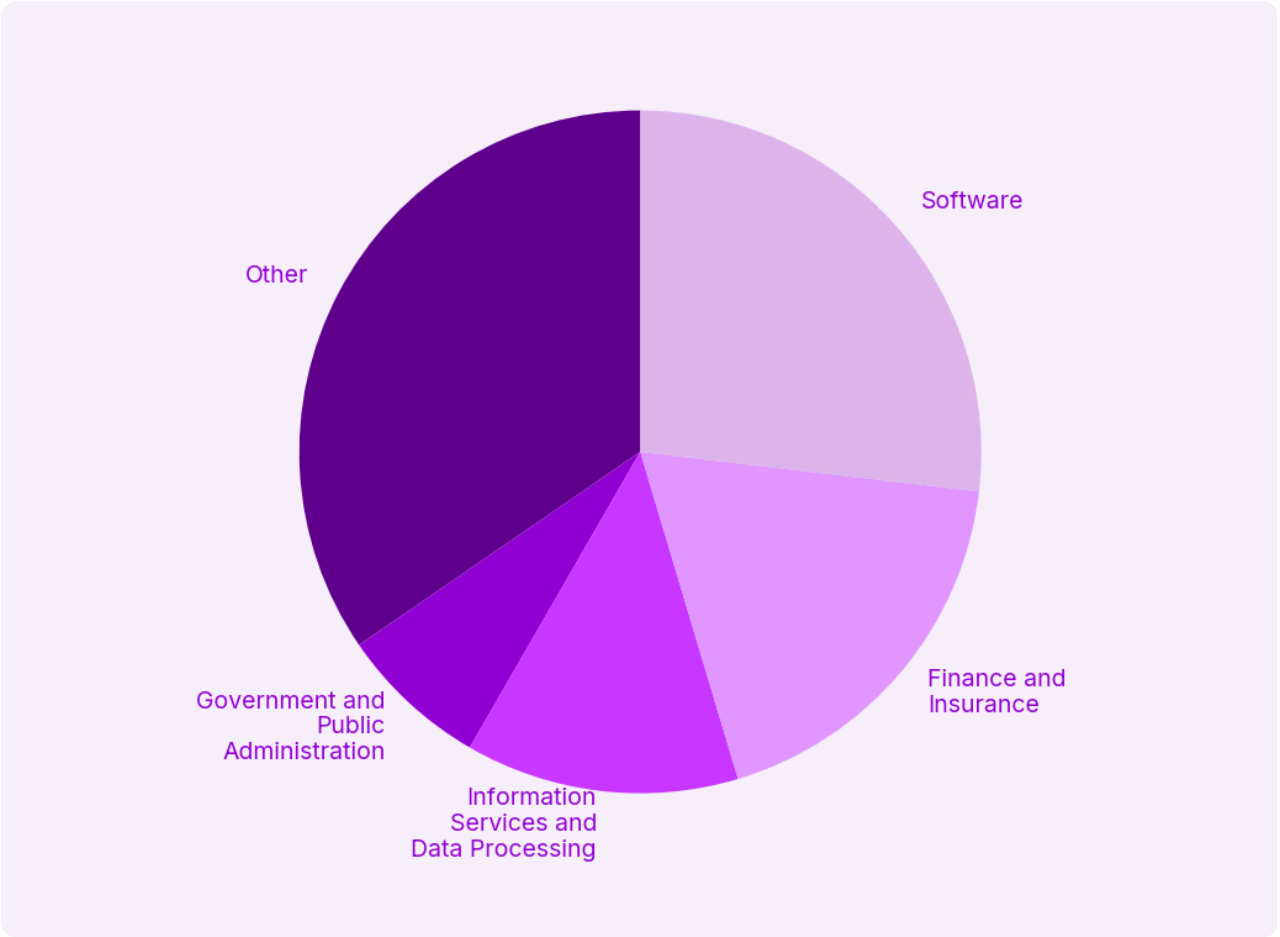
Our analysis is intended to present the current state of GitOps adoption, how differences in adoption impact outcomes, and practitioners' future attitudes to GitOps.

Interviews

We interviewed GitOps professionals, practitioners, and consultants to gain additional perspectives on our findings. Their views are included in the explanations for what we've found. This allowed us to draw on years of experience from different practical contexts to describe why GitOps looks the way it does.

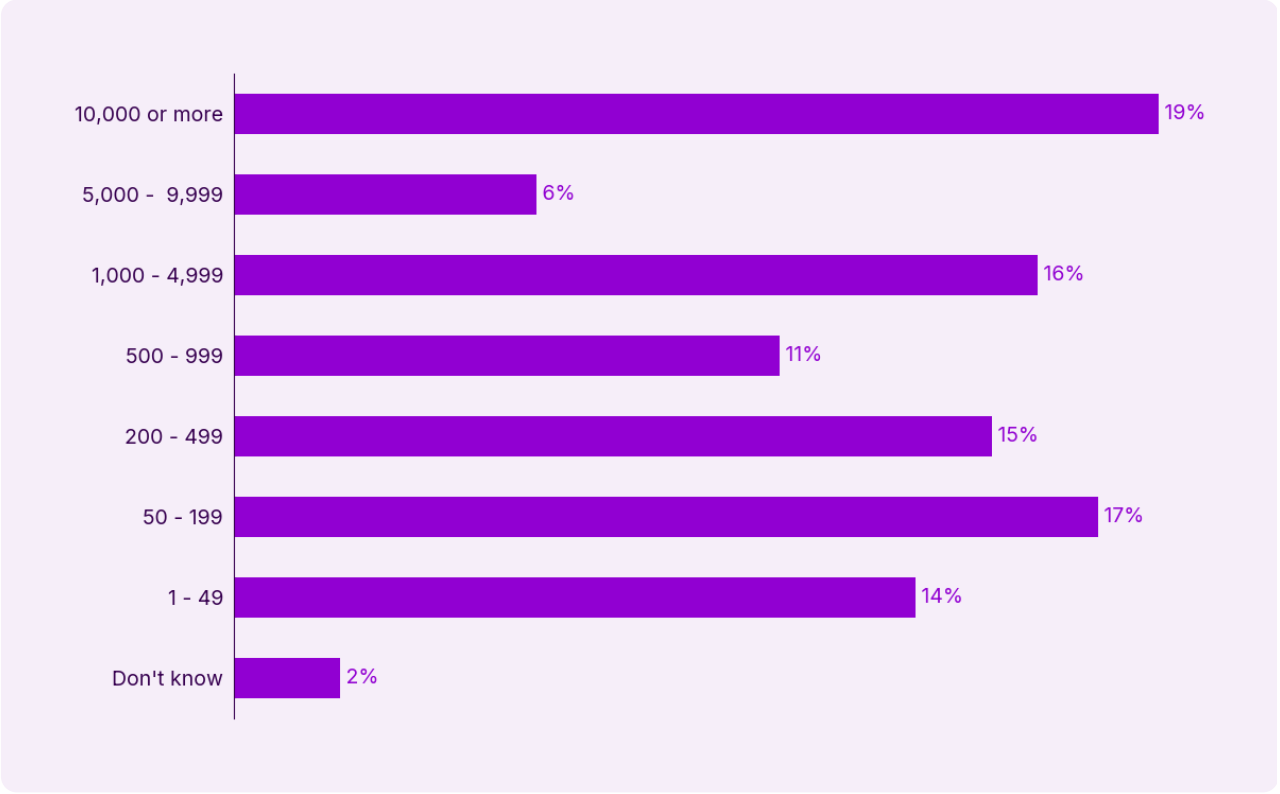
Firmographics

Industry



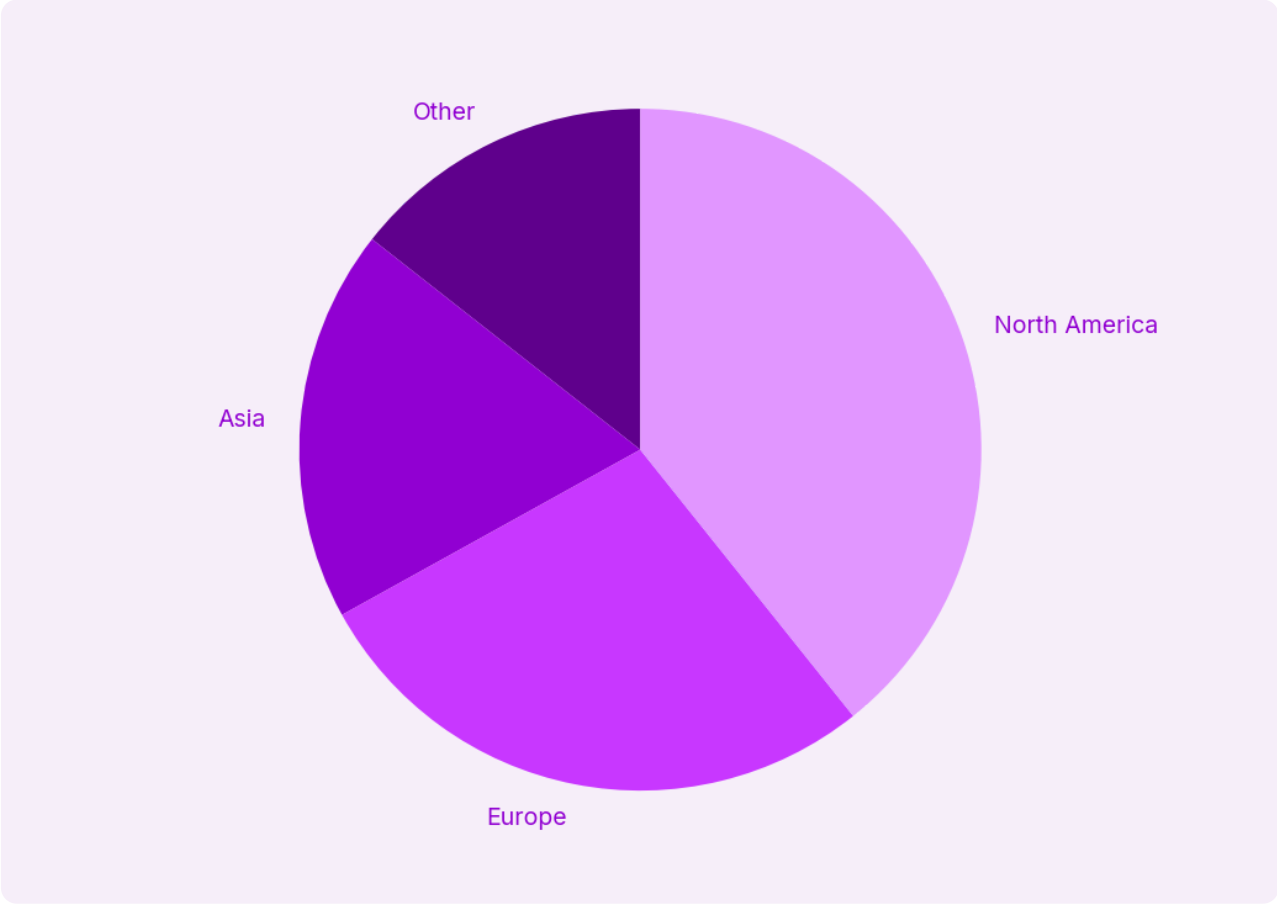
Overall, 22 industries were represented, though the top 4 industries accounted for two thirds of responses. The top industries were software (27%), finance and insurance (19%), information services and data processing (13%), and government and public administration (7%).

Organization size



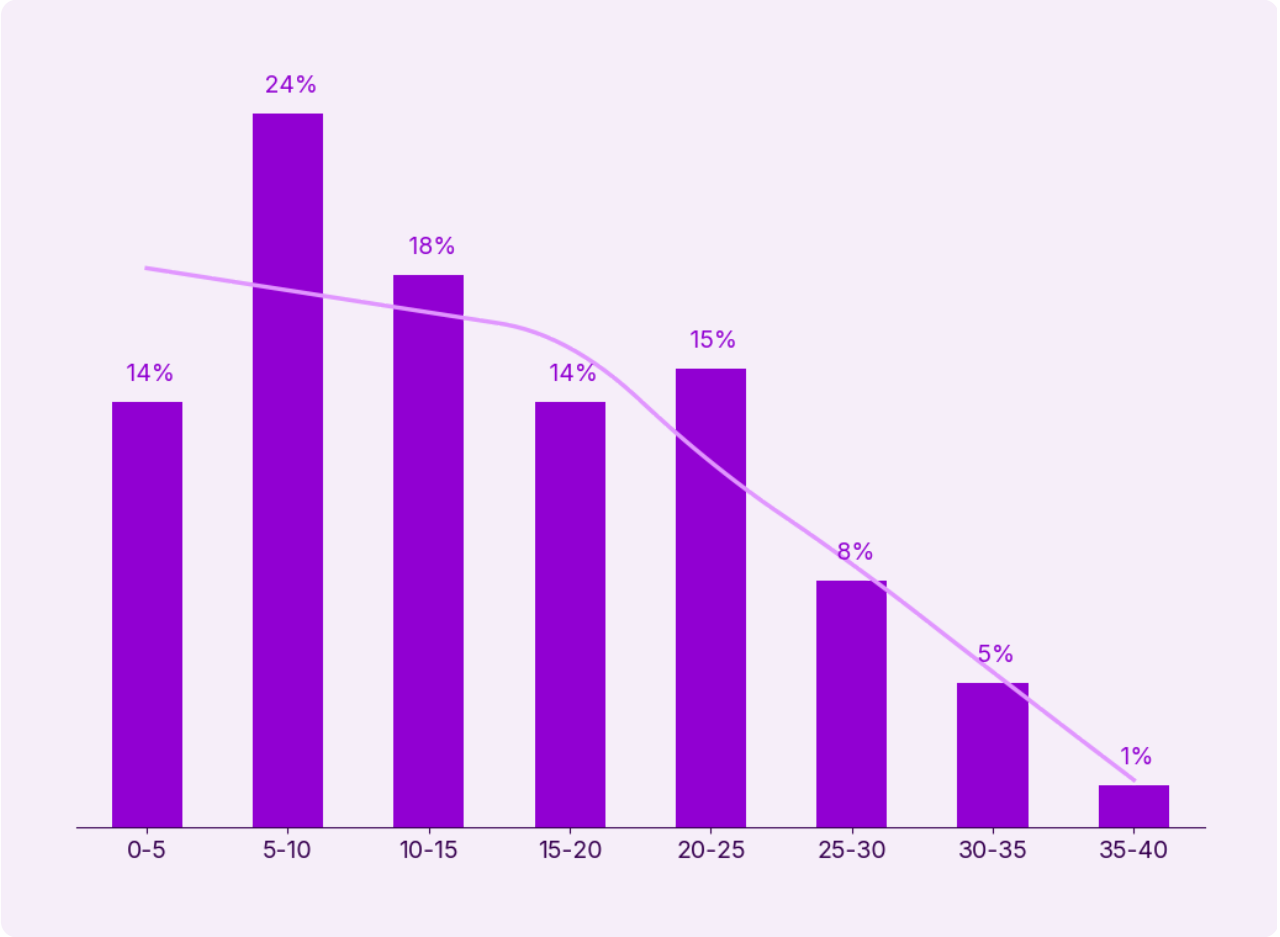
Responses were received from organizations of all sizes. Like comparable industry research, we saw lower numbers of organizations in the 500 to 999 and 5,000 to 9,999 size groups.

Location



The survey reached people in every continent on the planet, with most responses coming from North America (39%), Europe (28%), and Asia (19%). Oceania, South America, and Africa each had around 5%.

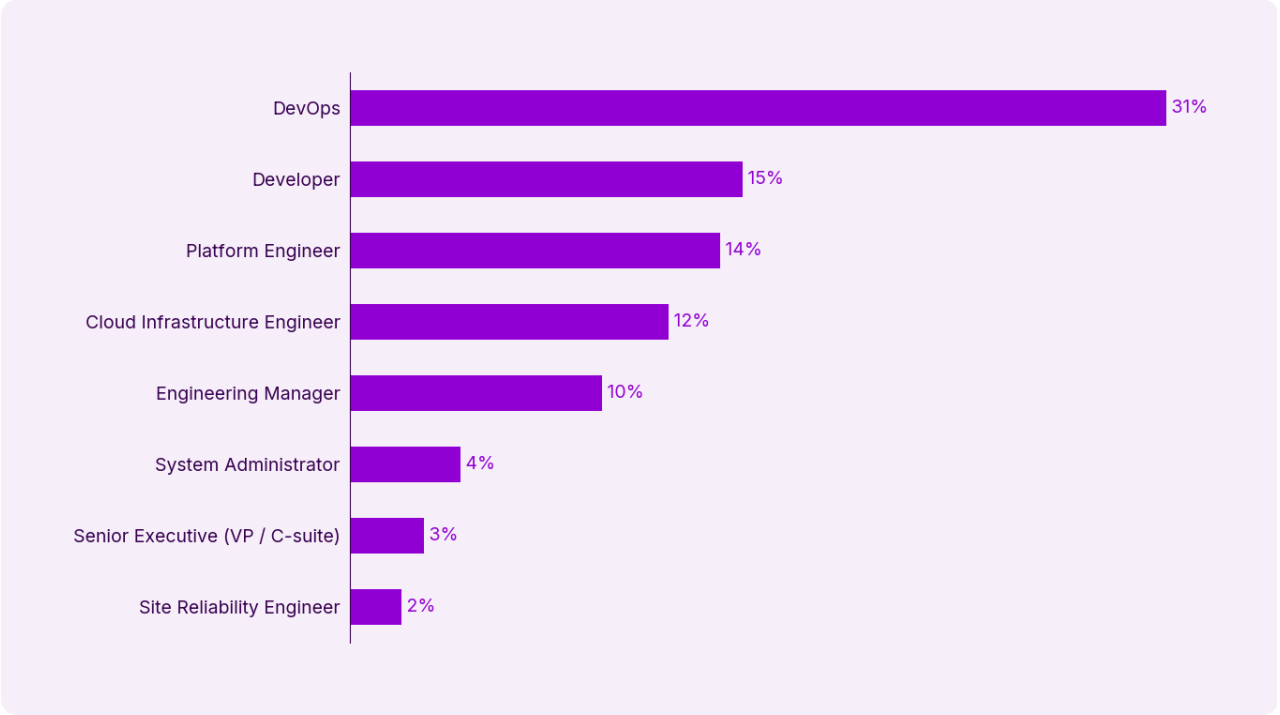
Years of experience



The experience of respondents largely follows the pattern we see in other industry research. The 20-25 years range would typically be between two adjacent ranges, so this differs from our expectation. The 0-5 year range often has a lower representation as early-career developers are less likely to respond to industry surveys.

As GitOps is a more advanced technique, we anticipate its practitioners are likely to have more experience, though this would need to be tested through a more general survey.

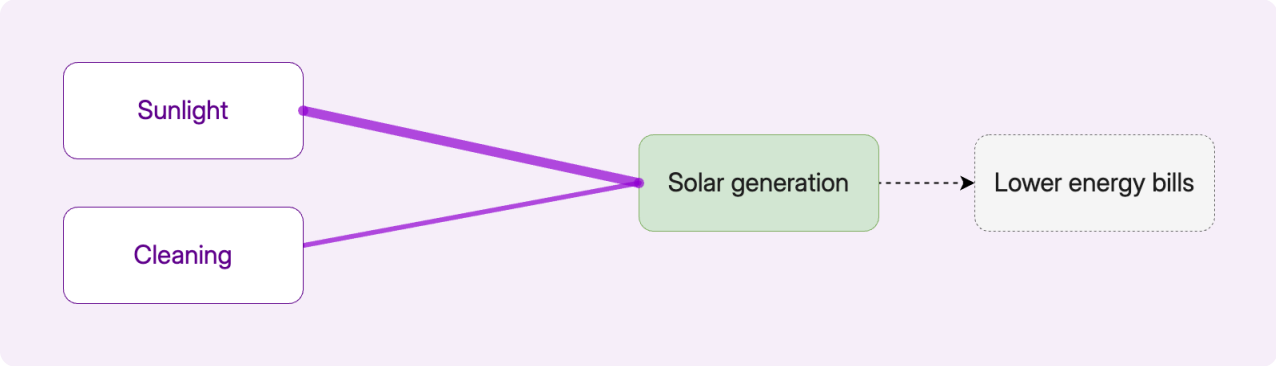
Job roles



The top 8 roles represented 90% of responses. They include the personas most likely to be practicing GitOps. The other 10% included 14 other roles, such as Architects, Database Administrators, Data Engineers, Testers, Security Professionals, and DevEx.

Strength of relationship diagrams

To illustrate the effect of practices on outcomes, we use strength of relationship visualizations. These show practices on the left with potential outcomes on the right. The presence of a line indicates a positive relationship was found and the line's thickness illustrates the size of the relationship. A dotted line refers to a pre-validated relationship that has not been re-examined in our research.



In the example, sunlight has the most significant impact on solar generation. Cleaning the solar panels also impacts the efficiency of generation, but not as much as the weather. The expected result of solar generation is lower energy bills.

To use these diagrams, you can work backward from the outcomes you want to achieve. By following the thickest line back to a specific practice, you'll get a suggestion for something you could introduce to boost your outcomes. If you've already implemented the practice with the strongest relationship, you can choose the next strongest, and so on. In other words, you place your solar panels in an optimal position to capture daylight first. If you still aren't getting enough solar generation, you may benefit from cleaning the panels to remove layers of dust.

Some diagrams have many lines, but you can filter out much of the information by focusing on your desired outcome. We plan to make interactive versions of these diagrams available.

Sponsors

Octopus makes it easy to deliver software to Kubernetes, multi-cloud, on-prem, and anywhere else at scale, in one platform.

Visit **octopus.com** to find out more.

Argo CD is great at syncing, but we help you test and promote changes in between. Your teams are drowning in scripts and pipelines to get changes from dev to prod, from east to west. With Codefresh you can define your promotion flow in a single CRD. Simple!

Visit **codefresh.io** to find out more.

References

1. https://github.com/bitnami-labs/sealed-secrets
2. https://github.com/crossplane/crossplane/releases/tag/v0.1.0
3. https://github.com/hashicorp/terraform/releases/tag/v0.6.0
4. https://opengitops.dev
5. https://codefresh.io/learn/gitops/
6. https://dora.dev
7. https://dora.dev/research/
8. https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Further reading

- CNCF: OpenGitOps
  https://www.cncf.io/projects/opengitops/
- CNCF: GitOps micro survey
  https://www.cncf.io/blog/2023/11/07/cncf-gitops-microsurvey-learning-on-the-job-as-gitops-goes-mainstream/
- GitOps 2.0: The Future of DevOps
  https://codefresh.io/ebooks/gitops-2-0-future-devops/
- Argo CD & Rollouts 2023 User Survey Results
  https://blog.argoproj.io/cncf-argo-cd-rollouts-2023-user-survey-results-514aa21c21df

Octopus Deploy

Level 4, 199 Grey St
South Brisbane, QLD 4101,
Australia

sales@octopus.com
+1 512-823-0256
octopus.com