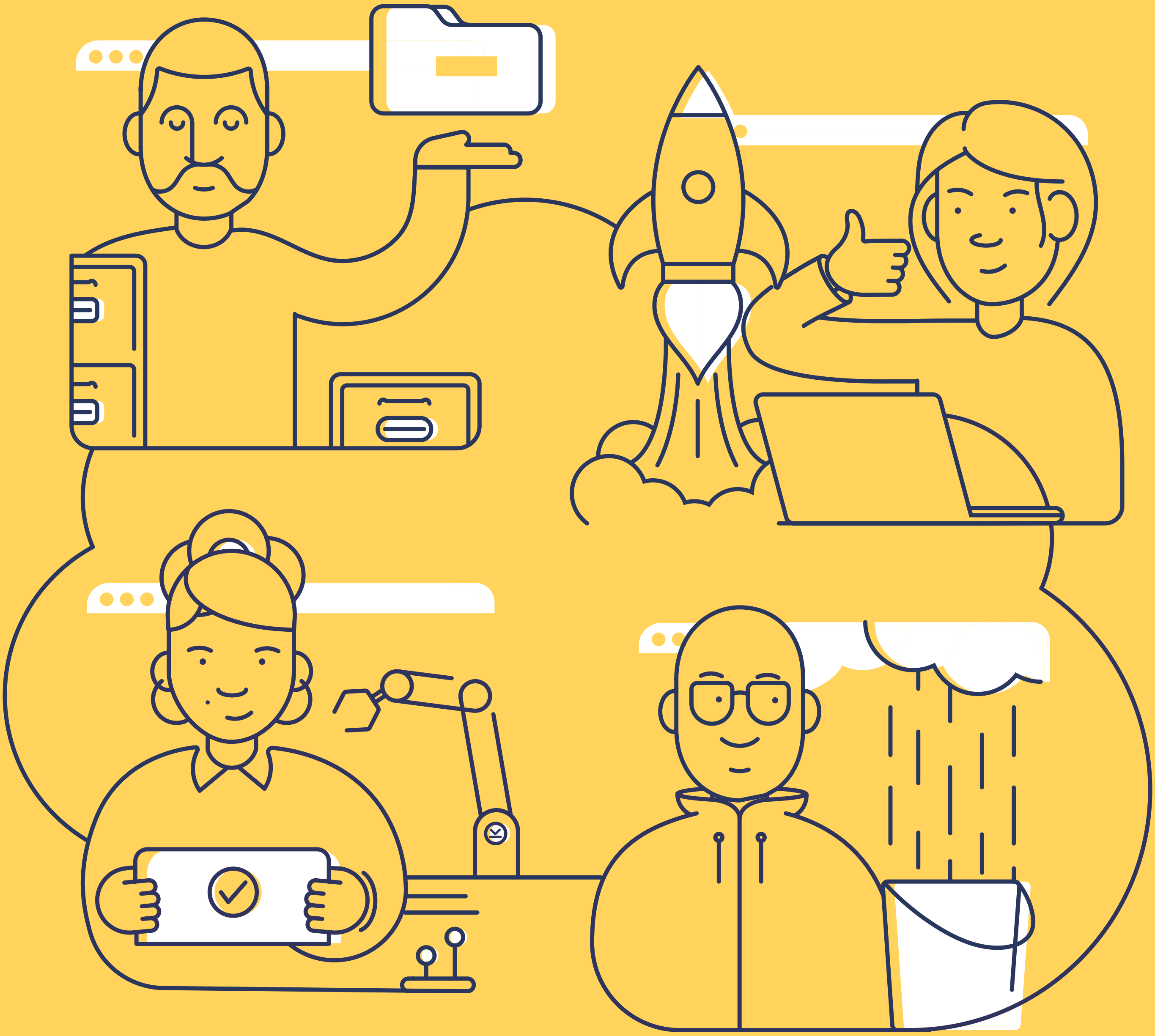
Gearset

CI/CD for Salesforce

Build an automated release pipeline for reliable deployments and powerful DevOps

Contents



Who is this ebook for?

If you're involved in building on Salesforce, this ebook is for you — whether you're an admin, developer, team lead, release manager, architect, or in any other role. Salesforce brings together declarative and programmatic developers, and DevOps has always been about breaking down silos. CI/CD isn't just for developers.

That said, successfully building an automated release pipeline for Salesforce involves a few stages, and CI/CD isn't the place to begin. So if your team is just beginning to look into DevOps, resources such as Gearset's whitepaper on version control would be a good place to start. Then come back to this ebook when you're ready!

What is CI/CD?

CI/CD is an integral part of DevOps. It involves practices and automation tools that support an agile approach to development, where incremental improvements are released more frequently. The aim of this approach is to increase the quality and pace of delivery.

As your release process matures, the number of environments and deployments you manage increases. The simplest development model for Salesforce is to create work directly in production. **That's fast but risky**. And it's only possible for declarative changes anyway, since Apex has to be deployed from another environment. So teams rightly move towards a mature development model with a release pipeline of multiple environments used for building, integrating and testing work. **That's safe but slow**.

Doing everything in the release process manually might be feasible to begin with, but becomes increasingly painful as your pipeline evolves, and at some point there's no alternative but to automate. With automation accelerating your workflow, CI/CD makes it possible to have the **best of both worlds: high release quality and velocity**.

What does CI/CD stand for?

‘CI/CD’ is often used as a catch-all term for release automation. But let’s be precise about what we mean by ‘CI’ and ‘CD’. There’s actually some ambiguity in the term: ‘CI’ always stands for continuous integration, but ‘CD’ might refer to continuous delivery or continuous deployment. It’s worth unpacking these three ideas and, since Salesforce is a unique platform, translating them into a Salesforce context.

Continuous integration

Continuous integration is the practice of frequently merging development work back into the main branch of Git, your version control system (VCS). Merging early and often confirms that your work can be successfully integrated with the rest of your code. The longer you hang on to development work in an org or Git branch, the more likely you'll hit a conflict when you try merging into main.

This practice goes hand in hand with automation tools that allow you to 'shift left' testing — that is, to test work in the earliest stages of your development and release cycle.

Merging work — or even opening a pull request — can trigger automated checks, such as:

Unit testing

Check your Apex classes execute as intended. It's also best practice to keep code coverage above 75% throughout your pipeline, as Salesforce enforces this threshold at the production stage.

Static code analysis

Find out which Apex classes need to be reworked or improved, helping you keep your team's code consistent and high-quality.

Salesforce deployment

Prove your work is deployable to a Salesforce org. It might all sit comfortably in your Git repository. But will a Salesforce org, which is essentially the compiler and runtime for your code, accept and build your changes?

Continuous integration gives you confidence that the work you're doing now isn't going to fail because of a mistake you made early on. Frequently merging work into the main branch also helps your teammates, as all of you are working from a codebase and source of truth that reflects the very latest development work. You'll be alerted to errors or merge conflicts sooner, be able to identify them more easily, and resolve them faster.

Continuous delivery

It's a fundamental aim of DevOps to deliver small, incremental changes to your end users more frequently, instead of waiting to release until the very end of a long project. Continuous delivery refers to this overarching objective: automating the promotion of work along your release pipeline to accelerate the flow of work and speed of delivery to end users.

This approach *doesn't* mean whisking work through the release pipeline at breakneck speed, only to discover it breaks everything in production. Automated testing and validation is an essential part of each automated deployment, meaning individual CI jobs function as gates to prevent any work that needs revision from progressing along the pipeline. In those moments, automation helps you fail faster. You discover problems much sooner, which means you can get to work on the fix straight away. So while no one hopes for a stalled CI job, ultimately it will speed up your team and help you achieve continuous delivery.

Continuous deployment

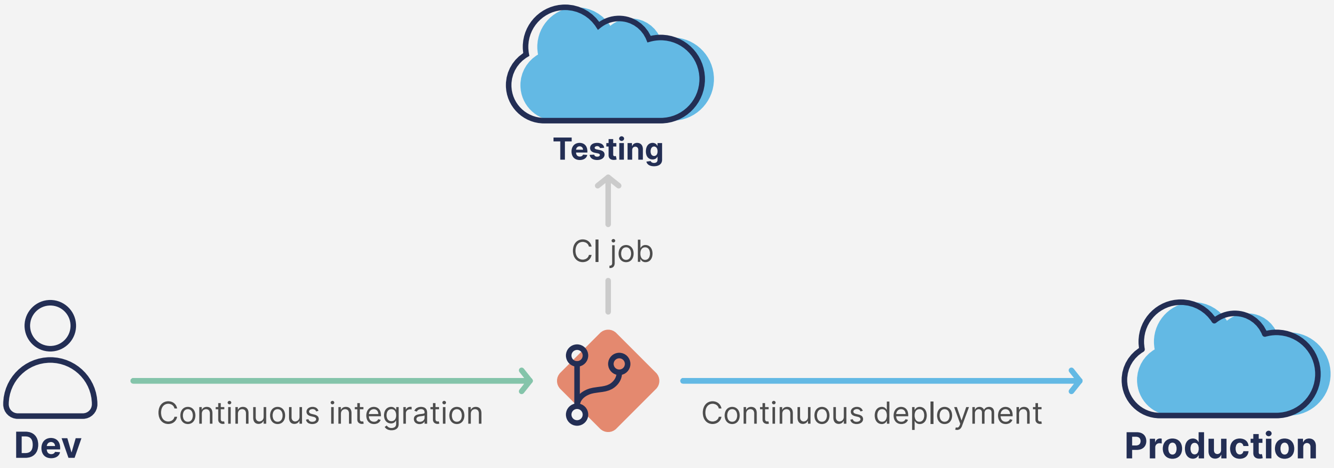
Sometimes 'CD' is used to mean continuous *deployment* instead of continuous *delivery*. Continuous deployment means automating the final release to production, so everything that passes your automated tests is also released automatically. This approach requires complete confidence in your tests and testing processes.

For most teams, the risks outweigh the potential time savings, and it's preferable for the final release to have human oversight. The final release package needn't be built manually, but having someone who does a last check and actually hits 'deploy' is recommended. This person is then in a good position to check the actual impact of the release on production, keeping operations running smoothly.

What we mean by CI/CD for Salesforce

In this ebook, ‘CI/CD’ refers to the practice of continuous integration and the automation tools that are required to achieve the aim of continuous delivery. We won’t use the term to imply a workflow with continuous deployment.

Continuous Delivery



82%

of Salesforce teams are working towards CI/CD

Source: State of Salesforce DevOps 2022

Why Salesforce teams want to master CI/CD

Most Salesforce teams want to master CI/CD. *The State of Salesforce DevOps 2022 report* shows that **51% of teams** have already begun to implement it, another **31% hoped to do so** this year, and just **18% have no plans** for CI/CD adoption. What's driving the demand for release automation?

The value of CI/CD

CI/CD is critical for DevOps maturity. In fact, most of the benefits teams have in mind when they begin their DevOps journey only materialize once CI/CD is in place.

DevOps is all about delivering changes more frequently, with less effort and fewer bugs. The aim is to release changes reliably and quickly as a routine part of the development life cycle.

Only with a reliable CI/CD pipeline can you unlock the benefits of DevOps:

Faster delivery of added value to end users

Tighter feedback loops

Higher productivity

Fewer bugs

Shorter time to recover

Better collaboration

Ever-increasing release reliability

Faster delivery of added value to end users

Accelerating your workflow with automation means faster delivery, boosting deployment frequency and — the thing your end users probably care most about — slashing lead times. If your team is struggling to clear a hefty backlog, CI/CD will help you reduce your release workload and get through more projects.

Tighter feedback loops for a user-driven development cycle

It's dispiriting to ship your work only to discover that it isn't *quite* what the end users had in mind, or that their requirements have already changed. CI/CD helps you release work in increments, so your end users can give feedback as you go and contribute to the shape of each iteration.

Fewer bugs in production and less risky releases

CI/CD also improves security and release quality. By taking full advantage of test automation and gating in your pipeline, most bugs will be caught and dealt with early on. And smaller release packages have a smaller testing surface, so more releases should sail through testing and QA without any issues.

Shorter time to recover from mistakes

When a bug or error inevitably still finds its way into production, following a CI/CD process means you'll find a resolution sooner. It's easier to pinpoint errors in a smaller release package, so you can identify what's wrong quickly. From here you can roll forward to the next release, releasing the hotfix within hours. If it's not clear what the issue is, smaller release packages are also more straightforward to roll back.

Better collaboration across multiple workstreams

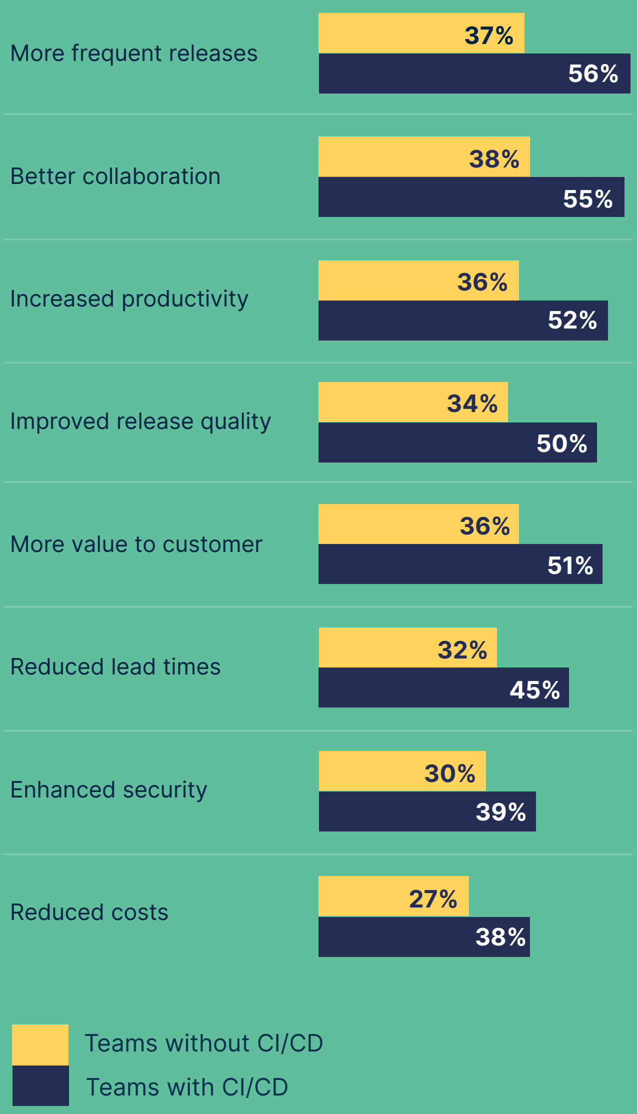
It’s hard to manage multiple teams working on several different projects. Adopting version control is a fundamental step towards resolving the challenge, but continuous integration represents the culture of collaboration that makes version control work effectively. Teams shouldn’t find themselves trying to release an entire project in one go and finding an eye-watering number of merge conflicts.

Ever-increasing release reliability

Releasing more often means your process is always being tested, tweaked, and improved, while you and your team learn which practices keep a free-flowing pipeline in action — and which cause blockages. All this takes the strain off your development team, and encourages a culture of continuous improvement — a key Agile principle.

Higher productivity and reduced costs

Making the most of automation saves your team from needing to tackle time-consuming deployment steps manually. CI/CD boosts productivity and reduces project costs, by freeing up your team to work on more valuable tasks.
In fact, all of the benefits of DevOps demonstrate why it's a more cost-efficient approach to development and release management.



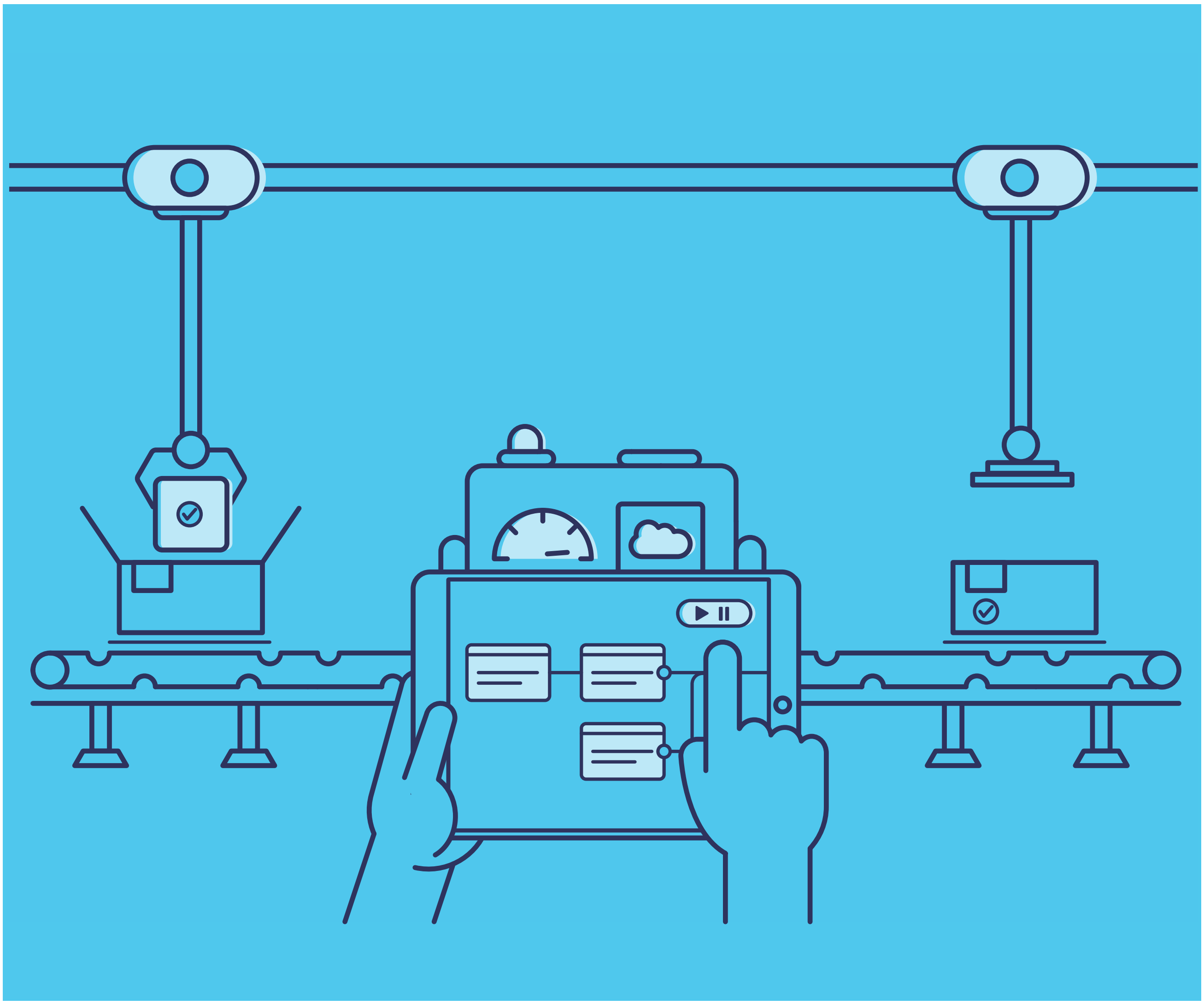
Teams with CI/CD report higher performance

Source: *The State of Salesforce DevOps 2022*

When to implement CI/CD

How do you know when your team and release process is ready for CI/CD? It's been proven that a 'big bang' approach to DevOps adoption, in which teams rapidly adopt a host of new practices all at once, is the least likely to succeed. For Salesforce teams, attempting to switch on CI jobs prematurely can be a source of frustration. In the worst case, the experience can even dent morale and put teams off trying again. It's important to wait for the right moment before implementing CI/CD, and there are two key things you need to have in place: **reliable deployments** and **version control.**

Deployment success



When you're wondering about implementing CI/CD, it's important to consider whether you're running repeatable and reliable deployments. If you try to automate before this point, you'll keep hitting barriers that stop you in your tracks and your CI jobs will repeatedly fail. This will force you to dissect your releases to find and fix bugs, creating even bigger deployment issues than you started with.

Once you make the switch to releasing more often and in smaller chunks, the lower the risk will be for each of your releases. Ramping up to CI/CD often involves manual effort to begin with — a bit like push starting a car. You'll need to begin doing the thing that CI/CD will eventually help with: releasing more often in smaller chunks. This way you can pinpoint the issues that often cause deployments to fail, fix those, and move to the point where you have a process that you can automate and step away.

Version control as a source of truth

CI/CD depends on version control. It *is* possible to automate deployments between orgs and run automated tests in the process, but that isn't CI/CD because it misses out the practice of continuously integrating work into version control.

Version control is foundational for a mature DevOps process and enables Salesforce teams to reap many benefits, including:

Multiple workstreams

Team members can work simultaneously in their own branches, allowing for parallel development without stepping on each other's toes.

Resilience

Since Git tracks the changing shape of your repository, you can roll back to earlier versions and revert merged branches.

Higher quality work

With all changes being reviewed in pull requests, teams can identify improvements and possible bugs before merging.

Visibility and auditing

Annotations on commits, pull requests and reviews produce an audit history of who made each change and why.

Alongside these benefits, version control is also valuable because it provides a single source of truth to which all development work is added and from which your team can release to production. If CI/CD is working properly, all the code in your main branch should be deployable — meaning you should always be in a position to release to production on demand.

According to the latest figures, 66% of Salesforce teams use Git. But nearly a quarter of those teams are still working towards making Git their source of truth, which is pivotal for implementing CI/CD. Until your team is deploying almost all development work through version control, **it's probably too early to attempt CI/CD in full**. Your Git repository needs to be the source of truth, with everyone developing from this single starting point.

The Maturity Matrix

Top-performing teams adopt DevOps tools and processes to make releases an asset, rather than an obstacle to Salesforce success. Gearset's maturity matrix helps you understand where you are in your DevOps journey and where you should go next.

Level 1: Novice

Novices still use change sets to deploy large, infrequent releases — or even make changes directly in production.

Level 2: Beginner

Beginners are just starting to adopt version control to get an understanding of its benefits and gradually create a repository of their metadata.

Level 3: Intermediate

Intermediate teams have made Git their source of truth, improving collaboration, code reviews and auditing.

Level 4: Advanced

Advanced teams are automating their release pipelines for CI/CD, helping them to make regular, reliable and small releases.

Level 5: Expert

Experts integrate automated backups into workflows, and can quickly restore data and metadata using familiar DevOps processes.

It's important to remember that DevOps maturity doesn't happen overnight. Teams that adopt each step in turn are **much more likely to succeed** than those who try to adopt everything at once.

How to implement CI/CD

Choosing the right tools

The first step to implementing a successful CI/CD process is finding the right tools. With so many on the market it can be hard to know where to start, but automation tools can be roughly split into two groups: **generic automation tools** that can be applied across different platforms, and **tools that are built specifically for Salesforce DevOps**.

CircleCI, Jenkins, TeamCity and Bamboo are the front runners of generic automation tooling. While cross-platform tools do give companies the flexibility to use the same toolchains across multiple platforms, it's worth noting that these options are more technical and require experience of DevOps and SFDX.

When setting up CI/CD for Salesforce with generic automation tools, you'll need to spend time:

Maintaining infrastructure

Writing and maintaining CI scripts

Configuring services
(that usually aren't Salesforce-friendly)

Salesforce teams typically invest around **90 hours** setting up a basic CI/CD workflow using a self-built pipeline.

Even once configured, teams that follow the generic tooling path often hit issues because of the Salesforce Metadata API — including difficulties deploying some types of metadata — which can lead to regular automation failures. The ongoing maintenance and troubleshooting of a pipeline configured with these kinds of tools can sink a lot of developer time.

The key advantages of using a Salesforce-specific CI/CD platform is that they're built to handle the peculiarities of the Metadata API, are often click-driven for an easier user experience, and don't require previous DevOps experience. Some Salesforce-specific platforms also offer additional functionality, such as automated testing, data backup, and change visibility for auditing. Using an all-in-one Salesforce DevOps platform means easier adoption for the team, simpler auditing, and less time spent bouncing between platforms.

When evaluating CI/CD tools for Salesforce, keep in mind the following considerations:

- Can you try out the tool yourself (with your own environments) before purchasing licenses?
- How much support would be needed to implement the tool?
- How quickly will you see a benefit from adopting the tool?
- Is the tool flexible enough to slot into your existing workflows?
- How well does the tool integrate with the other services you use?
- How much visibility does the tool give you?
- What additional functionality would the tool offer your team?
- How will you handle complexities such as environment variables?
- How much control over security does the tool give me?
- What do current users of the tool say about it?

Building a CI/CD process that works for your team

Your CI/CD process will be built around your Git branching strategy. Git is a flexible version control system, and there's no single branching strategy that works for every team. The most important thing is to *have* an agreed branching strategy. Beyond that, there are many factors to consider when deciding which branching strategy to go for. But the biggest question to answer is: how many persistent branches do you want to have?

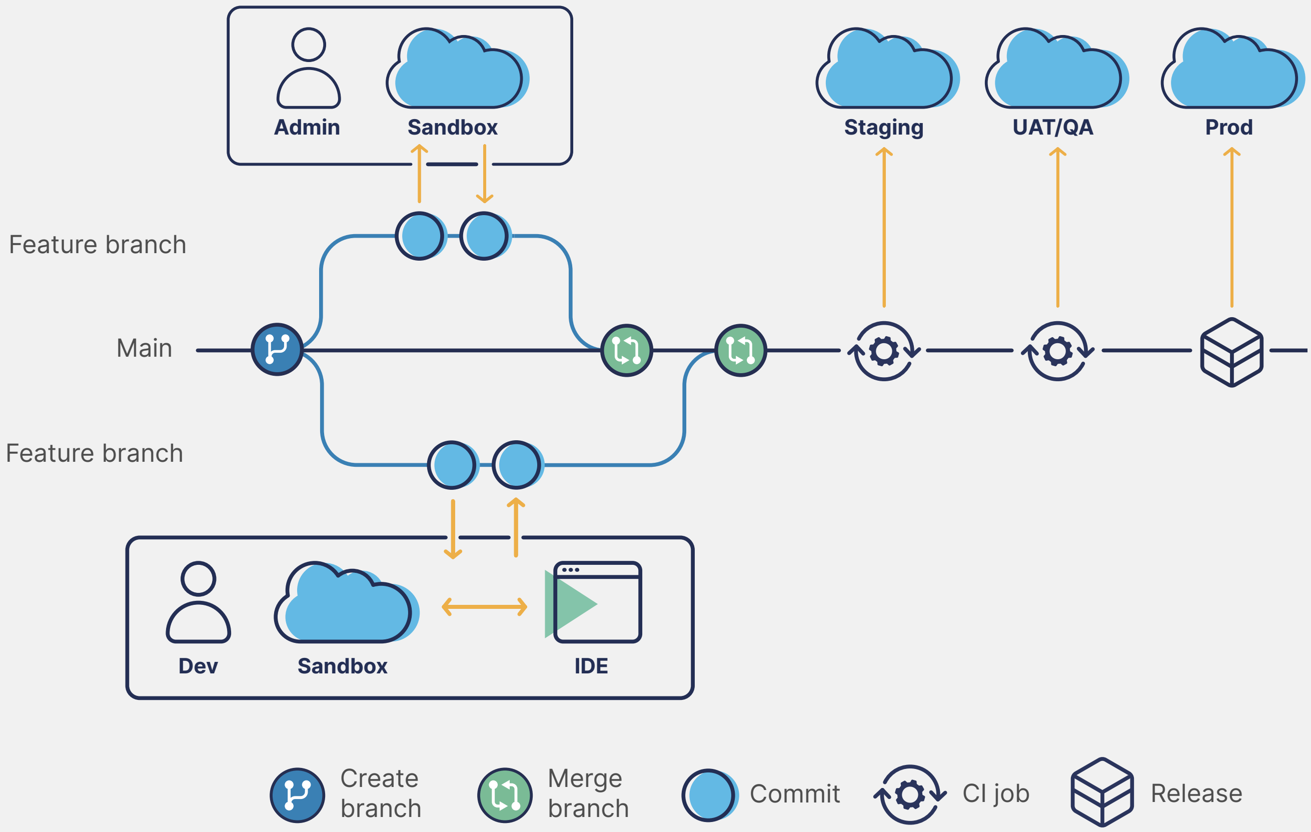
In a simple feature branch model, main is the only long-lived branch and feature branches get merged directly into main. This allows teams to work quickly and with minimal gating.

Other teams prefer the added security of intermediary branches, which are long-lived branches that a feature branch is merged into before reaching main. Often these branches will mirror the orgs that a team has in place.

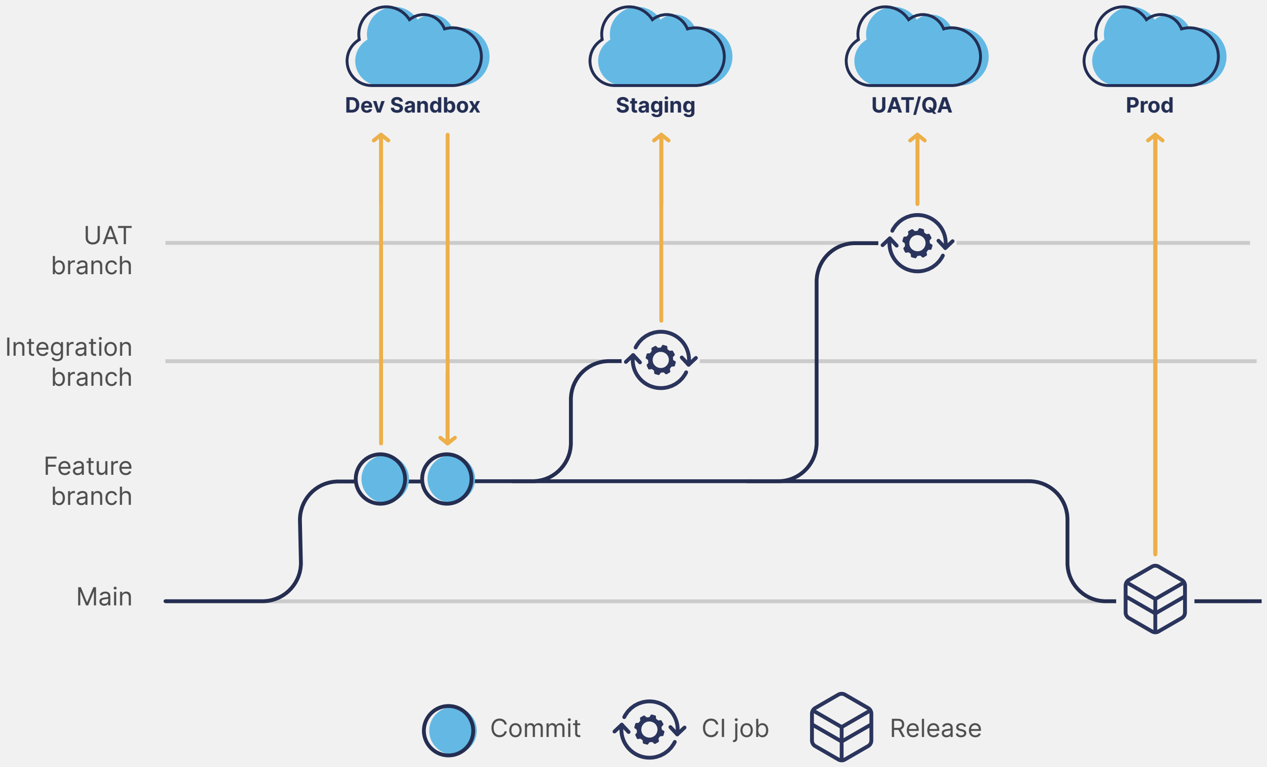
For example, if a team has separate orgs for QA and UAT, they may have long-lived QA and UAT branches that reflect each of those environments. While this means development will have been tested before reaching main, it does add additional work for the team merging multiple PRs.

Teams often choose to have intermediary branches because they're keen to make sure the metadata in main is always deployable. But with the right tooling, automatic validation can be configured to run as soon as a PR is opened against main, so nothing is merged that won't validate in production.

Feature branch model



Expanded branching model



Once you're up and running with your preferred branching strategy, **it's time to start looking at CI/CD**. The branching strategy your team implements will determine how you set up automation, but the underlying principle is getting development merged into a Git branch deployed to a Salesforce org quickly and without manual intervention.

If you use a simple feature branch model then your automation setup doesn't need to be complex: an automation build can simply push new development from main to your testing environments. On the other hand, if you have an **expanded branching model**, you'll need a **multi-staged automation setup** that **triggers deployments** when new development is merged to each branch.

Although automation will vary from team to team and there's no set picture of what 'good' looks like, there are two important things to bear in mind when configuring CI/CD:

1.

Don't set up automation for temporary branches

Teams have been known to create automation jobs that run whenever new development is committed to a feature branch. The idea is to get work back into an org at the earliest possible moment — so it's the **right aim but using the wrong approach**. There isn't really any continuous integration happening in this workflow. Automating from a feature branch suggests that development is lingering in a feature branch for a long time, which **isn't an agile way of working**, and increases the likelihood of **merge conflicts that would stall automation jobs**. Plus, a new automation job will need to be created for every feature branch.

2.

Continuous deployment isn't always the best option

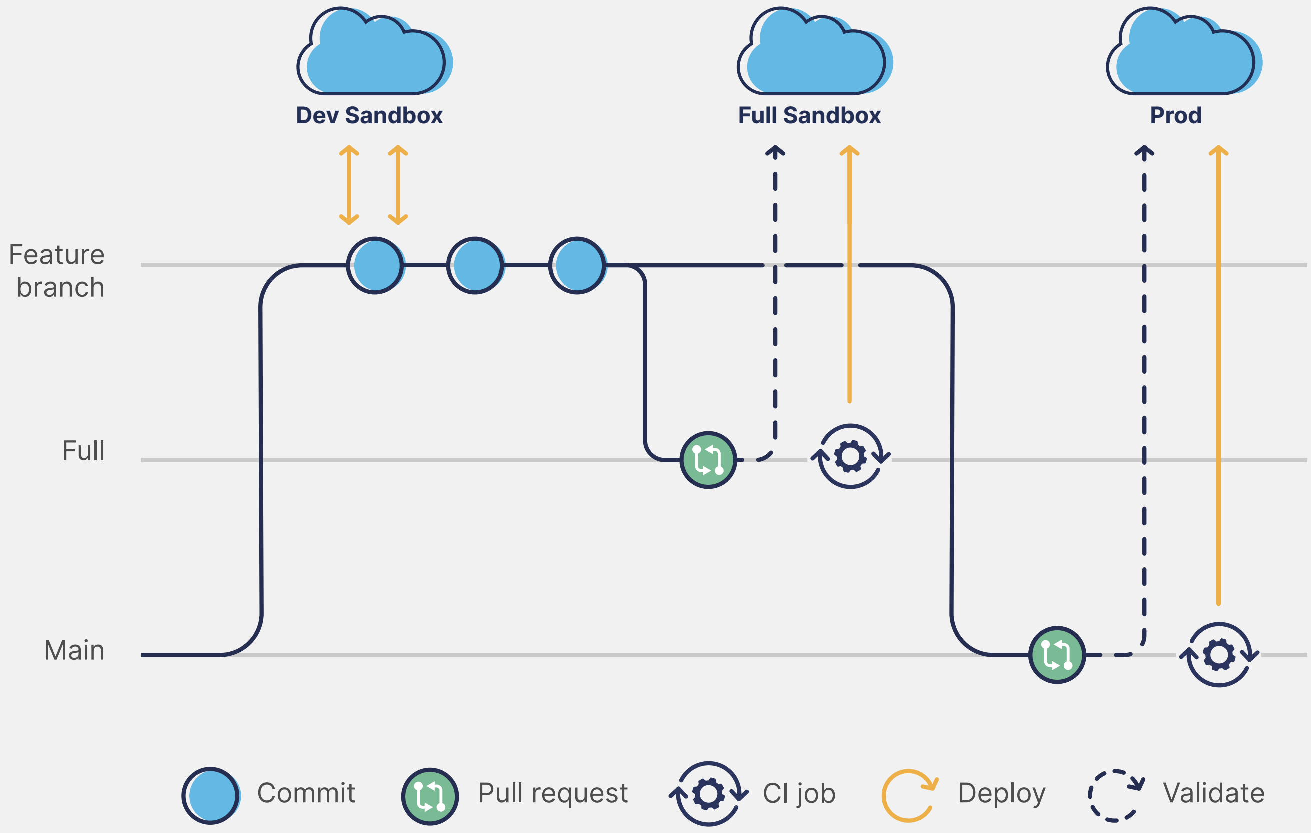
Teams with a very secure and resilient development pipeline, or with a high risk tolerance, may be confident enough to automate deployments to production. But for most teams it's best to have a final review of what's getting deployed to production, given that bugs in production can have big consequences for the wider business. With the right tools, it's possible to set up automation that will build a package for you. This package can then be deployed to production with **a single click**, which **streamlines releasing** while still **allowing for a final review** before a package goes live.

Intercom’s CI/CD workflow

The Salesforce team at Intercom have set up an effective and reliable CI/CD workflow, built around an expanded feature branching model.

Two Git branches, ‘full’ and ‘main’, correspond to Intercom’s full copy sandbox and production org. Feature branches are merged into ‘full’ for testing, then ‘main’, before being finally released to production.

To automate this workflow, Intercom have configured builds that automatically validate pull requests against the target org, and then automatically deploy all new, changed and deleted metadata when those pull requests are merged.



Adopting such a mature CI/CD process has enabled the Business Systems team at Intercom to ship at least 3–4 times a day. Automatically deploying to production isn’t for everyone, but the Intercom team is confident enough in their tests and validations to have implemented continuous deployment.

Ramping up over time

Once you've decided on your CI/CD tools and workflow, it can be tempting to dive straight in. But teams who try to introduce automation to their workflow in one 'big bang' tend to get stuck very quickly.

An org's metadata is usually complex — with dependencies between components and large permission sets — so automating delivery for all metadata types straight away can cause an array of errors. It's time consuming to unpick the amount of errors that a 'big bang' approach can cause, and this experience can put teams off their attempts to automate.

The most successful approach to introducing CI/CD to your workflow is to start with a subset of metadata. Automating delivery for a few metadata types gets the team comfortable with CI/CD while also reducing the likelihood of errors, and makes any errors that do crop up more manageable. Once your automation flow is running smoothly for that original subset of metadata, you can begin to add in more.

We recommend the following order for introducing metadata to your automation jobs:

1. Data tier

These are core components which most other customizations are built on top of. Starting with the data tier lays the metadata foundation for your automation flow. For example: Custom Objects, Custom Fields, Custom Apps, Value Sets, Static Resources, Content Assets and Picklists.

2. Programmability

Adding the custom code you've built on top of the platform is the next step, including Apex: Classes, Components, Tests, and Triggers.

3. Presentation

Then you can begin adding metadata that modifies how end users interact with the platform, e.g. VisualForce, Lightning Pages and Components, and Layouts.

4. Permissions and security model

Next up is the metadata that ensures all users have the correct access levels, such as Field Level Security, Profiles, Permission Sets, Security Settings, Roles, and Sharing Rules.

5. Other

Last but not least, any other required metadata types can be added in. The types of metadata left will vary from team to team, but it may include Email Templates, Reports, Static Resources, Flows, Workflows and Documents.

Creating a Salesforce CI/CD culture

You can have the right tools and process, but the people in your team are ultimately the key to successful CI/CD adoption. Adopting CI/CD tools without a shift in attitude and values is consistently reported as a blocker to DevOps maturity: more than half of respondents (54%) from the *State of Salesforce DevOps 2022 survey* reported deploying outside of their defined release pipeline some or most of the time.

But what is a DevOps culture? There are three traits that we typically see in high and elite performing teams:

1. Buy-in

It's important that every member of the team sees value in adopting a CI/CD process. Understanding the tools and processes is vital but keeping team members following a process is only possible if they see the value of working in this new way. Having even one member of the team who's confused or not bought in can undermine the work everyone else has put in.

Enforced change can disillusion employees, so the best way to get buy-in from the team is to take the journey to CI/CD and DevOps maturity one step at a time. Moving slowly means team members can see the positive impact of each process change, and are more likely to be enthusiastic about future changes as a result.

2. Collaboration

To get the most from CI/CD tools, it's important that team members are communicating with each other effectively. Everyone needs to be working within the same release framework, so they're on the same page about how projects are being delivered.

For example, a team member might be confused about the automation flow and deploy directly to a much later environment, such as pre-production, skipping out the automation flow through earlier environments like QA and UAT. If the deployment outside of the agreed CI/CD process conflicts with later development, it can easily start to break the automation flow and the audit trail is compromised.

Monitoring tools can be useful for catching work that's bypassed the CI/CD process. But the proper solution is cultural change in the team so no one wants to skip the process, and everyone proactively seeks support for parts of the process they don't understand.

3. Continuous improvement

Acknowledging processes that aren't working and being open to learning from each other enables a team to continuously improve their release processes. Incremental process improvements are the best way to keep achieving DevOps success. This should also avoid the need for larger, disruptive upgrades every few years.

Creating a blame-free environment is the best way to foster trust, so members of the team don't feel embarrassed about admitting things they don't know and discussing what's not working.

To really engrain a DevOps culture and ensure the success of CI/CD, people need to understand *why* they're expected to work in a new way. Without buy-in, collaboration, and continuous improvement being present across the entire team, new tools and processes can't deliver the full potential of Salesforce DevOps.

Overcoming common obstacles for CI/CD

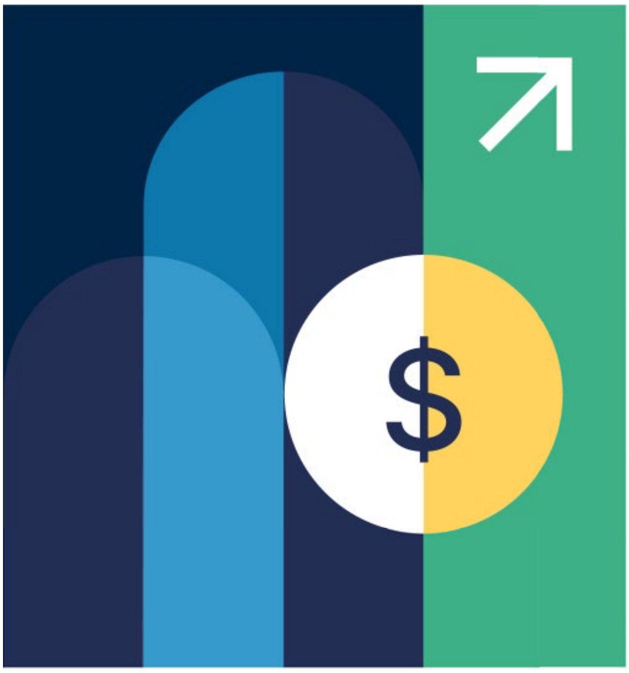
It’s a real achievement to build a CI/CD process that runs smoothly. While getting there doesn’t have to be a long and arduous journey, it’s fair to say plenty of teams hit snags along the way. So it’s worth identifying the most common obstacles for CI/CD adoption — and how to overcome them.

Getting to grips with new technical concepts

DevOps is still a relatively new way of working in the Salesforce ecosystem, so teams often struggle to visualize how CI/CD will work for them. They worry that less technical team members may still not understand version control systems, and particularly the importance of Git being the new source of truth for everyone. It's also common for teams to be nervous about upping their release cadence. If releases once a week or fortnight are challenging, how can it possibly help to move towards daily releases?

But to be successful, CI/CD needs the participation of whole Salesforce teams, and that makes adoption a significant step. It's not simply about automating a few steps in order to save a handful of developers some manual effort. It's a whole new way of working.

Solution: Educating admins and developers on version control and CI/CD is vital for DevOps adoption. Closing the skills gap will set your team on the way to DevOps success, and thankfully Salesforce professionals are typically enthusiastic about training. Our own training platform, DevOps Launchpad, includes product-agnostic courses on version control and CI/CD.



Convincing the business of CI/CD's value

DevOps transforms how Salesforce teams are able to support the wider business. But it takes time to mature into DevOps, and the benefits emerge more clearly over time. So in the early stages it can be a challenge to convince the business that it's a priority to invest in DevOps.

Solution: The key is translating the technical advantages of CI/CD into language that resonates with non-technical stakeholders, emphasizing the value to the business more than the quality-of-life improvements for the development team. CI/CD makes releases more efficient and so yields ROI. It also helps businesses — even large enterprises — be more agile, so they can adapt quickly and seize opportunities.

Once you've begun to see success, measuring the impact and calculating ROI will help to secure ongoing investment in DevOps. The DORA metrics are well established as an effective measure of DevOps performance.

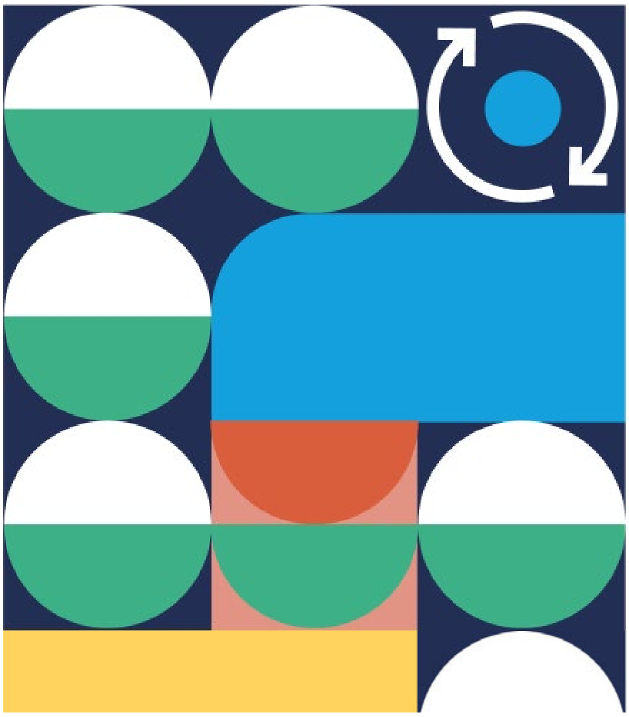
Unblocking stalled CI jobs

According to our research, 80% of Salesforce teams regularly struggle with stalled CI jobs. Given CI/CD aims to reduce manual effort, it feels self-defeating that so many automation jobs routinely need manual intervention.

Solution: Teams need to troubleshoot the problems that regularly cause their deployments to fail before attempting automation. And there are numerous culprits to look for:

- Certain metadata types can be tricky to deploy, and most deployment tools don't handle those quirks automatically.
- If your environments are out of sync, CI jobs will likely fail because they're trying to deploy a lot more than just the latest changes. Delta CI jobs are a nifty and popular solution, but getting environments back in sync is the ideal.
- There are some changes that will cause a conflict in Salesforce and so won't be deployable, but Git will happily merge them because they don't touch the same files.
- Sometimes there are dependencies between different feature branches and merging only one of them will cause the CI job to fail.

The most effective way to catch problems early is to validate that Salesforce can receive changes before you deploy or merge them.



Identifying reliable tools

These obstacles are worth bearing in mind when it comes to choosing the right solution or toolset for your team. Carefully researching your options is pivotal to your DevOps implementation, because finding the best solution for your team often makes the difference between success or failure.

Some teams go through the whole process of implementing CI/CD, only to find their chosen tool doesn't live up to their expectations and the vendor's support is lacking. Teams also need to find a solution that feels intuitive for their whole team and will support a scaling DevOps process.

Since DIY and open-source toolchains require a significant investment upfront and then ongoing maintenance, most teams opt for a Salesforce CI/CD solution that works out of the box. Gearset's DevOps solution has been designed specifically for Salesforce and keeps pace with the Salesforce platform as it evolves. Gearset helps you build a CI/CD process that perfectly suits your team's workflow and your business's requirements, overcoming all the obstacles in your way.

80%

of Salesforce teams routinely need to fix stalled CI jobs

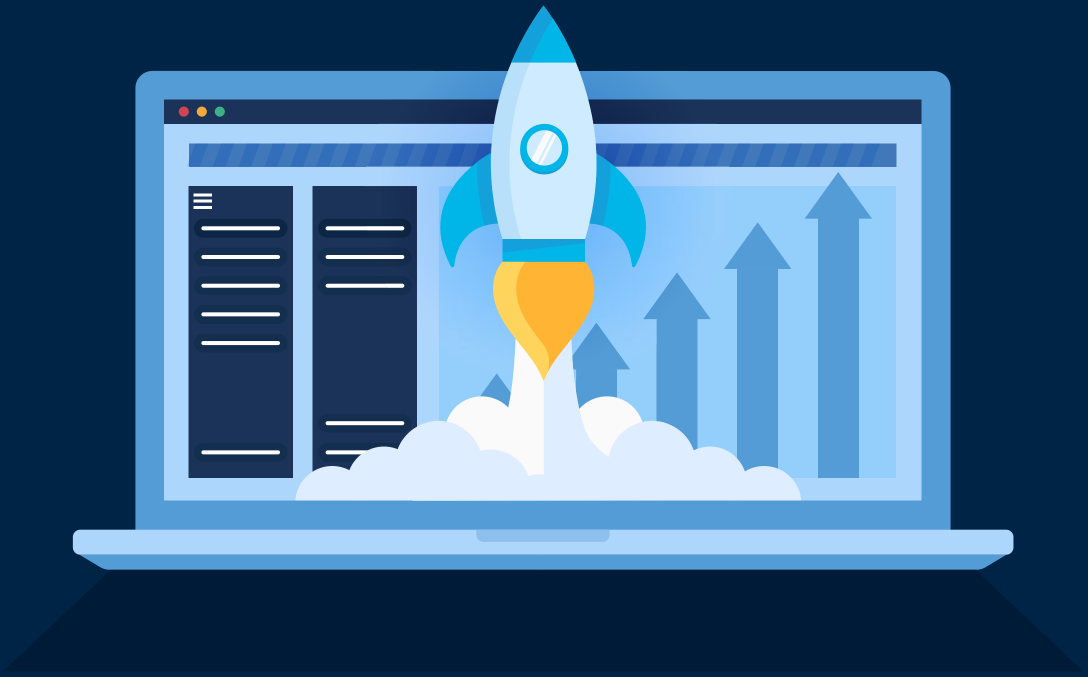
Source: State of Salesforce DevOps 2022

Gearset's CI/CD solution

To make a success of CI/CD for Salesforce, we've found teams need a solution that ticks all these boxes:

- ✓ **Runs reliably** despite the unique deployment challenges on Salesforce
- ✓ Flexes to **match your workflow**, not vice versa
- ✓ Builds on the **best Salesforce has to offer**, including DX
- ✓ Feels **comfortable for the whole team**, whether they're pro-code and no-code
- ✓ Helps everyone **visualize** the DevOps process
- ✓ Makes release management **easy for any setup**
- ✓ **Scales** with growing complexity and team size
- ✓ Fits with existing tech stacks

Gearset's complete DevOps platform has been developed to fit the bill, offering **built-in CI/CD automation tools.**



Deployment reliability

Automation can't work without reliability, but for many Salesforce teams deployments are anything but reliable. We all know there are plenty of quirks and challenges on the Salesforce platform that cause deployments to fail, with only 50% of change set deployments running successfully. A lack of reliability is the fundamental challenge for release management on Salesforce.

Gearset is unique among DevOps platforms for tackling deployment success head on, and we do this in two ways.

Firstly, deployments with Gearset are always based on a comparison of the metadata in the source and target environments — whether they're orgs or Git branches. In a manual deployment, this means you can see the exact differences, filter down to the changes you want to deploy, and understand exactly how that will impact the target environment before you deploy.

Secondly, Gearset has dozens of problem analyzers that find and fix common problems with deployment packages that cause them to fail. For example, if your deployment package is missing a dependency, Gearset will help you add in the missing item. Thanks to these problem analyzers, Gearset's users have an average deployment success rate of 93%.

Gearset's reliability as a CI/CD solution is possible because these comparisons and problem analyzers run automatically for every CI job.

93%

average deployment success rate of Gearset's users

Source: State of Salesforce DevOps 2022

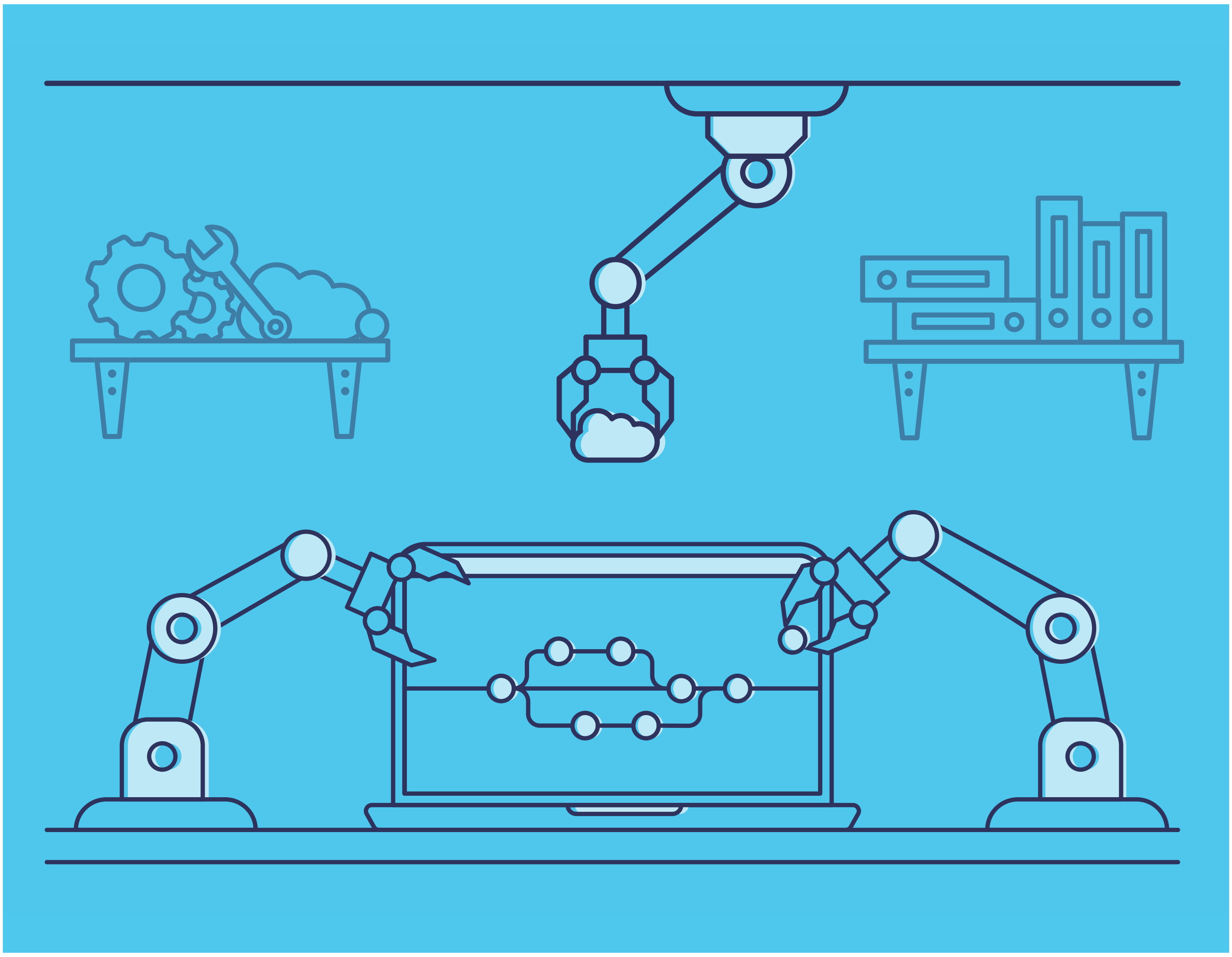
CI jobs

It's quick and easy to set up individual CI jobs between any two environments. For true CI/CD, the source of each CI job will be a Git branch and the target a Salesforce org. Taking full advantage of Salesforce DX, some teams use scratch orgs as the target environments for CI jobs, and these temporary orgs can be automatically created by Gearset.

You have full control over the configuration of each CI job:

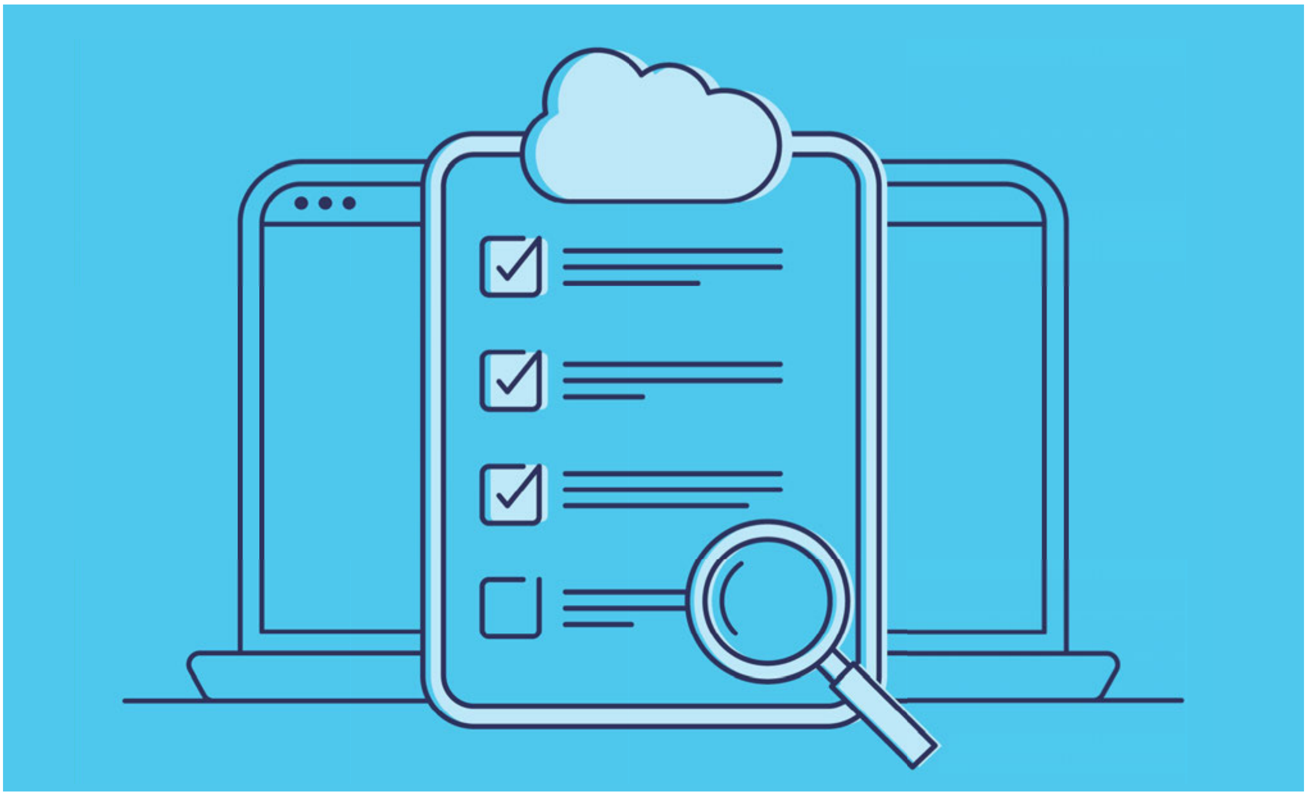
- What triggers the job? Or what schedule does it run on?
- What metadata types are deployed?
- Which differences should be deployed? For example, just changed and new metadata? Or deleted metadata as well?
- Which environment variables should be preserved?
- Should validations and tests be run?
- When and where will notifications be sent?
- Should the CI job trigger something outside of Gearset?
- Which problem analyzers are applied?
- What static code analysis rule set is applied?

A full history is kept for each CI job, with all details and reports recorded. You can roll back any deployment in the CI job history.



Delta CI jobs

Ideally, CI jobs run between environments that are kept in sync, but many teams aren't starting from that position. For example, the metadata in their main branch may not closely reflect the metadata in their UAT or QA org. In such cases, delta CI jobs are a nifty tool. As the name suggests, they only deploy the delta — that is, only the latest changes that have been made. A delta CI job just deploys the changes in any new commits, and so these are sometimes called 'commit-by-commit' deployments.

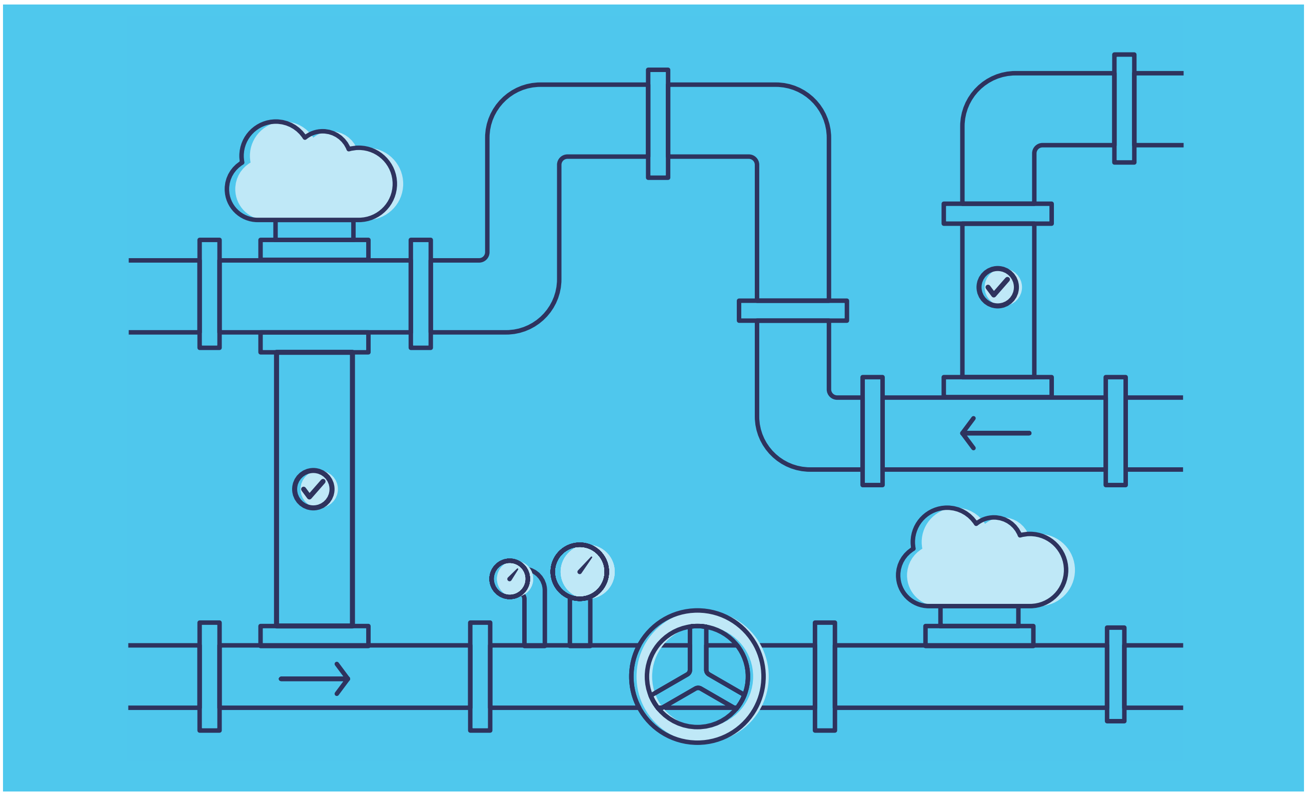


Salesforce validations

Salesforce validations let you know whether or not metadata is deployable. This Salesforce functionality is tightly integrated within Gearset and is extremely useful in some CI/CD setups, helping you identify issues early on and prevent blockages in your release pipeline.

Let's say you want to automate deployments of changes from your main branch to a testing environment such as QA or UAT. It's extremely useful to find out if the changes in your feature branch are deployable *before* you merge, saving you from merging undeployable changes into main and blocking the release pipeline for everyone else. With Gearset, you can automate validations of every pull request against the target org of your CI job.

You can also set up validation-only CI jobs in Gearset, which tell you if your deployment will be successful, without needing to fully deploy your changes. It's great to confirm that your metadata in Git is deployable to a Salesforce org, such as UAT. But ultimately the org that really matters is production. So you might want a validation-only job that automatically carries out a validate-only deployment to production. That way you can test changes in UAT, already knowing that they're deployable to production. Or you'll be able to catch any problematic changes early and begin working on the fix right away. When release time comes, there won't be any stress or scrambling around troubleshooting errors. You'll be confident of deployment success, and that final step is faster because you have a validated package ready to release.



Pipelines

CI jobs offer powerful automation for a stage in your release pipeline. But most CI/CD setups will need multiple CI jobs. It's essential to get visibility of your entire release pipeline from end to end. And ideally you want to manage all your automation in one place.

Gearset's Pipelines interface gives your whole team a birds-eye view of your whole setup. It shows you the current state of all the environments in your release pipeline and how changes are deployed through them. Gearset automatically detects changes in your environments and automatically creates pull requests for you, so you can promote changes downstream through to release and back-propagate changes into upstream environments that don't currently contain them — all in a matter of clicks.

Using the Pipelines interface promotes DevOps by making your release process more transparent and collaborative.

It also removes a lot of the friction that can stop teams maturing their DevOps processes and ultimately reaching their business goals. In particular, Gearset's semantic merge technology means it can handle most merge conflicts for you, where Git is none the wiser. You can reclaim the time you'd normally waste on manually resolving conflicts.

Your Pipelines dashboard in Gearset functions as the command center and source of truth for your Salesforce release process — even if only some of your team use Gearset's DevOps platform. For instance, if you have developers who only work in the CLI and GitHub, Gearset will still track their pull requests and help you manage their development work. And everyone can pop over to Gearset to easily visualize the whole workflow, even if it's not where they spend most of their time working.

The main features of pipelines include:

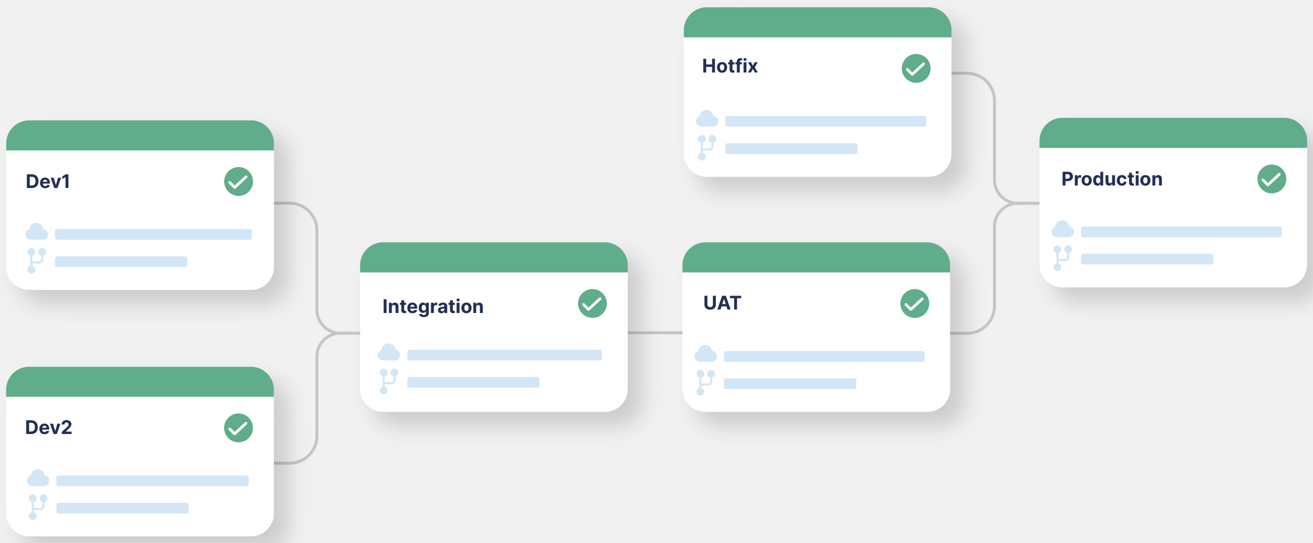
Visualization
Get an overview of your entire release pipeline, with quick visibility of where your changes are in your release process, what needs to happen to move them forwards, and how much work you have in flight.

In-app merging
Merge branches painlessly thanks to our purpose-built semantic merge algorithm that resolves conflicts.

Propagating changes
Keep all your environments in sync by making sure all changes are deployed to other environments in the release pipeline.

Tight version control integration
Whether you're working within Gearset or in your version control system, see the same changes and status, allowing your entire team to work together seamlessly.

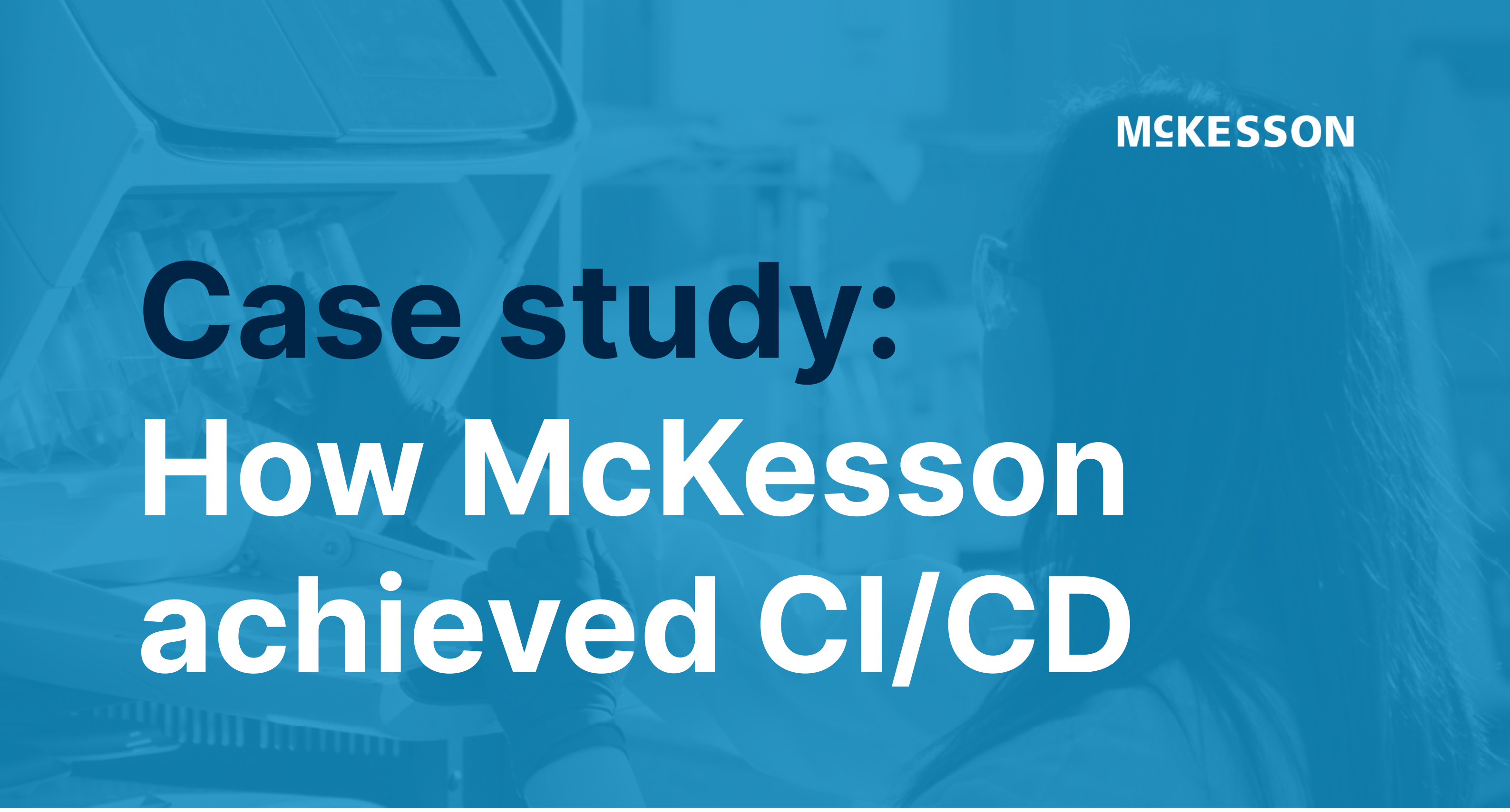
Ticketing integrations
Automatically update associated tickets on Jira, Asana or Azure DevOps when a work item gets into each successive environment in your pipeline.



12 + million

deployments using Gearset

gearset.com/trust



Case study: How McKesson achieved CI/CD

McKesson is one of the oldest and largest healthcare companies in the US. After implementing Salesforce several years ago, it soon became clear that the first-party deployment tools didn't provide the power or scalability McKesson needed.

"Deploying or preparing for deployment was taking anywhere between 6–12 hours each on average, and it was just painful," says DevOps Architect Matt Avitable. "A couple of 3am nights too many and we hit a tipping point."

To turn things around, McKesson adopted Gearset as a DevOps platform.

"Many tools on the market weren't intuitive or required that you modified your Salesforce instance for installations. The simplicity of Gearset was a big selling point for me," Matt explains.

Within months, McKesson were running an automated pipeline and had streamlined their entire release process, bringing developers and admins together. Monica Thornton-Hill, then Salesforce Admin at McKesson and now a Senior Technical Consultant, recalls the change: "Before, I was terrified to even touch anything that had something to do with a deployment — I couldn't imagine ever running one! But **with Gearset, I felt comfortable deploying after just one or two walkthroughs**."

Over the years, McKesson has continuously improved its DevOps process, iterating on its CI/CD workflows for Salesforce. Jason Gerstenberge, Principal Architect at McKesson, worked with Gearset to introduce version control and build a reliable pipeline around the feature branch model. Now, McKesson are evolving their approach again to make the most of Gearset's pipelines interface.

62,133
Comparisons

9,762
Deployments

27,143
CI job runs

"Gearset really changed the game for us in terms of being able to keep environments in sync, move metadata between sandboxes, and make the deployment process predictable."

Jason Gerstenberger
Principal Architect, McKesson

"With Gearset we have a fast, automated deployment pipeline which pushes changes into production throughout the week. My life has definitely gotten easier!"

Matt Avitable
DevOps Architect, McKesson

What next?

If you're looking to implement CI/CD or want to replace an existing setup that's unreliable, get in touch to arrange a consultation with us. We'll ask about your specific workflows and challenges, then provide tailored advice about how Gearset can help.

Book a consultation

You can also try Gearset now by starting a free 30-day trial. You don't need to install anything in your orgs or hand over any credit card details. See for yourself how Gearset helps with deployments, and try building your own CI/CD pipeline!

Start free trial

To learn more about CI/CD, and Salesforce DevOps in general, head over to DevOps Launchpad for free online training and certification. Join thousands of Salesforce professionals getting ahead of the curve by skilling up in all things DevOps.

Start your learning journey

About Gearset

Gearset is the complete Salesforce DevOps solution, with powerful tools for unparalleled deployment success, continuous delivery, automated testing and backups. Thousands of Salesforce professionals have already used Gearset's cloud-based app to run millions of deployments, back up billions of records, and save billions of dollars through productivity improvements.

Founded in 2015 by DevOps experts, Gearset is designed to help every Salesforce team apply DevOps best practices to their development and release process, so they can rapidly and securely deliver higher-quality projects. With inbuilt intelligence that solves the fundamental challenges of Salesforce DevOps, Gearset is a uniquely reliable solution trusted by more than 1000 companies, including McKesson, Accenture and IBM.

W: gearset.com
E: team@gearset.com
T: +1 (833) 441 7687