circleci

How to be a CI/CD Engineer

A Professional Guide

from Principles and Processes to Pipelines

By Angel Rivera

What is a CI/CD Engineer?

While the work of CI/CD Engineers isn't new (CI/CD has been practiced for over a decade now), there is increasing attention being paid to the role and its impact within organizations. In 2014, Twitter formed its "engineering effectiveness" organization. Google has a massive "engineering productivity" team. Cloud-native companies were among the first to embrace the role of CI/CD Engineers, but many other companies are now following in the giants' footsteps.

For some smaller companies, the role emerges organically. Maybe it's an informal responsibility shared by the entire team. Or perhaps one person sees an issue with a pipeline and steps up to fix it. Though people in this function often go by different titles (CI/CD Engineer, DevOps Engineer, Dev Experience Engineer, etc.) it's clear that this role has outsize impact on a team.

The world is increasingly built on code; over the next decade, the number of software developers is expected to grow by 75%. Without a dedicated CI/CD leader, effectiveness and scale will be challenging.

If you are a CI/CD Engineer, or you want to be, then this is the ebook for you. I'll walk you through the many responsibilities and skill sets a CI/CD Engineer can own on a team as well as their opportunities for creating massive efficiency and impact for the engineering organization they are a part of.

We'll look at:

1. Duties of a CI/CD Engineer
2. Characteristics and strengths common among CI/CD Engineers
3. Emerging patterns in the industry and opportunities for impact
4. Performance benchmarks for CI/CD Engineers to lead their teams by

Duties of a CI/CD Engineer

While this role varies in specific duties and responsibilities by company, CI/CD Engineers share some duties in common which are key to the role. These are:

1. Develop CI/CD principles
2. Review and modify CI/CD principles, iteratively
3. Maintain CI/CD tools/platforms (if applicable)
4. Develop and maintain pipeline configurations
5. Automate processes

Develop CI/CD principles

Adoption of CI/CD principles produces huge gains in efficiencies. These practices are composed of processes that are defined by stakeholders and facilitate optimized software development. A CI/CD Engineer is an authority on everything CI/CD. As an authority, they are actively involved in the development and implementation of these related processes.

Review and modify CI/CD principles, iteratively

Once practices are established, it's critical that teams continually revisit and optimize them regularly so that they support current operations. CI/CD Engineers are in a unique position to develop a deep understanding of how respective teams are operating and interacting with the other units. They have insight into what's working and what's not with the added ability to determine and offer corrective actions. Iterative reviews provide consistent visibility into the processes that drive software development and surface options for improvement.

Manage CI/CD tools/platforms (if applicable)

The next step after establishing and adopting CI/CD practices and principles is leveraging CI/CD tooling and platforms to facilitate the execution of related processes. CI/CD platforms bring these processes to life and provide the automation that executes specified pipelines. Implementing and managing CI/CD tooling requires a combination of skills (this is DevOps at its best), but I suggest that in this aspect a bit more emphasis on Ops is appropriate. I say this because ensuring that CI/CD tooling operates consistently definitely falls under the "operations" wheelhouse. This particular duty is also dependent on architecture and operational landscapes.

In my experience, organizations operate their CI/CD architectures in 2 different modes:

Managed CI/CD:

> The CI/CD architecture is provided by a 3rd party vendor such as CircleCI. These services provision, manage, and scale the underlying CI/CD infrastructure with very little effort from technical teams. These services are designed to directly serve developers and strip away all of the hard parts of CI/CD. Engineering teams are empowered by these tools to develop software and not to be bothered with the resource-intensive and time-consuming management of CI/CD infrastructure.

Self-hosted/on-prem CI/CD:

> Managed CI/CD is an excellent option for many organizations, but due to circumstances such as data laws and government regulations, self-hosted platforms are still required for certain industries. Self-hosted CI/CD requires teams to provision and manage all aspects of their CI/CD infrastructure. Teams manage these infrastructures holistically which means they have to design, implement, and manage connectivity, servers/nodes, operating systems, and the chosen CI/CD tooling that must run securely on this elaborate infrastructure. This requires lots of time and effort on behalf of DevOps teams and is a more complex operation.
>
> Regardless of CI/CD architecture, achieving efficiencies in CI/CD is critical. Once the tooling is operational, it must remain that way or risk a decline in performance. Of course, there are many CI/CD tools available, and selecting the most appropriate one is no small feat. I advise vetting multiple solutions and choosing the tooling that best serves your team's needs.
>
> CI/CD Engineers could be responsible for some or all aspects of these CI/CD infrastructures. Ideally, this is a shared duty between CI/CD Engineers and DevOps Engineers. CI/CD Engineers manage the CI/CD tooling services and DevOps Engineers manage the underlying architectures. At the very least, the CI/CD Engineer must have deep knowledge of the CI/CD platforms they're supporting, including the underlying host infrastructure.

Develop and maintain pipeline configurations

All CI/CD tooling requires a form of pipeline configuration, which is the mechanism that specifies the various steps and segments to execute within your CI/CD processes. Configuring pipelines is completed in different ways depending on the CI/CD tool being used. Some tools make use of a Graphical User Interface (GUI) to configure pipelines, while others require that pipelines are specified in code. In all cases, CI/CD pipelines must be specified and maintained, and this duty falls squarely on the CI/CD Engineer.

Developing, managing, and executing CI/CD pipelines is the biggest responsibility that this role owns. These pipeline configurations orchestrate the execution of the steps specified. In other words, pipeline configurations are how we program the CI/CD tools to do what we want.

CI/CD Engineers ensure that pipelines are properly defined and performing optimally. They have a deep understanding of the pipeline's goals and the transactions occurring within them. They interface with DevOps teams to coordinate and optimize the steps that execute within the pipeline. The CI/CD Engineer documents the pipelines so that all of the interested parties are aware of how their software is built, tested, secured, and deployed.

Automate processes

Automation is at the core of CI/CD. It facilitates processing pipeline commands, inter-platform connectivity, and valuable integrations with 3rd party services. If you peel back the layers, at the core, CI/CD Engineers design and manage the automation that powers continuous integration and delivery operations. The level of experience can vary, but a foundational knowledge of automation is required to be an effective CI/CD Engineer.

Characteristics of a CI/CD Engineer

I've watched the emergence of this role and spoken to hundreds of engineers doing this work, and several patterns have become clear among the most successful CI/CD Engineers. These shared traits include:

1. Strong communication skills
2. Keen analytical skills
3. Ability to decompose complex processes into understandable components
4. Proficiency in automating and optimizing processes
5. Competent in team building and team communication strategies

I'll go into more detail on how each of these skills contributes to being a top tier CI/CD Engineer.

Strong communication skills

This role converges on multiple verticals, which requires individuals to competently communicate and interact with various teams. I can't stress this enough: the ability to clearly communicate is critical for this role.

Keen analytical skills

Much of the work in this role requires an individual to gain a deep understanding of a variety of concepts and how they relate to and impact the relevant domains. Being able to accurately assess respective landscapes is a vital step in improving inefficient processes.

Ability to decompose complex processes into understandable components

Breaking down complex topics into digestible components is a great trait to possess in many roles, but it's especially important in this context. Someone who combines this ability with keen analytical skills can help teams understand and capture their current operational states and existing deficiencies. Having an accurate view of operational landscapes enables innovative solutions for achieving and maintaining desired states.

Proficiency in automating and optimizing processes

Identifying deficient steps enables teams to expose pain points and bottlenecks that diminish the effectiveness of processes. Identification must happen before teams can address them with viable solutions. For instance, imagine a process that executes sequentially, meaning one step needs to complete before another can begin. This behavior is often called blocking. It's my experience that most sequential processes have steps that can be executed in parallel, which removes this blocking. Being able to devise and coordinate these optimizations is especially beneficial in this role.

Competent in team building and communication strategies

This characteristic is based on a collective of the traits I've already mentioned, with the addition of developing and maintaining credibility within teams and organizations. Credibility is the quality of being trusted and believed in. People are way more comfortable buying into and supporting decisions when they're confident in the individuals championing the efforts. The goal here is to develop consensus around decisions and to execute plans that benefit all of the stakeholders.

Emerging Patterns

Opportunities for CI/CD Engineers to Add Value

Over the past few years, concepts and patterns related to DevOps and continuous delivery (CD) have become more widely accepted in the industry. It's easy to see why: competition has increased, and it's clear that software is a key differentiator for businesses. Being able to optimize all aspects of software delivery, from collaboration to deployment and operation, is more important than ever.

The concepts that I see growing in popularity center around security, pipeline optimizations, and how to use healthy CI/CD benchmarks.

→ Security

→ Pipeline Optimizations

→ Performance Benchmarks

Security & DevSecOps for CI/CD Engineers

I suggested earlier that security-related duties should be an element of the CI/CD Engineer's role. With the increased adoption of DevSecOps and more developer-friendly tools, security responsibilities and processes are beginning to "shift left". This essentially means that developers are able to confidently incorporate security-related activities into the initial stages of their CI/CD processes, hence shifting security considerations left (toward the beginning of these processes) rather than right (towards the end).

A CI/CD Engineer can establish strong communication channels between security teams and those responsible for compliance/regulatory requirements.

These communication channels generate constructive collaborations between developers and security teams that keep everyone well-informed about critical requirements. These collaborations also facilitate adoption of DevSecOps principles, ensuring developers are knowledgeable and invested in new or unfamiliar security practices.

I've identified some security practices that should occur during the "shift left," or initial segments, of a CI/CD pipeline. CI/CD Engineers are in a good position to define these practices. The "shift left" process streamlines required security tasks and provides valuable feedback loops, preventing wasted time and resources that occur when you continue to process downrange pipeline segments that will ultimately end in failure. It's much better to catch and fix issues earlier in the pipeline than later.

Here are some security practices that can be defined and maintained by a CI/CD Engineer:

- Vulnerability scans
  Probe applications for security weaknesses that could expose them to attacks
- Container image scans
  Analyze the contents and the build process of a container image in order to detect security issues, vulnerabilities, or deficient practices
- Regulatory/compliance scans
  Assess adherence to specific compliance requirements

There are many other industry standard security practices that are usually implemented in release processes. The examples I mentioned earlier are just some of the tasks that a CI/CD Engineer can implement and manage in collaboration with security and DevOps teams. This kind of collaboration ensures all security and compliance requirements are automated and consistently applied within CI/CD pipeline segments.

Pipeline optimizations for CI/CD Engineers

I've seen a lot of teams treat their CI/CD pipelines in a "set it and forget it" manner. This is unfortunate. For whatever reason, **teams invest lots of time and effort in automating their software development and release processes, only to abandon work on them after implementation.** This is mostly driven by fear of disturbing or breaking something that is critical to releasing software.

The reality is, CI/CD pipelines are meant to represent software development and release processes. That means they must be continually monitored, assessed, and adjusted to make sure they are not only accurately automating your software development and release processes, but doing so efficiently. These pipelines must be regularly revisited and tweaked, and that can be an overwhelming task for individuals that aren't knowledgeable about all of the pipeline segments.

This is where the expertise of a CI/CD Engineer can add immense value. As administrator of the CI/CD pipeline, they can ensure the pipelines are not neglected and keep functioning efficiently. Some of the best ways CI/CD Engineers can add value by properly maintaining and optimizing pipelines in collaboration with appropriate teams include:

- Pipeline reusability
- Properly sized compute/resource nodes
- Parallelism
- Performance benchmarks for CI/CD Engineers

Pipeline reusability

Many of the pain points I've experienced and often discuss with others revolve around managing the pipeline configurations in CI/CD tooling. These configurations define pipelines and serve as the execution code for the automation on CI/CD platforms. They are often expressed in YAML, domain specific languages (DSL), or some other similar variant.

The syntax in these configuration files are generally limited in capabilities, especially in regard to reusability. This mainly stems from the fact that syntax, like YAML, is a declarative data structure and not a programming language.

Restrictive code reuse due to configuration syntax can be overcome, but it requires extra effort, such as building execution scripts that can represent pipeline segments while simultaneously encapsulating functionality. These configuration reusability issues are common in most CI/CD platforms and generally have the same impact: they are very difficult to maintain.

Within CircleCI, this config reusability challenge is solved with configuration parameters and orbs, an invaluable mechanism that allows users easily package, maintain, and implement reusable pipeline configurations.

Regardless of which config reuse tactic teams adopt, **it's very clear that these config reuse efforts can become very difficult for teams if there aren't dedicated maintainers.** That's where I see a great opportunity for the CI/CD Engineer role to drive these efforts. The CI/CD Engineer can build a solid understanding of common patterns and functionality, which can be captured and encapsulated into useful dynamic execution code.

Properly sized compute/resource nodes

A key aspect of maintaining blazing fast pipelines and valuable feedback loops is ensuring that CI/CD builds are executing on adequately resourced compute nodes.

It's very common that pipeline builds are executed on severely underpowered build resources, and this directly contributes to slower build speeds and longer feedback loops. Determining the specifications for an adequately-sized compute node is not a trivial task. The right balance is one between hardware requirements such as CPU, RAM, network, disk IO capabilities, and maintaining acceptable build times. Hardware requirements differ widely between technology stacks and services, and these details are often neglected by teams. In my experience, this can occur because teams don't fully understand their tech stacks, or they assume that increased compute node capacities are more expensive than they actually are.

Beefier compute nodes do tend to cost more in general but the cost is often not as high as most teams fear. I've had experience with decreasing certain build jobs by more than half after moving to an adequately-powered compute node. By way of example, this specific build job was taking five minutes to complete on a resource class node using two CPU cores and 4GB RAM. I upgraded the resource class to 4 CPU cores + 16GB RAM, which completed the build in 2.1 minutes and only cost a few cents more.

In this scenario, the cost of the resource class did increase a very small amount but the overall decrease in time for that build job also decreased tremendously which created greater savings when factoring other expenses, namely developers waiting for builds to complete. By decreasing the build times, developers are getting feedback faster and can move onto other tasks in their sprints.

CI/CD Engineers can assist teams in getting builds executed on adequately resourced compute nodes. As with many duties shared by developers and operators, some of these seemingly irrelevant details are not monitored or addressed until the impact of undersized nodes is glaringly obvious. Having someone in a role tasked with monitoring and adjusting these compute node issues can save teams time and money, while also ensuring that build times are optimal and stay that way.

Parallelism

I've observed teams who either don't fully understand all of their tech stack's capabilities, or don't take full advantage of these capabilities. For example, I've interacted with individuals with comprehensive test suites that took over 45 minutes to complete within their pipeline builds. They were convinced that this was the only way to execute their tests. I was able to help them take advantage of the multi-threaded processing capabilities included in their tech stack.

Most tech stacks have the capability to execute code in parallel, which means executing multiple elements and functions at the same time using the available unused CPU cores of the compute node. Parallelism, also known as concurrency, is dependent on the tech stack. It is either offered natively, or it can be implemented by using existing multi-threading libraries or features. Multi-threading capabilities speed things up dramatically. They can be engaged at the stack level -- versus the CI/CD pipeline level -- where code is executed as defined in the CI/CD configuration syntax. By executing code concurrently, execution times are optimized from the beginning, and when applied to a build job in a pipeline, those build jobs become substantially faster, without having to tweak CI/CD configuration parameters in the build directives.

In this case, a CI/CD Engineer could assist teams in enabling multi-threading in the core tech stack, and leveraging the often underused CPU cores when code is executing. **This role can identify the build jobs that are not efficient, and can collaborate on implementing effective execution strategies.** These strategies implement multi-threading at the tech stack level, spawning concurrent process instances that exploit all the CPU resources available to it.

These optimizations can also be implemented and executed within CI/CD platform builds. CircleCI has a parallelism concept which is not related to the multi-threaded concurrency I discussed earlier. CircleCI's version of parallelism enables the execution of multiple build jobs to occur at the same time on individual executors. Having a CI/CD Engineer who can oversee these potential optimization opportunities is yet another justification for the role among DevOps teams.

Performance benchmarks for CI/CD Engineers

A CI/CD Engineer's job is to help maintain and improve pipeline consistency and velocity without risking quality. At CircleCI, we have extensive data regarding the CI/CD builds executed on our platform, and this data has enabled our team to generate valuable performance benchmarks. These performance benchmarks are at the core of some interesting delivery metrics that can be used by teams as goals. The 2020 State of Software Delivery: Data-Backed Benchmarks for Engineering Teams shows how software development teams can measure their performance based on these data points or benchmarks:

Throughput	The number of workflow runs matters less than being at a deploy-ready state most or all of the time
Duration	Teams want to aim for workflow durations in the range of five to ten minutes
Recovery Time	Teams should aim to recover from any failed runs by fixing or reverting in under an hour
Success Rate	Success rates above 90% should be your standard for the default branch of an application

These four benchmarks are baseline metrics that should be monitored, captured, and improved upon. Every organization and team has unique challenges and goals which impact development productivity. Controlling that impact hinges on expanding and improving the underlying processes.

See the research for a more in-depth look at what 55 million data points tell us about CI/CD performance in practice.

The role of CI/CD Engineer can absolutely help teams conduct valuable monitoring and analysis on the current health of delivery and CI/CD performance. With their technical expertise and deep understanding of pipeline execution, they can collaborate on developing performance goals and metrics that will help teams achieve and maintain the results they want.

All too often software development teams are interested in increasing development velocity but are too disconnected from the actual build activities that factor into and control these outcomes. **A CI/CD Engineer is closely tied to the delivery process and can effectively monitor and address deficiencies as they occur, as well as expand and improve on new or existing benchmarks.** They can provide near real-time surveillance to enable teams to successfully hum along at their desired pace.

Conclusion

The CI/CD Engineer is a role that will continue to increase in importance. As software delivery gets more complex, the role becomes more impactful to the overall bottom line. A CI/CD Engineer is an enormous asset to a fast-moving software team. Imagine having someone on your team who creates paved roads to increase speed of feature release, monitors security and overall health of the delivery pipeline, and constantly seeks opportunities to finely tune and optimize the team's delivery.

The most impactful CI/CD Engineers know that the right combination of tooling, culture and collaboration, and an eye toward continuous improvement, is what sets successful teams apart. They are committed to finding efficiencies, smoother paths, and more finely-tuned optimizations to keep their team delivering the highest quality software.

To explore the preferred tool of CI/CD Engineers at Facebook, Spotify, and Coinbase, visit circleci.com.