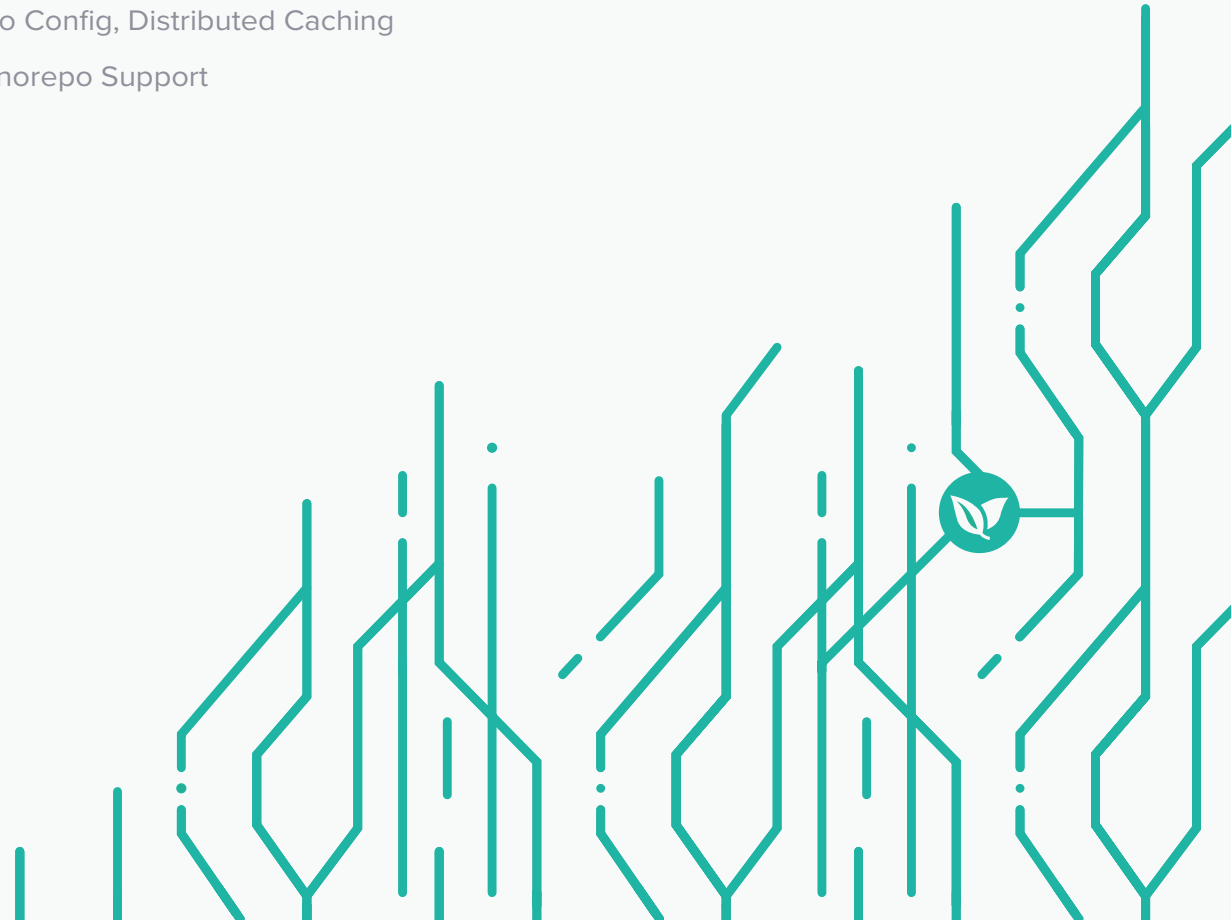
codefresh

13 Key Features Every Modern CI/CD Tool Should Contain

DevOps methodologies have become a huge staple of the Software Development Lifecycle. As such, more and more companies have been adopting the need for Continuous Integration/Continuous Delivery (CI/CD) tools. Building a successful CI/CD workflow can be a tedious process that requires your team to participate in a large amount of preparation and planning — but it doesn't have to be that way.

There are a growing number of CI/CD tools available on the market today, so much that your team may be overwhelmed in narrowing down what to use. In this blog post, we hope to help simplify your choices by breaking down the 13 key features every modern CI/CD tool should contain:

1. Docker-based Architecture Right From the Start
2. Cloud and Version Control Agnosticism
3. Pipeline Creation with Standardized Definitions
4. Graphical Pipeline View
5. Parallel Steps
6. Standardized Plugin Mechanism (Docker-based)
7. Configuration Options Through Both Code and UI
8. Reusable Pipelines for Microservices
9. Live Pipeline Debugging with Breakpoints
10. Native Support for Kubernetes, Helm, and Docker
11. Saas, On-prem, and Hybrid Installation Methods
12. Zero Config, Distributed Caching
13. Monorepo Support



Docker-based Architecture Right From the Start

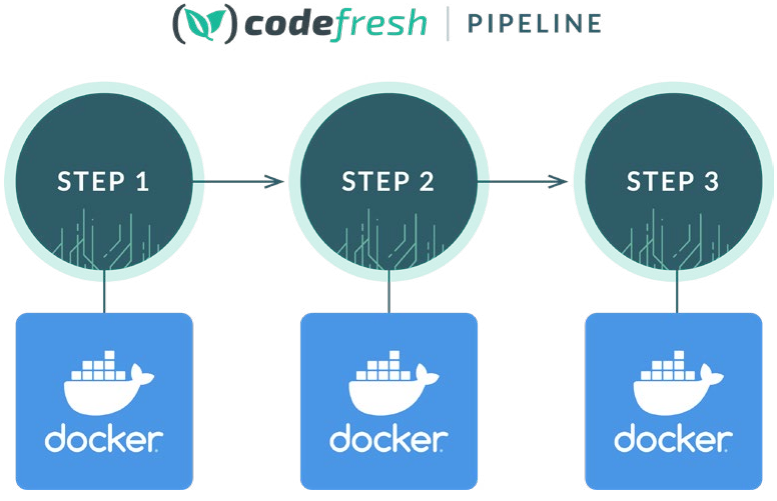
A modern CI/CD tool should have Docker-based architecture since its inception. Many people consider “Docker-based” to mean the usage of Docker as a deployment package, but the usage of Docker-based in this sense means using Docker for the build tooling itself. Traditionally, in VM-based CI/CD tools, you had to make sure all of your build tooling (and correct versions) were pre-installed on the virtual machine in order to run the pipeline. Tools were static and used for all pipelines.

With Docker-based CI/CD tooling, only Docker needs to be pre-installed on your build nodes — nothing else. Your build tools are now isolated to each pipeline, and you can dynamically pop them in or out based on your needs.

Dockerizing the CI/CD architecture itself allows your team to create and reuse building blocks to make pipeline creation much faster and easier, cutting the amount of time and cost it takes to get your CI/CD builds up and running:

- Docker is the de-facto standard for delivery and not tied to any vendor
  - Build, share, and run your steps/images across any cloud provider, OS, language/framework
- Any Docker image is a potential step for a pipeline
  - As long as you have a Docker image, you can use it as a part of your pipeline tooling
- Create pipelines quickly from existing images
- Extending a pipeline step requires Docker knowledge and nothing more
  - No need for Dev teams to understand the underlying infrastructure

While a number of tools have strategically shifted to a “Docker-based” approach, Codefresh offered this functionality right from the start from the heart of its architecture. All Codefresh pipelines use Docker images in one form or another, whether it be by using them as runtime tools, or creating them as deployment artifacts. Codefresh has always required that all tools used in a pipeline be Dockerized, and each step in the pipeline runs in the context of its own Docker image:



Cloud and Version Control Agnosticism

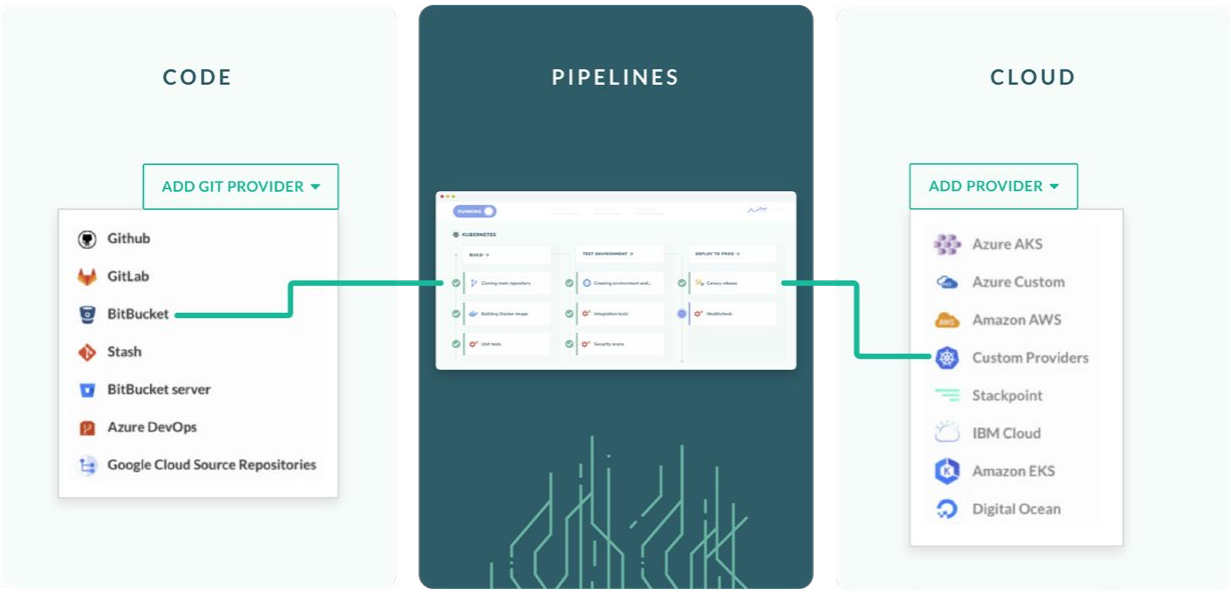
Many CI/CD tools limit which providers you can use for both version control and cloud providers. A good CI/CD tool should be cloud and version control agnostic — it should:

- Play well with any Git or Cloud provider
- Give you a great amount of flexibility on how you choose to version control and deploy your code
- Have no vendor lock-in

Many tools do not offer such agnosticism with version control, and are tied to their own version control systems. The same can be said for Cloud agnosticism as well. Some CI/CD tools from major Cloud Service Providers can be examples of this.

One of the biggest advantages of Codefresh is the fact that it offers native support (and great documentation!) for **any** Git or Cloud provider, giving you a great amount of flexibility on how you build your pipelines. With Codefresh, there is no vendor-lock in and you are free to choose the tools best-suited for your needs.

codefresh | Cloud and Git Agnostic



Oftentimes, companies use multiple different cloud and git providers, both on-premise and in the cloud, so offering cloud and git agnosticism is essential.

Pipeline Creation with Standardized Definitions

Using a standardized YAML approach for programmatic deployments is important. Some tools offer non-standardized approaches, offering pipeline configuration in different modes, such as scripted, procedural Groovy pipelines. Scripted pipelines do not provide a strict and pre-defined structure, which introduces a lot of complexity when it comes to building out your pipelines, making them complicated and hard to manage. In addition, scripted pipelines force operators into learning a full programming language just to pick up and implement the CI/CD system.

Codefresh simplifies this by offering only one mode for constructing pipelines: declarative YAML. This keeps your pipelines (and the documentation supporting it):

- Standardized and easy to pick up and implement, speeding up the time it takes to get from source code to deployment.
- Managed via source file and able to go through a standard PR / code review process, meaning the state of your code and your pipeline are always aligned
- Reusable, as YAML can be used as building blocks to distribute to other projects

In addition to a one-size-fits-all approach to pipeline configuration, a modern CI/CD tool should allow for easy, global, one-time authentication/integrations for Docker registries and Kubernetes clusters.

With a modern CI/CD tool, the less manual configurations, the better, as it:

- Enables your DevOps team to get up and running, focusing on building out their pipelines as opposed to dealing with messy configuration options
- Reduce the amount of time it takes to administer authentication options within your pipeline
- Makes CI/CD easy for everyone to use without requiring everyone to learn a new language to create pipelines

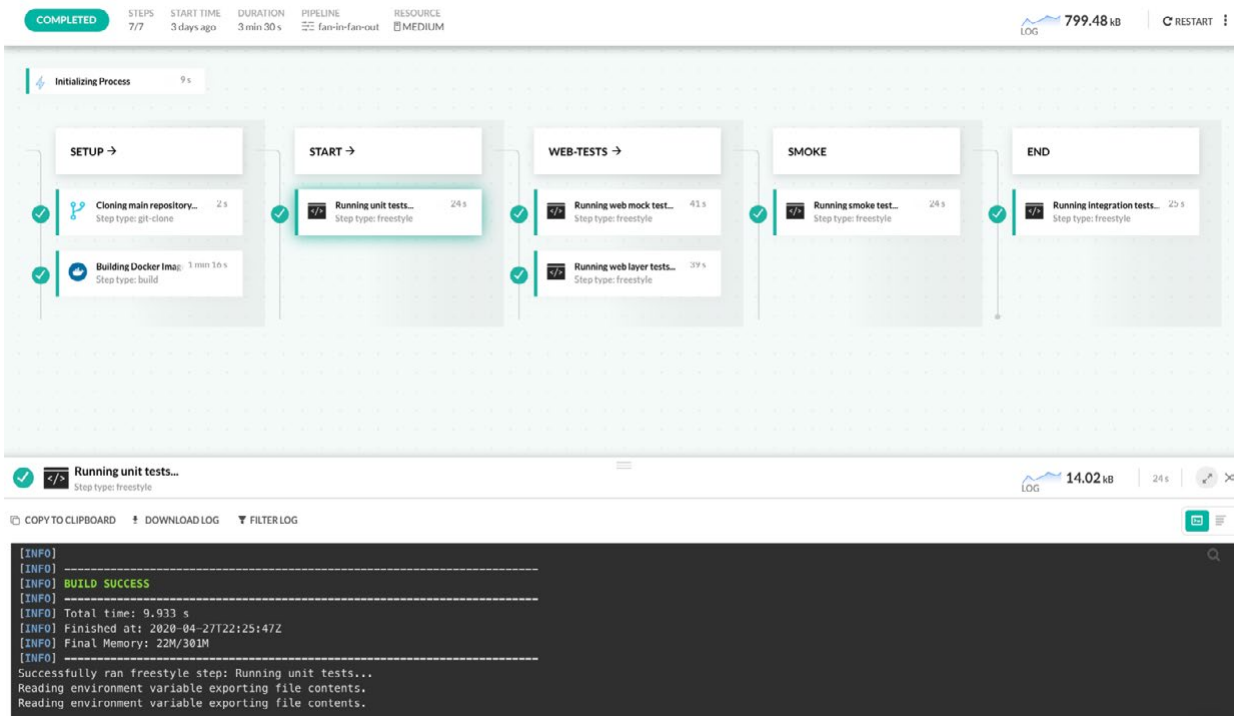
Codefresh provides this capability automatically to all pipelines. You authenticate one time through your global account settings to Docker registries (without the need to run docker login), Kubernetes clusters (without the need to run kubectl commands), Secret repositories, Helm repositories, and so on — once you configure this one time, authentication through the pipeline is all handled automatically for you. deploying them further apart from each other.

Graphical Pipeline View

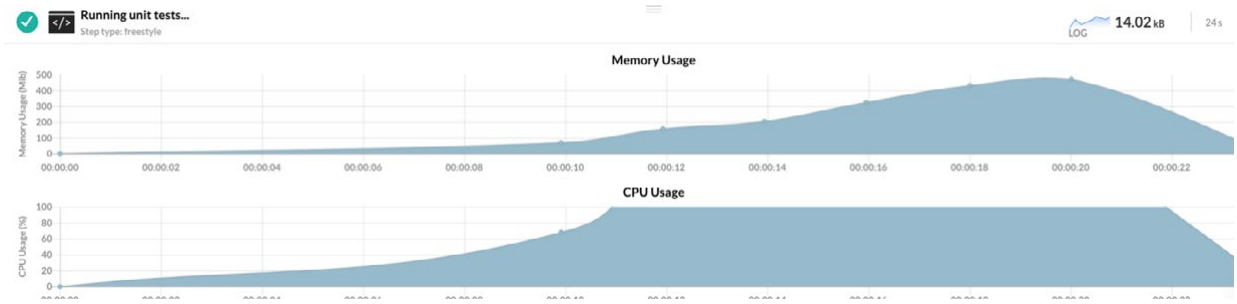
Any modern CI/CD tool should provide an intuitive user interface, complete with a graphical pipeline view of your workflow. Most CI/CD tools offer this, but the user interface/ experience can be lacking in critical build information and modern design principles.

At Codefresh, we provide a user-friendly experience when it comes to viewing your pipelines graphically. Your pipeline steps can even be organized into different stages — they are completely customizable. You define as many stages as you wish, and define what steps are grouped under each stage. Our graphical pipeline view provides:

- **Any** sort of stakeholder an overview of your pipeline and what is occurring at each step
- The YAML, logs, state, and metrics of the build as a whole or as a part of any particular step



Getting a holistic view of your pipeline is extremely important, the larger and more complex your pipeline gets.



The ability to see the metrics of your pipeline allows you to instantly view bottlenecks and optimize slow build times without any guesswork.

Parallel Steps

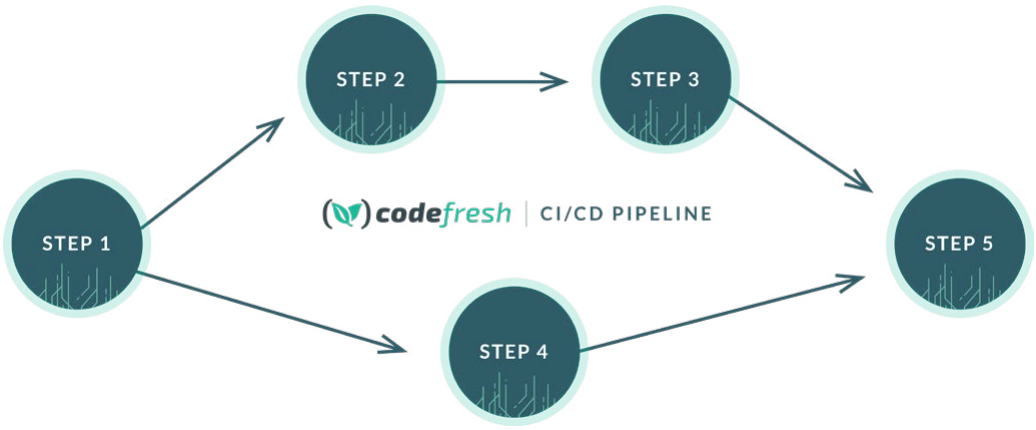
Most common pipeline steps run sequentially, meaning each step in the pipeline is executed individually, one at a time. The ability to run steps in parallel as opposed to sequentially is a very important feature of any modern CI/CD tool, as it can dramatically decrease the amount of time it takes for your pipeline to complete.

For example, consider a sequential pipeline with 5 steps, each step running its own set of tests: unit tests, fast tests, slow tests, smoke tests, and integration tests. For simplicity, let's say each set of tests takes 1 minute to run. In a sequential pipeline, the build time would be a total of 5 minutes. Parallelizing all of these steps would cut the time it takes the pipeline to run down to 1 minute, reducing the time your engineers wait for a build to finish. CI/CD tools that offer parallelization are beneficial in:

- Cost efficiency
  - Running steps in parallel makes the cost per step (the amount of time it takes for the entire build process to complete) much lower
- CI/CD optimization / faster feedback
  - Run testing steps as soon as a developer pushes a new commit (throughout the entire software development lifecycle), giving your team quick feedback

Most modern tools offer this capability, but only at a basic level of splitting a single task into parallel tasks Codefresh takes it one step further, by allowing you to create a custom graph of dependencies between parallel steps, giving you maximum flexibility on any complex flow.

For example, at Codefresh, to run steps in parallel, we support two modes: one for parallelizing your entire pipeline, and one for parallelizing only specific portions of it.



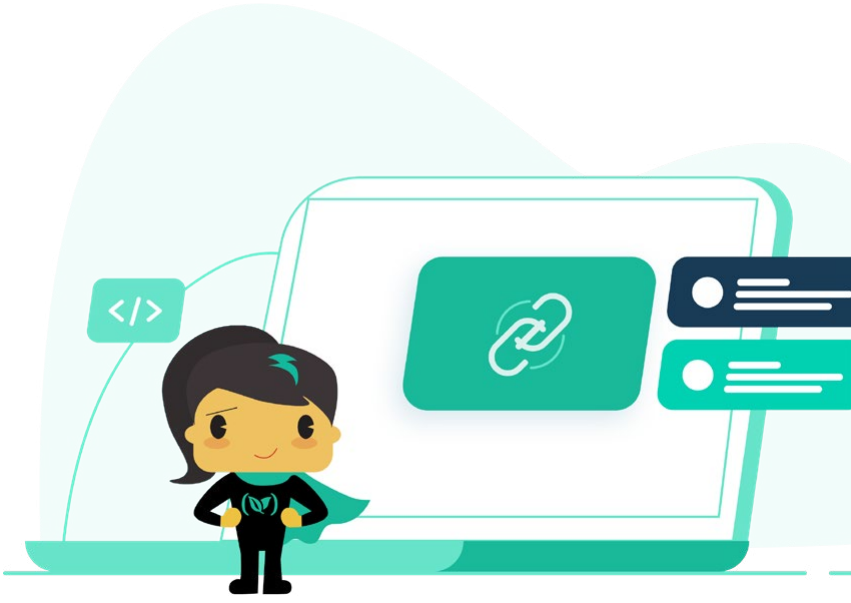
The advantage of parallelism within Codefresh is that we offer support for an advanced parallel mode with full step definitions and their explicit dependencies, allowing for complex fan-in, fan-out configurations as seen in the diagram above.

Standardized Plugin Mechanism (Docker-based)

Many CI/CD tools support a marketplace of plugins, but the method of creating one is not standardized. A modern CI/CD tool should have a standardized, Docker-based mechanism for plugin creation. This allows:

- Developers to create their own plugins without special knowledge of the CI/CD tool's API or specific language used for that platform.
- Every team member to create plugins in a standardized environment, i.e. solving the issue of "it works on my machine"
- Reuse of existing plugins from other teams
- The ability to write a plugin in **any** programming language, as long as it is packaged in a Docker image
- Any public Dockerhub image is a potential pipeline step within your plugin

At the base of every plugin, Codefresh uses a Docker image, simplifying the plugin creation process. And, since Codefresh has a Docker-based build architecture from its heart, you can easily plug-and-play with any plugin from our marketplace into your pipelines.



Take a look at any of our plugins (they are all open-source!) and you will see that they are composed of a Dockerfile, making it easy for developers to extend and add their own, without any background knowledge of our API. In addition, plugins are scoped per pipeline, so you have no dependency hell to monitor between plugins, or a need to monitor which versions are compatible with your platform version.

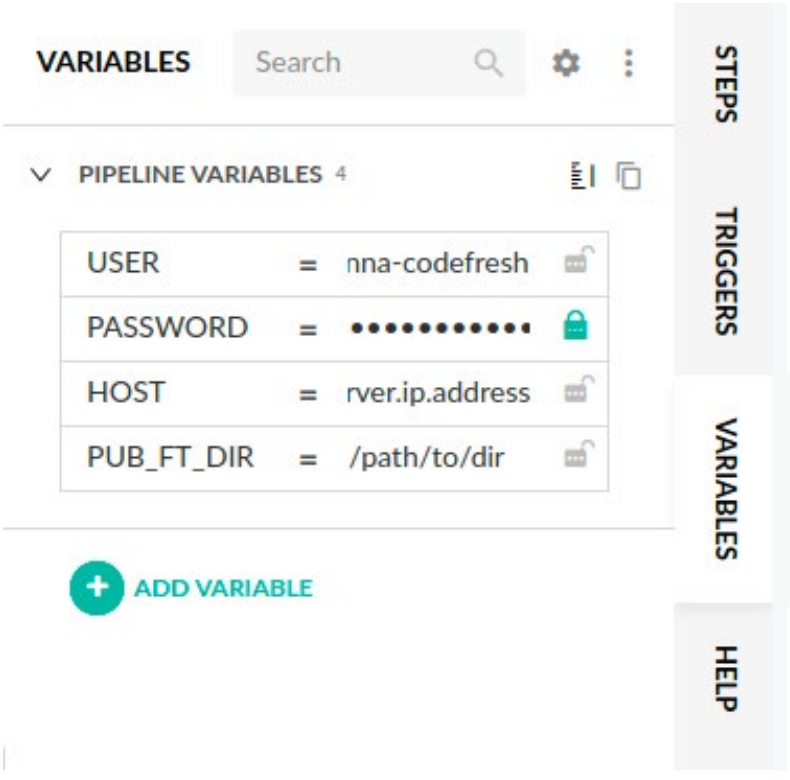
Configuration Options Through Both Code and UI

A modern CI/CD tool should have both Configuration as Code as well as Graphical User Interface options. Many tools offer configuration through code, but lack in UI configurations, or vise versa. Allowing a means of both offers:

- Flexibility, in the event you have someone on your team who wants to take a less-technical approach to managing their configurations
- Versioning of changes
- Configuration as code allows configuration to be stored as code in your version control system, allowing you to easily track changes by whom and when

Codefresh offers both types of configurations for many options. For example, we offer an in-line editor you can use directly from the UI to perform dry-runs of your pipeline, and once your pipeline is ready, you can move it to source control and use it from there. The pipeline can even be in a different repository from your application's source, allowing for maximum flexibility.

Moreover, you can define variables as a part of the YAML itself, or, we give you the flexibility to define these variables as a part of the pipeline settings, through the UI:



Developers and beginners tend to work with the GUI, while Ops teams and advanced users tend to work with configuration as code. Offering a means to support both keeps teams happy and is vital to any organization.

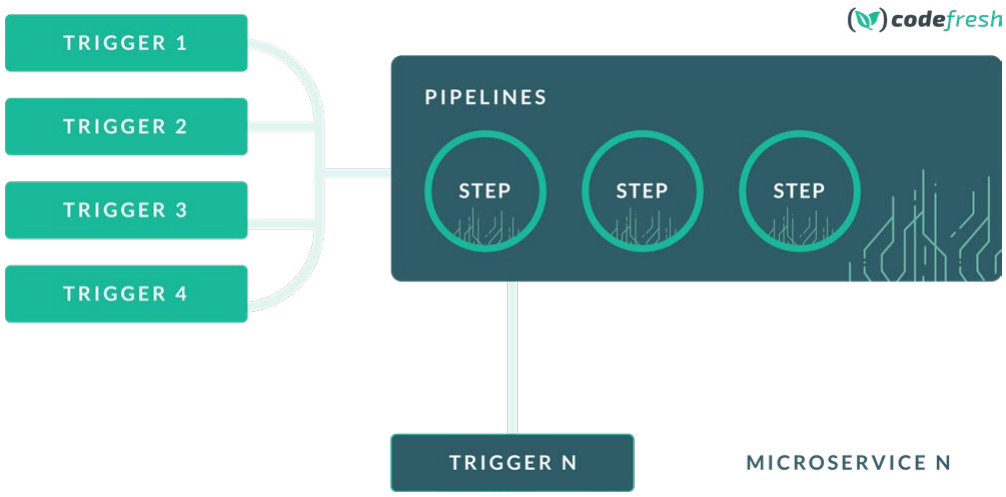
Reusable Pipelines for Microservices

A modern CI/CD tool should have a mechanism in place to reuse pipelines for Microservice applications. Most microservices have a similar pipeline pattern in place. Since those similarities exist, it should be possible to reuse a single pipeline across many microservices.

One of the ground-breaking features of Codefresh is that pipelines are not bound to specific Git repositories, meaning you can re-use them amongst different microservices by just adding more triggers. This allows for simplified:

- Pipeline construction
  - You have only one pipeline to build and it acts as one, reusable unit
- Project construction
  - Any time a new microservice is added, you only need to add a new trigger. No need to rebuild the existing pipeline
- Pipeline enforcement
  - Updating a single pipeline affects all other microservices associated with it, so you don't have to hunt down which pipeline affected what

Codefresh offers a variety of triggers, including Git, Dockerhub, Azure, Quay, Artifactory, Cron, and the Codefresh API/CLI.



Take for example, the above diagram. The pipeline process is the same for all microservices, and four microservices already exist (and thus, four triggers). When a new microservice is created, the Ops team simply adds a new trigger to the existing pipeline, and the work is finished. This speeds up the amount of time needed to create a new project, compared to other solutions where bootstrapping a new microservice can take days to complete.

Live Pipeline Debugging with Breakpoints

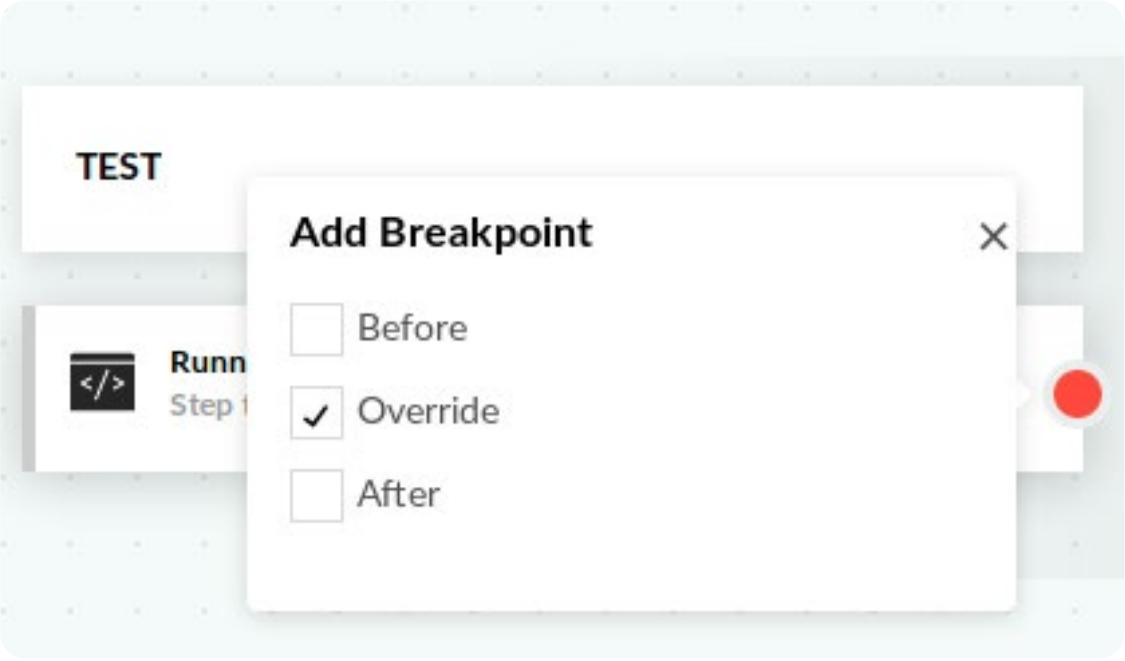
The ability to troubleshoot a failed build is absolutely essential when it comes to an efficient CI/CD tool. Many CI/CD tools offer simple debugging with SSH, enabling users to debug a failed build from the builder node.

Codefresh takes a unique approach to pipeline debugging, on the other hand, by being the **first ever** CI/CD solution to make pipeline debugging live:

- Live debugging is more user-friendly and the way most developers are already accustomed to debugging their code
- It makes the CI/CD creation process easier for engineers to test, troubleshoot, and fix their pipelines

Similar to the way developers debug applications live in their IDEs, you can place breakpoints within your pipeline steps to inspect the live states of your build.

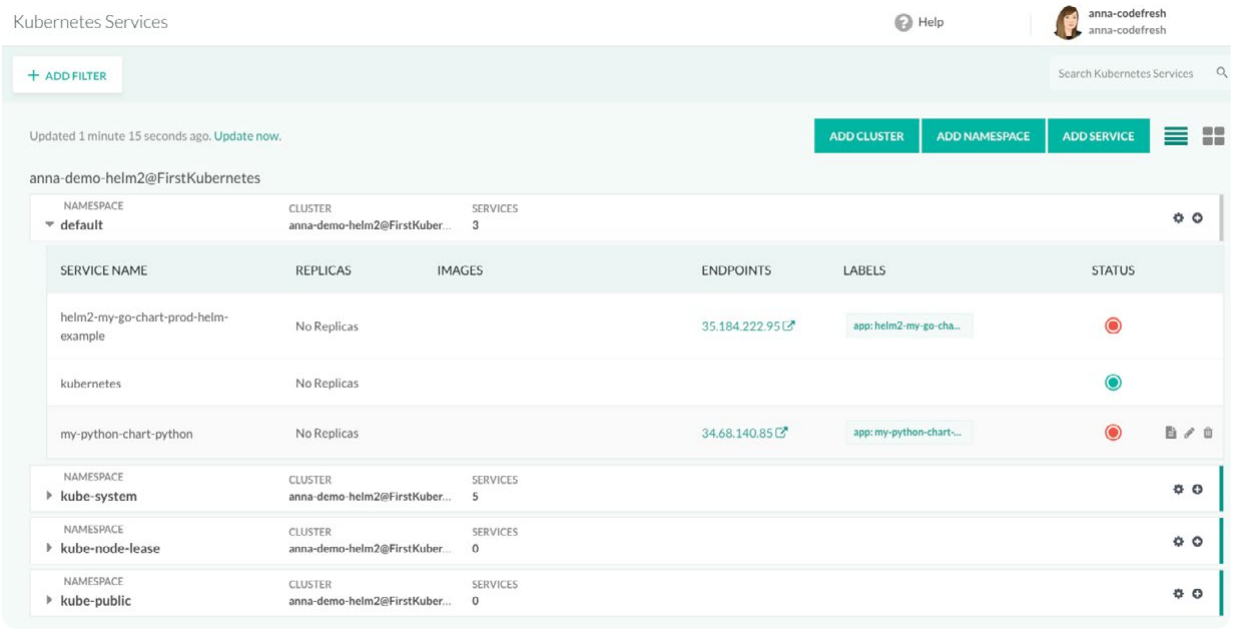
Once the pipeline pauses at your breakpoint, you have the ability to run any commands you wish to understand the state of your container, making creating, modifying, and testing your pipelines much easier.



Native Support for Kubernetes, Helm, and Docker

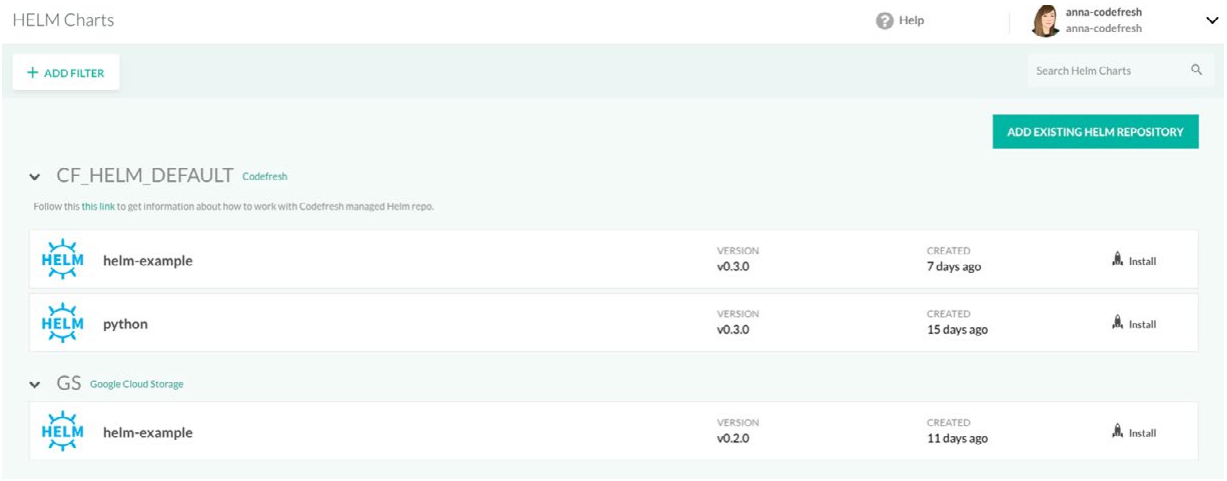
This is a growing area of interest, as many companies wish to adopt cloud native approaches. Many tools offer integrations with Kubernetes, Helm, and Docker through the use of plugins, but few have native support for all three. In fact, Codefresh works out-of-the-box with Kubernetes, Helm, and Docker, without the need of any special plugins or configurations. It is the only CI/CD solution that provides native Kubernetes, Helm, and Docker dashboards.

- With our Kubernetes dashboard, you get:
  - Full traceability into your clusters, as you can monitor what images are currently running, and trace them all the way back to an individual commit.
  - The ability to create and edit services, deployments, and more
  - Visibility across clusters all in one place, along with their state, status, and any errors

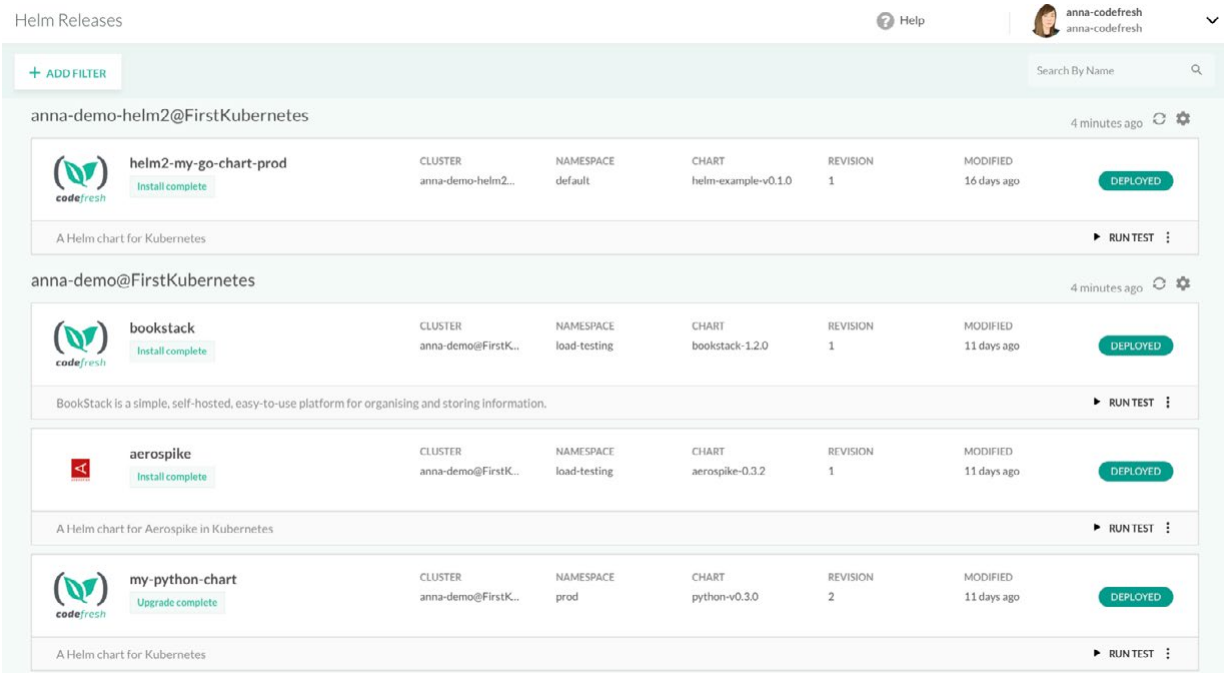


Kubernetes dashboard with full traceability into your clusters

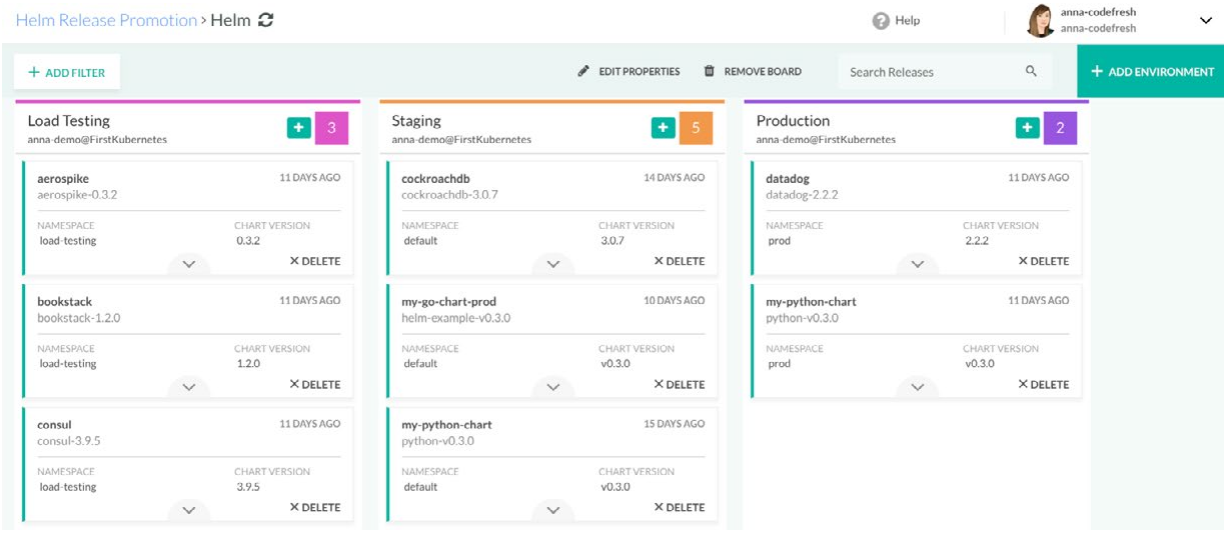
- With our Helm dashboards, you get:
  - A built-in Codefresh-hosted chart repository with built-in, one-click installation
  - View the status of currently deployed releases
  - One-click rollbacks of your production Kubernetes cluster
  - A rolling version history (with diffs) of various chart versions
  - Drag-and-drop Helm release promotions to different environments



Built-in Helm chart repository view, with one-click installation methods.

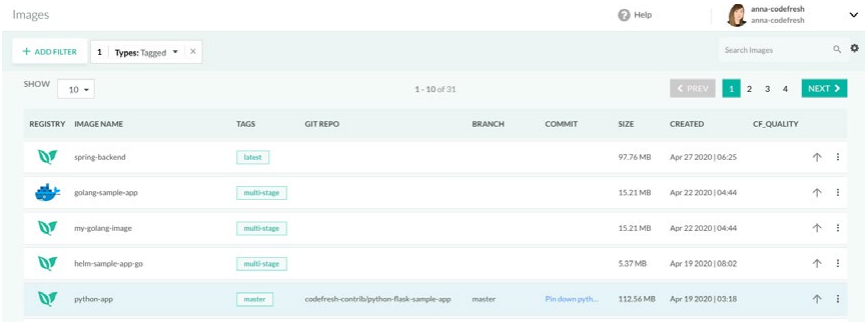


Helm release dashboard, where you can view the status of currently deployed releases.



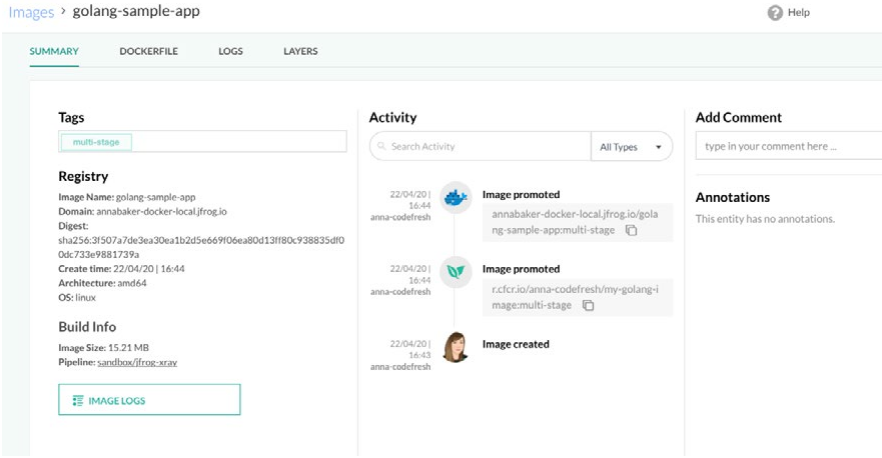
Helm promotion dashboard, giving you an overview of where each release exists in your development lifecycle, and the ability to promote different releases to different environments.

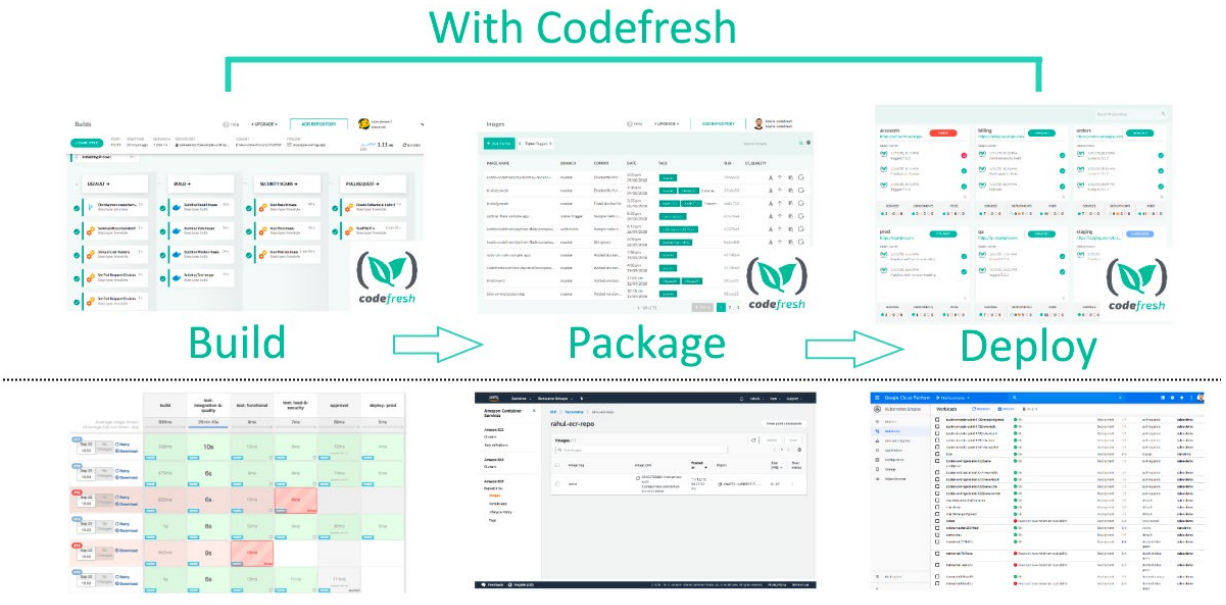
- With our Docker image dashboard, you get:
- Details such as the git branch, commit message, and hash that created it, date of creation as well as any tags
- You can also select any image and look at its individual metadata



From your first commit to your production deployment, Codefresh offers full traceability of your software development lifecycle. You can view all phases of development within a single, unified platform.

With other CI/CD solutions, you don't receive the benefit of this holistic view and have to investigate using other platforms just to trace what is going on with a release.





Without Codefresh 3 different solutions are needed

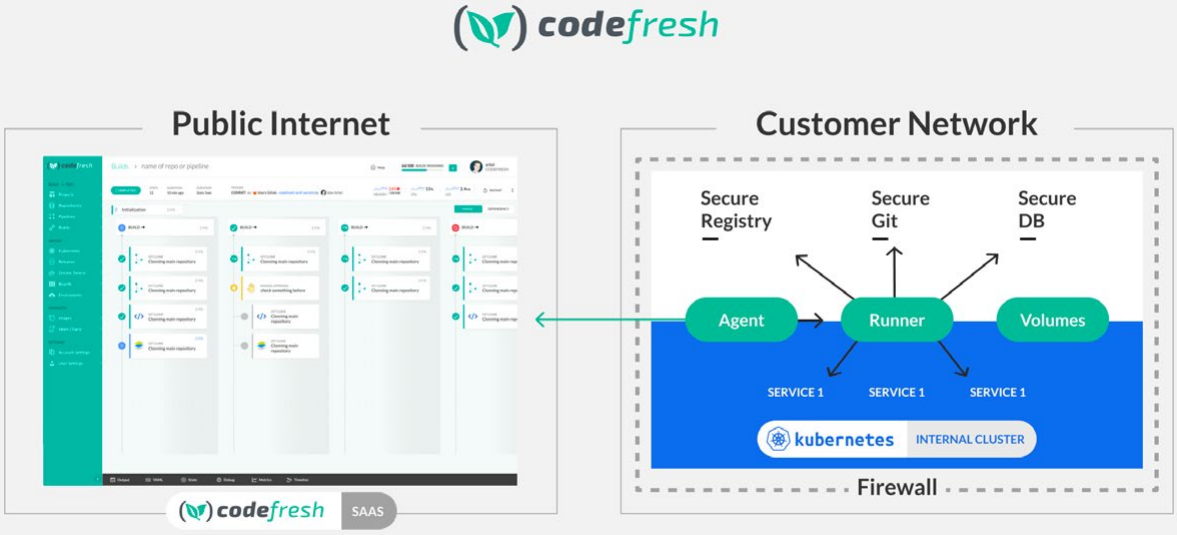
SaaS, On-prem, and Hybrid Installation Methods

There is always a debate as to whether or not to host the CI/CD system yourself, or offload the work to a cloud-based/SaaS service. There are pros and cons to each approach, namely, offloading the work can be beneficial and faster, as you can access a shared pool of the CI/CD tool's hosted resources to run your pipelines. But, with this approach, there are often security concerns around code/data accessibility and many companies feel safer running their pipelines from their own machines. The downside to this approach is that any and all maintenance is now your responsibility.

Any modern CI/CD tool should not lock its customers down into one installation method, but give customers flexibility. In fact, to solve the debate between on-prem vs. SaaS solutions, Codefresh offers the best of both worlds, by offering a Hybrid installation model. With this mode:

- The Codefresh UI runs on the Codefresh infrastructure
- Actual builds happen via the customer's location (behind the firewall, from a Kubernetes cluster).

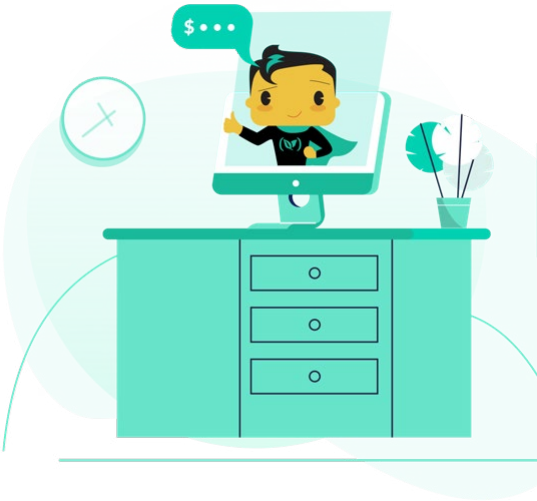
This keeps security concerns at bay, while still leaving all the heavy lifting and maintenance of the platform up to Codefresh.



With the Hybrid mode, you can take advantage of specific custom infrastructure within Codefresh, such as GPU nodes and other specialized hardware. You can also use SAAS and Hybrid simultaneously, so some pipelines can run in the cloud, while others run behind your firewall.

Zero Config, Distributed Caching

A modern CI/CD tool should contain a multitude of caching mechanisms that take place as a part of your pipelines. These caching mechanisms should be:



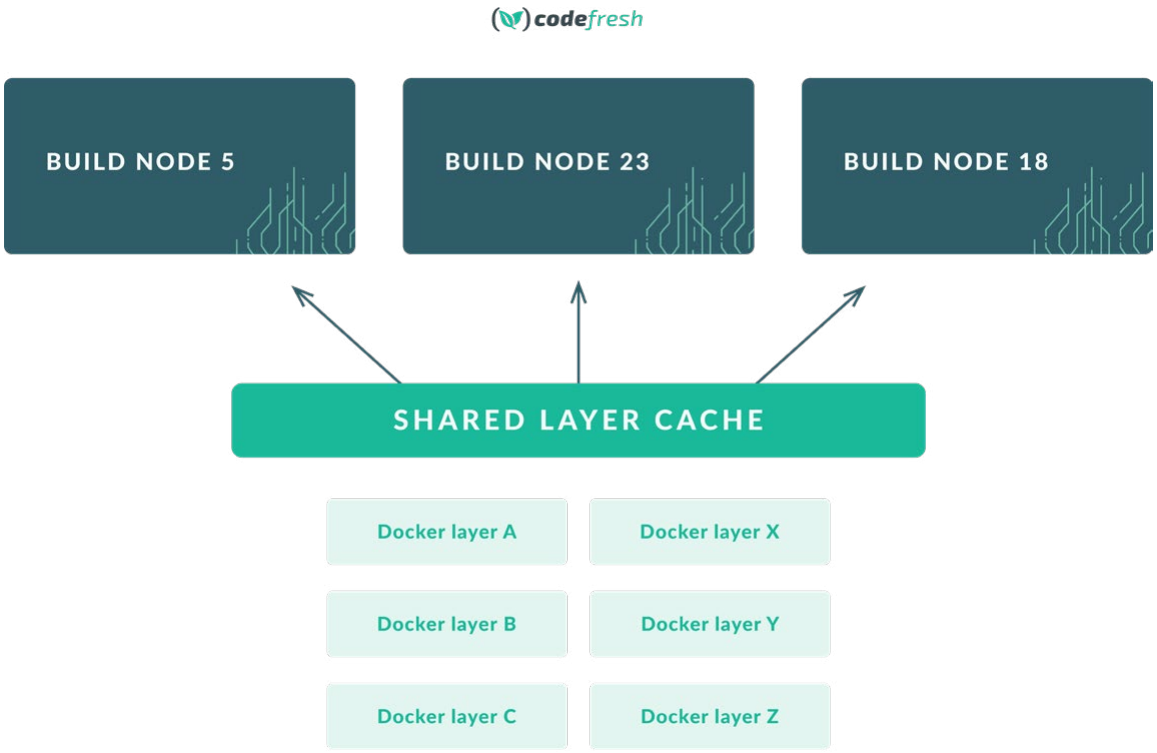
- Automatic, and require little to no configuration
- Distributed across all build nodes or pipeline steps

This allows for decreased network costs, improved responsiveness, and increased hardware performance — all in all, reducing your build times. Lengthy build times are bad for your team for various reasons:

- Reduction in commit frequency
- Due to the cost in time it takes to build, developers will put off building and submit code in larger batches of work, creating a larger surface area for change
- This leads to more difficult integration and a greater risk for merge conflicts
- Small improvements become costly, leading to increased life cycle time overall

Distributed caching is a great way to mitigate the risks of slow build times, and Codefresh offers several types:

- Distributed Docker Image Caching
- During subsequent runs of a pipeline, Codefresh will automatically fetch Docker images from a shared image cache
- Distributed Docker Layer Caching
- Mimics the way Docker Layer Caching behaves locally on your workstation — when building images, Docker will cache intermediate layers making future builds much faster
- Docker Layer Caching is distributed, meaning all build nodes in your pipelines share the same cache. Any available node can pick up your next pipeline build, as all of them share access to previously cached Docker layers.



- Traditional Build Caching
- Many CI solutions make you manually configure what folder you want to cache
- Codefresh offers an internal Docker volume, under /codefresh/volume that is automatically cached for subsequent builds of the same pipeline

Monorepo Support

Monorepo-style projects take an approach where:

- Multiple projects are committed to the same repository
- Projects depend upon one another and can share a common codebase
- When changes are made, only specific projects affected by that change are rebuilt/retested, as opposed to rebuilding the entire monorepo itself

The advantages of monorepos are:

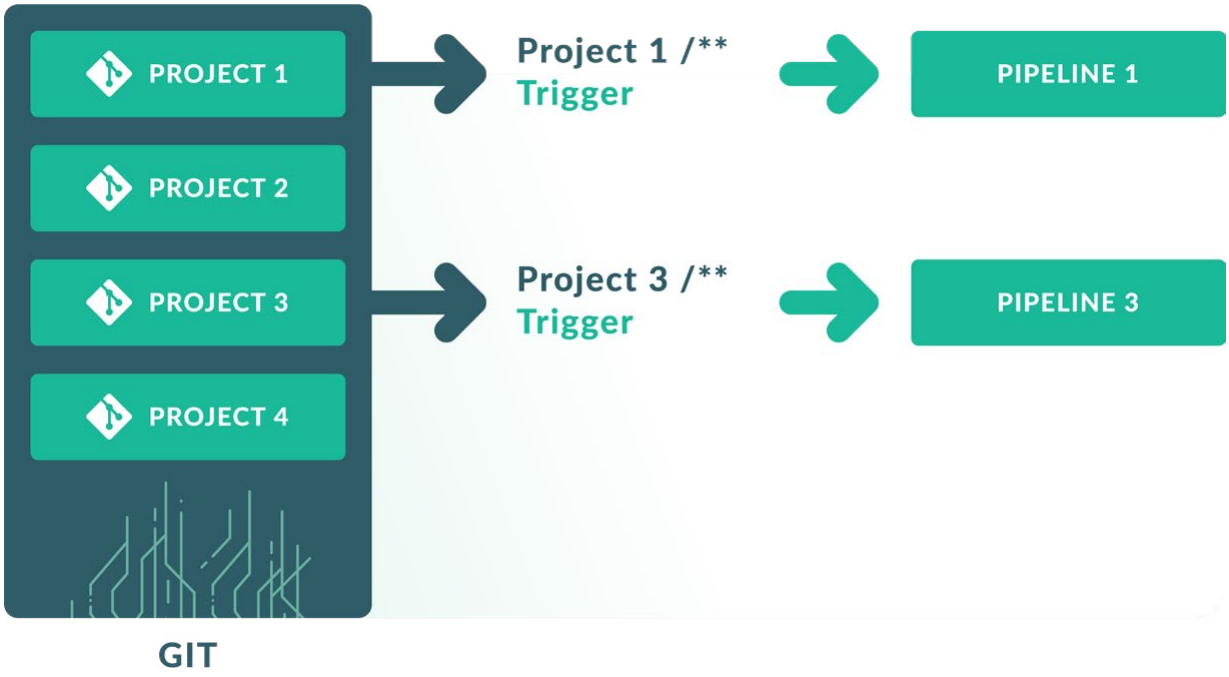
- Developers only need to update and check-out one repository
- Single source of truth for all versions of your code and their dependent projects
- Atomic commits: changes to multiple projects can be checked in as one commit
- Merge code only once, as opposed to multiple times across multiple repositories
- Simplification of merge conflict handling

Monorepo support should be a key feature of any modern CI/CD tool, as it greatly decreases the amount of builds occuring in your daily development cycles. If you use a CI/CD tool without monorepo support, you will:

- Go through the entire build/test/deploy process for the entire repository when any changes are introduced in any one microservice
- Have to create custom in-house tools that mimic the same functionality Codefresh offers out-of-the-box

Transitioning to a monorepo from a multi-repo approach may require you to rethink how you structure your CI/CD process. You only need to build the artifacts affected by that change, as you are no longer building a single application, i.e. triggering pipelines to execute only when changes happen in specific folders.

Codefresh simplifies this process for you, by providing easy-to-use monorepo support. In the trigger of each pipeline, you define a glob expression that maps to the project's files. Only when the expression is matched, will the pipeline execute, meaning you can define pipelines that run a sequence of operations only on the microservices you specify, despite them being located in the same repository.



In the above diagram, we have four projects in our mono-repo. We create two separate pipelines, that execute on different triggers based on their glob expressions.

Conclusion

I hope this list was helpful in providing you with the information you need to choose an efficient, modern CI/CD tool. Obviously, you will need to select a tool that best fits your needs as a team, but the items listed here are what I consider to be a bare-minimum in terms of what your selected product has to offer. Technology moves very fast, and it's important that your CI/CD tool of choice is up-to-date and evolves with new development trends and practices.

Author: **Anna Baker**

Anna Baker is a Software Engineer/Technical Writer. She previously worked at Red Hat and is passionate about the open source community. In her free time, she enjoys drawing and cooking dishes from all over the world.

codefresh

Join us at https://codefresh.io