

CLOUD-NATIVE CI/CD: SEVEN REQUIREMENTS



TABLE OF CONTENTS

- 2 Introduction**
 - What is CI/CD?
 - What is Cloud Native?
- 4 Built on Cloud Native Standards**
 - Born in the Cloud
 - Built on Established Tools
 - API Control
- 6 Infrastructure Agnostic**
 - Managed for Multicloud
 - Self-Hosted for Hybrid
 - Feature Equivalence
- 8 Infinitely and Dynamically Scalable**
 - On-Demand Computing
 - Pipeline Concurrency
- 9 Configured by Code**
 - Speaking Cloud Native
 - Code with Best Practices
- 10 Container Run**
 - Runtimes Out of the Box
 - Custom Runtimes Enabled
 - Orchestrated by Kubernetes
- 12 Observable**
 - Real-Time Activity
 - Rich Logging, Monitoring and Analysis
 - Capturing Performance
 - Connectable for Power
- 14 Enables Cloud Native Development**
 - Docker Smart
 - Kubernetes Connected
- 15 How JFrog Measures Up**



INTRODUCTION

With the rise of the cloud in IT operations, it's become extremely important that those operations, and the tools you use for them, be "cloud native."

At the same time, as enterprises adopt DevOps as part of their digital transformation, CI/CD is now the powering engine of their Software Development Life Cycle. How efficiently your CI/CD runs in your operations environment -- which is increasingly reliant on the cloud -- can make the difference between success and failure. So it's no surprise that many CI/CD solutions seek to claim the mantle of cloud native. But are they?

WHAT IS CI/CD?

The software development practices of continuous integration/continuous deployment (CI/CD) mean developing and releasing with smaller, incremental changes, leading to much more frequent builds. Instead of major annual, semi-annual, or quarterly releases, CI/CD may produce a new minor release every day -- or even several each day. This greater frequency is enabled by a CI server tool, whose job is to use the power of the network to automate and distribute the work of building, testing, releasing, and deployment.






WHAT IS CLOUD NATIVE?

Here's how the Cloud Native Computing Foundation defines cloud native:

"Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach."

That's a very broad definition. Here's how we like to think of it:

Cloud native solutions make use of methodologies that effectively utilize cloud technology infrastructure, and enable the inherent best characteristics of the cloud:

-  Elasticity - Leveraging the power of the network to expand and release resources as needed.
-  Global Scalability - Using the reach of worldwide networks to provide concurrent service to and from anywhere.
-  Always Available - Ensuring uninterrupted 24/7 performance, regardless of load.
-  Resilience - Composed of loosely coupled services that can either self-correct or isolate failure while the remaining services continue to operate.
-  Observability - Making operations transparent to administrators or users through standard or custom tools.

Now that we're clear on what CI/CD and cloud native means, how does this shape our expectations of a cloud native CI/CD solution?

What should a cloud native CI/CD solution look like?

The Cloud Native Computing Foundation (CNCF) is the central, vendor-neutral home of several cloud-enabling open source projects, including Kubernetes, Prometheus and Envoy. With all the world's largest cloud computing companies as members, as well as many innovative solution providers (including JFrog), the CNCF is the cooperative organization that helps define and promote cloud native development.



CLOUD NATIVE
COMPUTING FOUNDATION



BUILT ON CLOUD NATIVE STANDARDS

When you update your automobile engine, you need to use tools and parts designed to fit. Sure, you might make something else work but how well and for how long?

The same principle applies to your operations infrastructure. Any solution run in a cloud should use the established technologies meant to make cloud computing work at its best.

BORN IN THE CLOUD

A truly cloud native solution is one that was written for the cloud from the start, and architected around the technologies designed to enable the cloud's benefits.

There are many CI/CD solutions available, but often they were initially developed for pre-cloud technologies. Those legacy solutions may have been adapted to run in container-based environments, but they're still adaptations, often limited by their original design. Each new innovation in cloud technology will put new stress on their architecture. A legacy solution made functional in the cloud isn't necessarily optimal for the cloud.

Your CI/CD solution shouldn't just work in the cloud, it should be architected for the cloud. At minimum, it should operate as a set of microservices that use cloud native technologies like containers, immutable runtimes, and orchestration. Cloud native CI/CD should be designed to use, and make the best use of, cloud native standards.

BUILT ON ESTABLISHED TOOLS

Your cloud operations already rely on established cloud native tools like Docker and Kubernetes. Your teams have experience with them, know how they work, and have a large ecosystem of tools available to manage and monitor them. You can count on those tools being widely supported.

A cloud native CI/CD solution should run on these standard tools, making it easy for you to leverage the tools you possess and the techniques you've established to run your operations.

At best, your CI/CD solution should be more than just integrated with these cloud staples -- Docker and Kubernetes should be part of their DNA. Your CI server engine should not only install on K8s, but run tightly aligned with K8s as well, relying on its orchestration facilities to keep workloads running.

API CONTROL

You'll want other software and scripts to interact with your CI/CD solution, in order to automate its configuration, to integrate with other tools, and to enable oversight.

REST is the technology that cloud applications use to talk to each other. A cloud native solution should provide a robust set of RESTful APIs for CI/CD, to fully integrate with the rest of your operations.



INFRASTRUCTURE AGNOSTIC

To be cloud native, your CI/CD must be interoperable on many systems, regardless of the underlying hardware. Any solution will require resources such as file storage, but it should be able to operate with several service types, to maximize system compatibility.

When your CI/CD is truly cloud native, it enables you to be cloud nimble, and operate on any cloud you choose without sacrifice, whether that's a commercial cloud service or on an on-premises cloud in your own datacenter.

MANAGED FOR MULTICLOUD

A solution that's cloud native can and should be readily provided by its vendor as a managed service through the SaaS model. When you subscribe to a managed service, you gain an inherently scalable, always-available solution that is administered by the vendor. You can contain your CI/CD operational costs by paying for resources such as file storage and build minutes as you use them.

But availability on a single cloud service provider isn't enough. Cloud native is of limited benefit if you are locked into a single cloud platform. To be cloud nimble, your managed service CI/CD should be available on at least all of the major cloud providers (AWS, Google Cloud Platform, and Azure).

Having these provider options enables you to practice a multicloud strategy for your CI/CD, to increase agility, share and shift workloads across cloud providers, maximize cost savings, and create redundancy for disaster recovery.

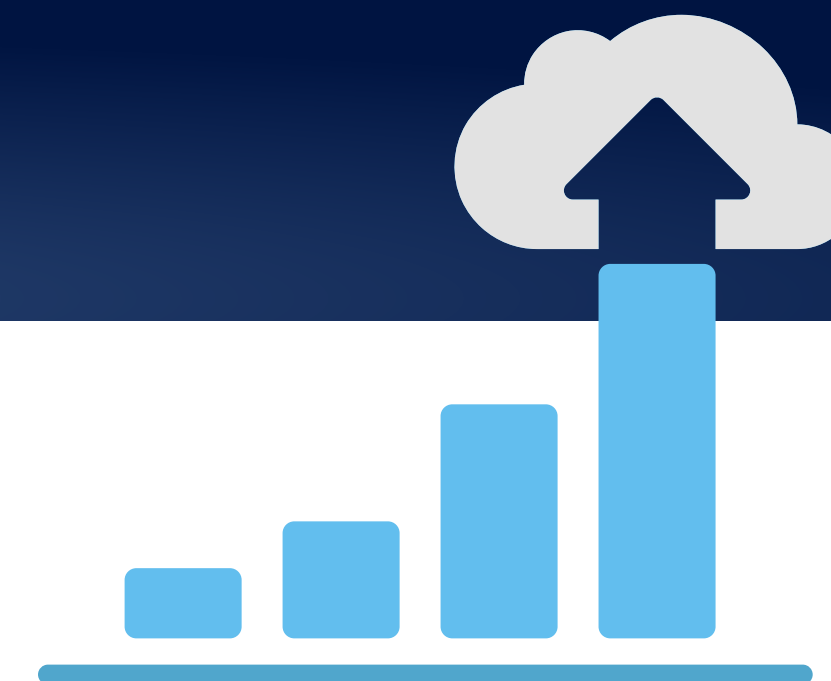
FEATURE EQUIVALANCE

Being cloud native means never having to give anything up for your chosen platform. If your CI/CD is architected around standard cloud technologies, it should operate with the same capabilities no matter what cloud it runs in.

A solution whose cloud and on-prem editions run with a different UI, features, or functionality simply isn't likely to be cloud native. At least one of those editions is certain to be an adapted legacy product, rather than one that was designed for the cloud from the start.

Your developers should never have to think about what platform their CI/CD is running on, because it always looks and functions the same. When a solution is fully platform congruent, you'll never have to compromise features to interoperate cloud-to-cloud or cloud-to-on-prem.

INFINITELY AND DYNAMICALLY SCALABLE



Scalability is what the cloud is all about: to serve changing demand, support growing teams, and stretch your operations across the globe. And you need all your services to scale up and down quickly without interrupting work.

ON-DEMAND COMPUTING

Any cloud native CI/CD should be as elastic as the cloud it runs in, and smart enough to respond to high and low demand. It should fetch build resources when they're needed and release them when they're not.

On-demand computing can keep your CI/CD humming nonstop without limit, while helping your costs drop as demand does.

PIPELINE CONCURRENCY

This cloud native feature enables *pipeline concurrency*, a feature that provides benefits in two dimensions.

First is the ability to execute concurrent workloads within a single build pipeline. Steps that aren't dependent on each other's outputs can be run in parallel in separate runtime resources, speeding the work to twice as fast or more. When spun up on-demand, spare compute engine resources don't need to be held idling in reserve.

Additionally, it also enables running many concurrent pipelines from a single, central CI/CD server, eliminating the sprawl of multiple CI servers spread throughout an organization. To help keep that growing number of users and jobs performant, a proper cloud native solution can be deployed with high availability through redundant, load-balanced nodes.



CONFIGURED BY CODE

Cloud native solutions enable operations through code, bringing a developer-focused experience to tasks like provisioning infrastructure and deployment.

SPEAKING CLOUD NATIVE

Cloud native technologies like Helm, Kubernetes and more are powered through declarative configuration files. Those files are written in YAML, the language of cloud native. The simple key-value structure of YAML enables developers to specify what they want and rely on the tool to decide how to fulfill it for the underlying infrastructure.

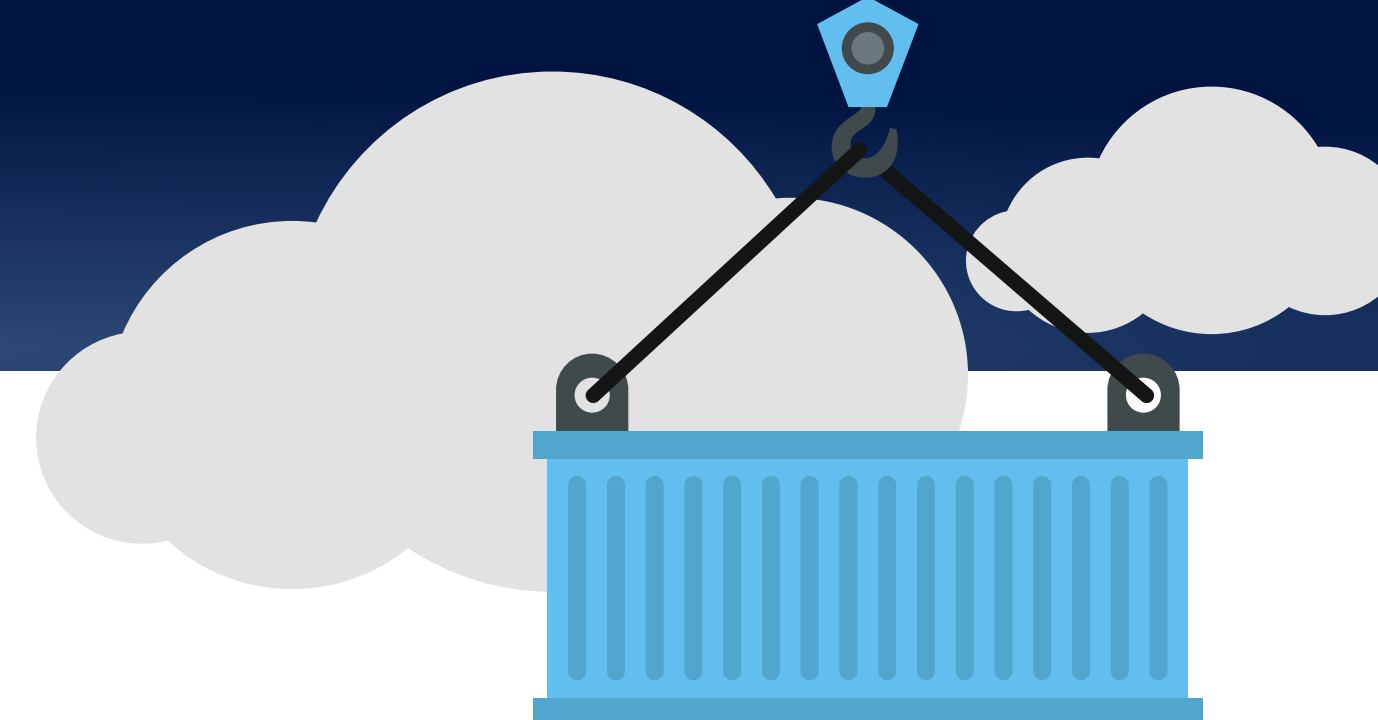
CI/CD that's cloud native should do the same, relying on declarative pipelines that can be specified through a YAML-based DSL. This frees developers to concentrate on the actions their pipelines must perform, and leave it to the CI/CD tool to get them done.

CODE WITH BEST PRACTICES

Developers should be enabled to treat their pipeline code with the same software best practices as they do the rest of their code, through a version control system. So cloud native CI/CD should have the ability to easily integrate with Git source control repositories from the moment it's first installed.

This enables your teams to store and maintain your pipeline configuration files in source repos, and practice the same collaborative procedures to develop and maintain them as they are accustomed to. Your pipeline code files will also be protected by the same backup and recovery precautions you already practice for source control.

Once the CI/CD server is connected to your source repos, it can load the pipeline configuration files stored there directly from those repos, and refresh any time there is a change. By being separate from the CI/CD solution, it's easy to reliably reload your pipelines for any needed systems restore, keeping time to recovery as brief as possible.



CONTAINER RUN

An essential cloud native technology, container architectures are especially suited to the dispatch of build nodes that is the core operation of a CI server.

While virtual machines must be preconfigured by operators to perform all possible duties, container-based runtimes create and destroy an operating environment as they're needed. Containers can rapidly deploy repeatable, self-sufficient runtime environments to execute workloads of limited scope and duration.

A cloud native CI/CD tool must leverage container technology for runtimes, to provision your build node environments with the tools and state they need to perform their work. This is a must for creating deterministic builds.

RUNTIMES OUT OF BOX

With runtimes so easy to manage, your cloud native should provide you with a core set of runtime container images that anticipate the most common workloads. In addition to commonly used CLIs, the set should be diverse enough to support key language types, and comprehensive enough for compatibility with several popular operating systems.

A well-considered set of runtimes will enable your teams to get started working right away.

CUSTOM RUNTIMES ENABLED

While a standard set of runtimes may fulfill most needs, you'll want to be able to create and deploy your own runtime containers for special needs. Containers are inherently easy to create from many OSS layer components; your CI/CD tool should be able to provision them to your build nodes using the same mechanisms as out-of-the-box runtimes.

When your runtime libraries are administered through container registries, IT administrators can redirect default registries to entirely customized sets, and govern which set of runtimes departments use through an invisible flick of a switch.

Container-based runtimes encourage specialization and division of labor across DevOps teams. Operations admins can keep runtimes current, while developers can concentrate on the steps of their builds. This specialization also enables security, as admins control what gets deployed into build nodes.

And even as container-based runtimes offer these advantages, a CI/CD solution should still permit any workload portion to run directly on a VM, to support special circumstances that might arise.

ORCHESTRATED BY KUBERNETES

Container-based runtimes enable your build nodes to be orchestrated through Kubernetes. Any cloud-native CI/CD should leverage K8s to manage your build node clusters for stability and scale.

OBSERVABLE



With their reliance on machine virtualization through containers, interfaces, and orchestration, cloud native systems are inherently opaque. The activity of a CI/CD server can be especially complex, with concurrent, interdependent tasks distributed across multiple computing planes. So any cloud native solution must provide human operators ways to ask questions about its state and get actionable answers.

REAL-TIME ACTIVITY

Your CI/CD should enable you to watch in real time the current activity of executing pipelines, presented in ways that are both rich in information and easy to understand. You will want to gain a holistic view of resource use, but also be able to focus on and examine key activities of interest.

You'll want to lift the lid on everything that's cooking, one at a time, and see how each is faring. That means being able to watch the output of any of your workloads as they happen. And you'll also want to monitor your CI/CD operations as a whole, to ensure that everyone continues to be served.

RICH LOGGING, MONITORING, AND ANALYSIS

Of course you'll need to capture all of that information in logs for future viewing and analysis. You'll need to remember what succeeded and failed, not just today, but last week, and figure out why. And you'll need to be able to share that data to collaborate on corrections and improvements. So all information that can be seen in real time should be captured and easily navigable by operators, developers, and all that may need it.

CAPTURING PERFORMANCE

A cloud native CI/CD solution should help operators to analyze and tune their configurations by monitoring resource usage over time. Operations teams that can see where and when usage of CPU, memory, build nodes and more face greater and lesser load will be better able to assure systems are fully performant. They'll also be able to see where resources are wasted, and help keep computing costs down.

CONNECTABLE FOR POWER

To expand your observability, your CI/CD should be able to export the data it captures to other tools for complex analytics. Your teams will be able to create their own dashboards and metrics, and even combine this data from other sources to tune all systems for maximum efficiency.

ENABLES CLOUD NATIVE DEVELOPMENT



Cloud native CI/CD can't just be cloud native, it must help you build cloud native. It should include the facilities you need to develop and produce your applications as containerized services that can be orchestrated in the cloud.

DOCKER SMART

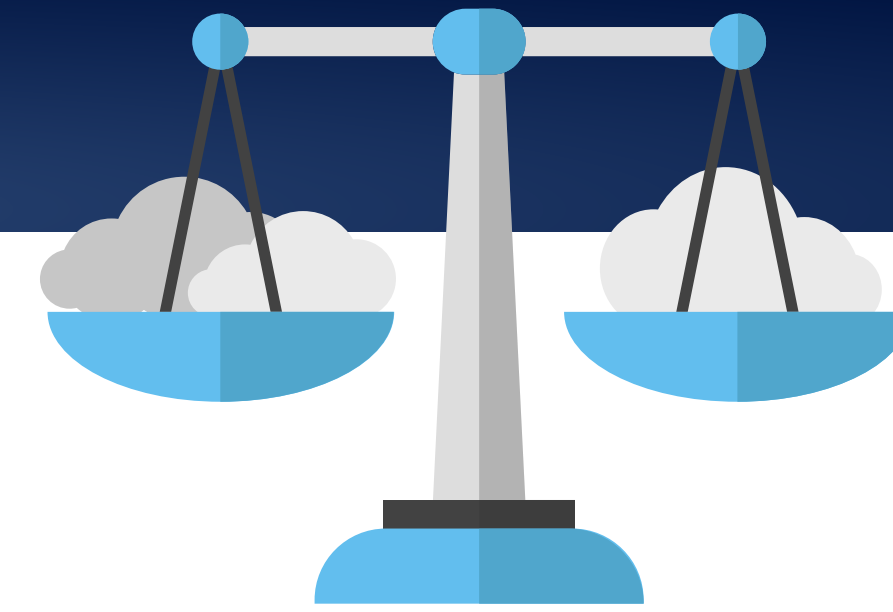
At minimum, your CI/CD needs to be Docker smart, ready out of the box to build and manage Docker images for containerized applications. You shouldn't need to separately install the Docker CLI and supporting tools into your build node environment or configure its state.

The declarative language used to configure your pipelines should enable Docker inherently through its command set. Developers should be able to specify what their container images should include and where to put them, then leave it to your CI/CD tool to load the Docker engine runtime, command it to build those images, and push them to registries.

KUBERNETES CONNECTED

Similarly, your cloud native CI/CD must be equally friendly for deploying those container images to Kubernetes as part of continuous delivery. It needs to be equipped to enable Helm, so you can reliably automate those deployments.

Declarative commands for managing Helm chart repositories and deploying to clusters through Helm charts enable developers to complete the end-to-end CI/CD process from build to delivery into test and production.

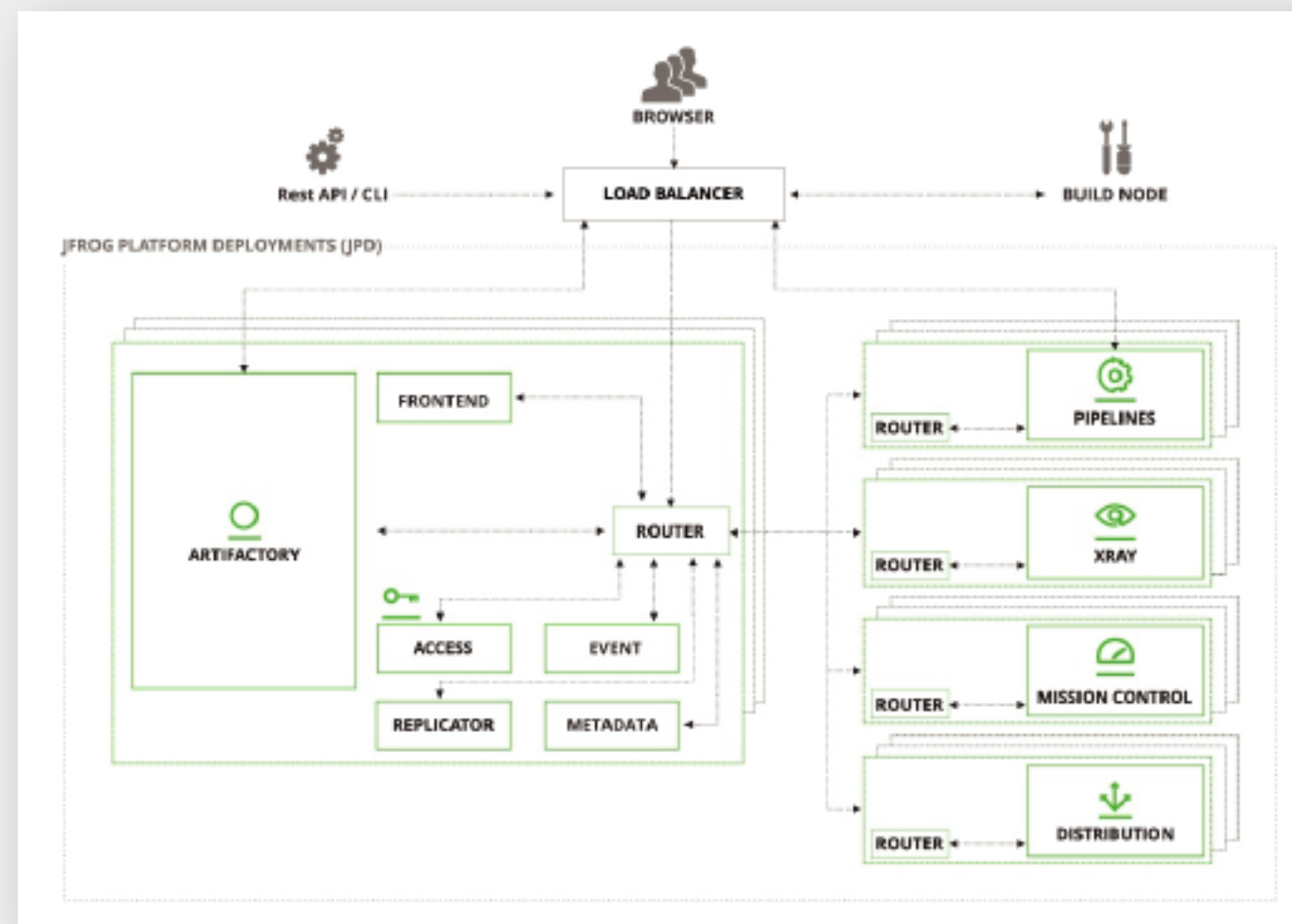


HOW JFROG MEASURES UP

These principles were our guideposts for creating JFrog Pipelines CI/CD, to make the best use of the cloud for creating a robust, elastic solution that can be available anytime, from anywhere.

BUILT ON CLOUD NATIVE STANDARDS

As part of the JFrog Platform, Pipelines relies on Docker and Kubernetes to deploy and orchestrate its own microservices.



INFRASTRUCTURE AGNOSTIC

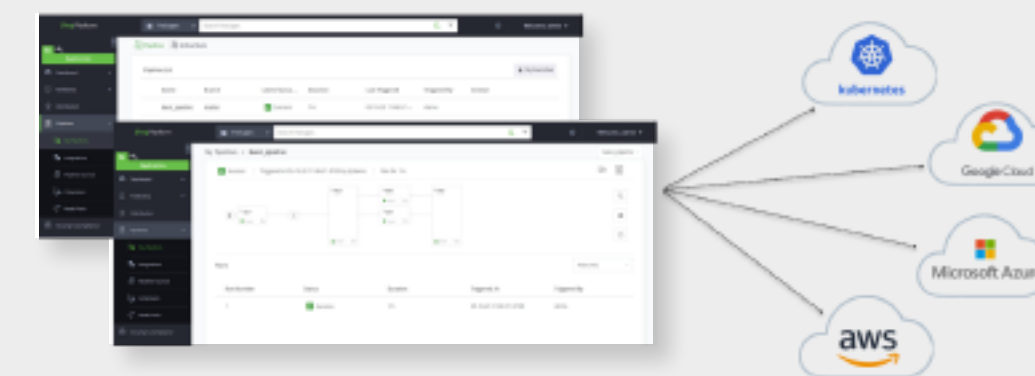
Pipelines CI/CD is available as a managed service on the major cloud providers.



Pipelines can be installed as a self-hosted system on any public cloud, on an on-prem cloud in your own datacenter, or a hybrid of on-prem and public clouds.



Once running, Pipelines operates with the same features and functionality on all platforms.



INFINITELY AND DYNAMICALLY SCALABLE

Through dynamic build nodes, Pipelines can fetch build resources from a cloud service or K8s when they're needed and release them when they're not.

The screenshot shows a table of build nodes with the following data:

Status	Name	Version	Workload	Created	Updated	Last check-in	Installed
Cached	node-1cc68a46	1.6.6		12-11-20 18:56:16 +0...	12-11-20 19:25:07 +0...	12-11-20 19:25:06 +0...	12-11-20 19:02:33 +0...
Cached	node-37a1d97f	1.6.6		12-11-20 18:56:07 +0...	12-11-20 19:24:06 +0...	12-11-20 19:24:05 +0...	12-11-20 19:02:34 +0...
Cached	node-5894e175	1.6.6		12-11-20 18:55:29 +0...	12-11-20 19:23:08 +0...	12-11-20 19:23:06 +0...	12-11-20 19:02:34 +0...

A single, central installation of Pipelines can drive many concurrent CI/CD pipelines at once.



CONFIGURED BY CODE

Pipelines uses a declarative DSL to define workflows that are based on YAML, “the language of cloud native” that also powers technologies like Kubernetes and Helm.

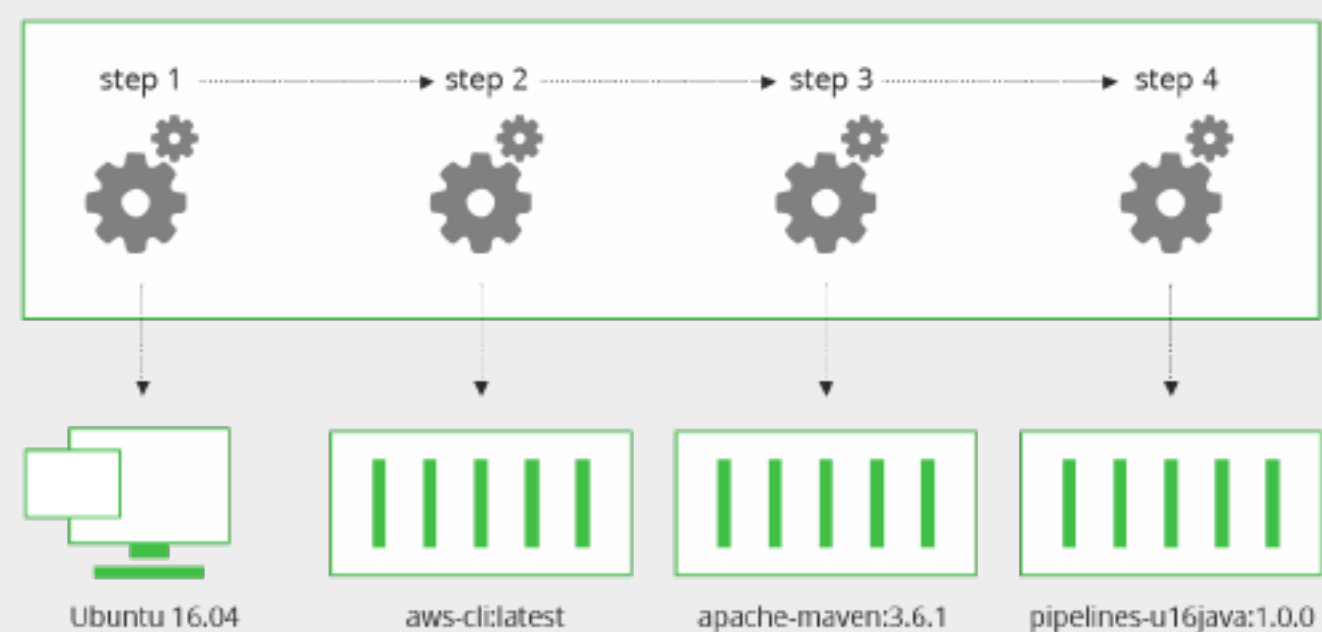
```

1 pipelines:
2   - name: gradle_build
3     configuration:
4       environmentVariables:
5         readOnly:
6           Version: 1.1.$run_number
7
8
9     steps:
10    - name: gradle_bld_svc_demo
11      type: GradleBuild
12      configuration:
13        runtime:
14          type: image
15          image:
16            custom:
17              # The Docker image is used to support a java 8 image for the gradle build
18              name: docker.bintray.io/jfrog/pipelines-w18java
19              tag: "8"
20          gradleCommand: clean artifactoryPublish -b build.gradle --stacktrace
21          sourceLocation: tutorial/step1-create-gradle-app
22          configFileLocation: *
23          configFileMame: jfrog-gradle.yml
24          autoPublishBuildInfo: true
25          inputResources:
26            - name: gitRepo_code
27          integrations:
28            - name: artifactory
29          outputResources:
30            - name: gradleBuildInfo
  
```

Pipelines’ native steps provide out-of-the-box access to most common CI/CD operations for building, promoting, and distributing artifacts.

CONTAINER RUN

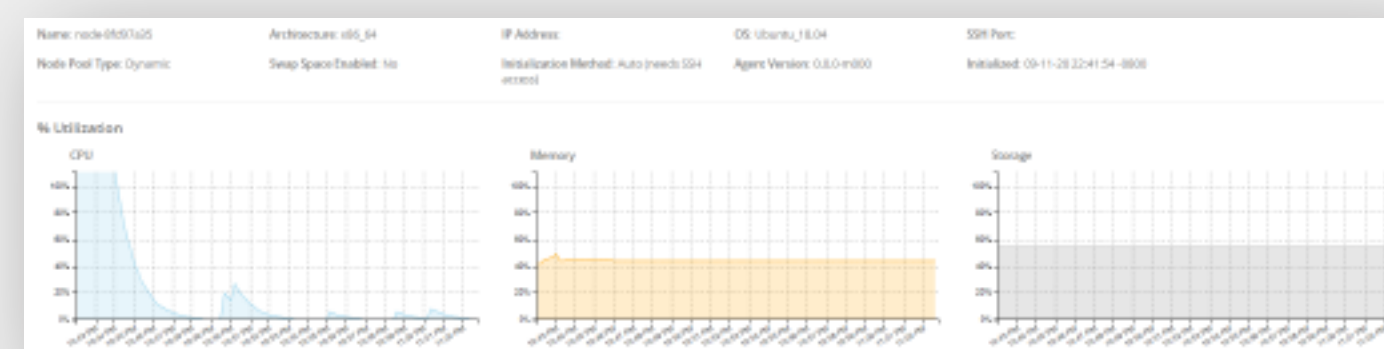
By default, Pipelines executes each step of your pipeline in an immutable runtime container on a build machine node. You can use the set of runtime Docker images Pipelines provides out of the box, or you can create your own custom runtime images. Or, if you need to, you can run any step directly on the host VM instead.



OBSERVABLE

Pipelines' rich workflow diagrams enable you to see how your steps connect and watch the progress of your pipelines as they execute. Pipelines captures a complete record of every run for you to examine.

The Pipelines UI through the JFrog Platform provides visibility into the allocation and activity of build machine nodes and their supporting resources.



Pipelines' comprehensive RESTful APIs enable querying and commanding your CI/CD server, to monitor resource utilization as well as automate configuration.

ENABLES CLOUD NATIVE DEVELOPMENT

Out-of-the-box integrations for essential cloud native tools like Docker registries and Kubernetes, enable you to deliver the apps you build to the cloud.

Pipelines makes it easy to build for Artifactory's many supported package types, including Docker. Native steps make Pipelines ready out of the box to build Docker images, push to Docker and Helm registries in Artifactory, and deploy to Kubernetes as part of continuous delivery.

Start for free:

jfrog.com/artifactory/start-free/



The Liquid Software Company