v1

CLOUD NATIVE
COMPUTING FOUNDATION

CLOUD NATIVE GLOSSARY

glossary.cncf.io

The CNCF Cloud Native Glossary Project is intended to be used as a reference for common terms used when talking about cloud native applications.

WORKING WITH DEFINITIONS

To make it easier to work on individual terms, we've moved them into individual files in the definitions folder. Each term falls under one of three categories: 1) technology, 2) property, or 3) concept.

To learn how to navigate this GitHub page and submit issues and/or PRs, please refer to the how-to guide. Before submitting a PR, please ensure you follow the style guide.

ACKNOWLEDGEMENTS

The Cloud Native Glossary was initiated by the CNCF Marketing Committee (Business Value Subcommittee), including contributions from Catherine Paganini, Chris Aniszczyk, Daniel Jones, Jason Morgan, Katelin Ramer and Mike Foster.

LICENSE

TABLE OF CONTENTS

API Gateway

TECHNOLOGY

WHAT IT IS

An API gateway is a tool that aggregates unique application APIs, making them all available in one place. It allows organizations to move key functions, such as authentication and authorization or limiting the number of requests between applications, to a centrally managed location. An API gateway functions as a common interface to (often external) API consumers.

PROBLEM IT ADDRESSES

If you're making APIs available to external consumers, you'll want one entry point to manage and control all access. Additionally, if you need to apply functionality on those interactions, an API gateway allows you to uniformly apply it to all traffic without requiring any app code changes.

HOW IT HELPS

Providing one single access point for various APIs in an application, API gateways make it easier for organizations to apply cross-cutting business or security logic in a central location. They also allow application consumers to go to a single address for all their needs. An API gateway can simplify operational concerns like security and observability by providing a single access point for requests to all web services in a system. As all requests flow through the API gateway, it presents a single place to add functionality like metrics-gathering, rate-limiting, and authorization.

Application Programming Interface (API)

TECHNOLOGY

WHAT IT IS

An API is a way for computer programs to interact with each other. Just as humans interact with a website via a web page, an API allows computer programs to interact with each other. Unlike human interactions, APIs have limitations on what can and cannot be asked of them. The limitation on interaction helps to create stable and functional communication between programs.

PROBLEM IT ADDRESSES

As applications become more complex, small code changes can have drastic effects on other functionality. Applications need to take a modular approach to their functionality if they can grow and maintain stability simultaneously. Without APIs, there is a lack of a framework for the interaction between applications. Without a shared framework, it is challenging for applications to scale and integrate.

HOW IT HELPS

APIs allow computer programs or applications to interact and share information in a defined and understandable manner. They are the building blocks for modern applications and they provide developers with a way to integrate applications together. Whenever you hear about microservices working together, you can infer that they interact via an API.

Autoscaling

PROPERTY

Autoscaling is the ability of a system to scale automatically, typically, in terms of computing resources. With an autoscaling system, resources are automatically added when needed and can scale to meet fluctuating user demands. The autoscaling process varies and is configurable to scale based on different metrics, such as memory or process time. Managed cloud services are typically associated with autoscaling functionality as there are more options and implementations available than most on-premise deployments.

Previously, infrastructure and applications were architected to consider peak system usage. This architecture meant that more resources were underutilized and inelastic to changing consumer demand. The inelasticity meant higher costs to the business and lost business from outages due to overdemand.

By leveraging the cloud, virtualizing, and containerizing applications and their dependencies, organizations can build applications that scale according to user demands. They can monitor application demand and automatically scale them, providing an optimal user experience. Take the increase in viewership Netflix experiences every Friday evening. Autoscaling out means dynamically adding more resources: for example, increasing the number of servers allowing for more video streaming and scaling back once consumption has normalized.

Blue green deployment

CONCEPT

WHAT IT IS

Blue-green deployment is a strategy for updating running computer systems with minimal downtime. The operator maintains two environments, dubbed "blue" and "green". One serves production traffic (the version all users are currently using), whilst the other is updated. Once testing has concluded on the non-active (green) environment, production traffic is switched over (often via the use of a load balancer). Note that blue-green deployment usually means switching the entire environments, comprising many services, all at once. Confusingly, sometimes the term is used with regard to individual services within a system. To avoid this ambiguity, the term "zero-downtime deployment" is preferred when referring to individual components.

PROBLEM IT ADDRESSES

Blue-green deployments allow minimal downtime when updating software that must be changed in "lockstep" owing to a lack of backwards compatibility. For example, blue-green deployment would be appropriate for an online store consisting of a website and a database that needs to be updated, but the new version of the database doesn't work with the old version of the website, and vice versa. In this instance, both need to be changed at the same time. If this was done on the production system, customers would notice downtime.

HOW IT HELPS

Blue-green deployment is an appropriate strategy for non-cloud native software that needs to be updated with minimal downtime. However, its use is normally a "smell" that legacy software needs to be re-engineered so that components can be updated individually.

Canary deployment

CONCEPT

WHAT IT IS

Canary deployments is a deployment strategy that starts with two environments: one with live traffic and the other containing the updated code without live traffic. The traffic is gradually moved from the original version of the application to the updated version. It can start by moving 1% of live traffic, then 10%, 25%, and so on, until all traffic is running through the updated version. Organizations can test the new version of the software in production, get feedback, diagnose errors, and quickly rollback to the stable version if necessary.

The term "canary" refers to the "canary in a coal mine" practice where canary birds were taken into coal mines to keep miners safe. If odorless harmful gases were present, the bird would die, and the miners knew they had to evacuate quickly. Similarly, if something goes wrong with the updated code, live traffic is "evacuated" back to the original version.

PROBLEM IT ADDRESSES

No matter how thorough the testing strategy, there are always some bugs discovered in production. Shifting 100% of traffic from one version of an app to another can lead to more impactful failures.

HOW IT HELPS

Canary deployments allow organizations to see how new software behaves in real-world scenarios before moving significant traffic to the new version. This strategy enables organizations to minimize downtime and quickly rollback in case of issues with the new deployment. It also allows more in-depth production application testing without a significant impact on the overall user experience.

Cloud computing

CONCEPT

WHAT IT IS

Cloud computing is a model that offers compute resources like CPU, network, and disk capabilities on-demand over the internet. Cloud computing gives users the ability to access and use computing power in a remote physical location. Cloud providers like AWS, GCP, Azure, DigitalOcean, and others all offer third parties the ability to rent access to compute resources in multiple geographic locations.

PROBLEM IT ADDRESSES

Organizations traditionally faced two main problems when attempting to expand their use of computing power. They either acquire, support, design, and pay for facilities to host their physical servers and network or expand and maintain those facilities. Cloud computing allows organizations to outsource some portion of their computing needs to another organization.

HOW IT HELPS

Cloud providers offer organizations the ability to rent compute resources on-demand and pay for usage. This allows for two major innovations: organizations can try things without wasting time planning and spending CAPEX on new physical infrastructure and they can scale as needed and on-demand. Cloud computing allows organizations to adopt as much or as little infrastructure as they need.

Cloud native application

CONCEPT

WHAT IT IS

Cloud native applications are specifically designed to take advantage of innovations in cloud computing. These applications integrate easily with their respective cloud architectures, taking advantage of the cloud's resources and scaling capabilities. It also refers to applications that take advantage of innovations in infrastructure driven by cloud computing. Cloud native applications today include apps that run in a cloud provider's datacenter and on cloud native platforms on-premise.

PROBLEM IT ADDRESSES

Traditionally, on-premise environments provided compute resources in a fairly bespoke way. Each datacenter had services that tightly coupled applications to specific environments, often relying heavily on manual provisioning for infrastructure, like virtual machines and services. This, in turn, constrained developers and their applications to that specific datacenter. Applications that weren't designed for the cloud couldn't take advantage of a cloud environment's resiliency and scaling capabilities. For example, apps that require manual intervention to start correctly cannot scale automatically, nor can they be automatically restarted in the event of a failure.

HOW IT HELPS

While there is no "one size fits all" path to cloud native applications, they do have some commonalities. Cloud native apps are resilient, manageable, and aided by the suite of cloud services that accompany them. The various cloud services enable a high degree of observability, enabling users to detect and address issues before they escalate. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

Container

TECHNOLOGY

WHAT IT IS

A container is a running process with resource and capability constraints managed by a computer's operating system. The files available to the container process are packaged as a container image. Containers run adjacent to each other on the same machine, but typically the operating system prevents the separate container processes from interacting with each other.

PROBLEM IT ADDRESSES

Before containers were available, separate machines were necessary to run applications. Each machine would require its own operating system, which takes CPU, memory, and disk space, all for an individual application to function. Additionally, the maintenance, upgrade, and startup of an operating system is another significant source of toil.

HOW IT HELPS

Containers share the same operating system and its machine resources, spreading the operating system's resource overhead and creating efficient use of the physical machine. This capability is only possible because containers are typically limited from being able to interact with each other. This allows many more applications to be run on the same physical machine.

There are limitations, however. Since containers share the same operating system, processes can be considered less secure than alternatives. Containers also require limits on the shared resources. To guarantee resources, administrators must constrain and limit memory and CPU usage so that other applications do not perform poorly.

Continuous delivery

CONCEPT

WHAT IT IS

Continuous delivery, often abbreviated as CD, is a set of practices in which code changes are automatically deployed into an acceptance environment (or, in the case of continuous deployment, into production). CD crucially includes procedures to ensure

that software is adequately tested before deployment and provides a way to rollback changes if deemed necessary. Continuous integration (CI) is the first step towards continuous delivery (i.e., changes have to merge cleanly before being tested and deployed).

PROBLEM IT ADDRESSES

Deploying reliable updates becomes a problem at scale. Ideally, we'd deploy more frequently to deliver better value to end-users. However, doing it manually translates into high transaction costs for every change. Historically, to avoid these costs, organizations have released less frequently, deploying more changes at once and increasing the risk that something goes wrong.

HOW IT HELPS

CD strategies create a fully automated path to production that tests and deploys the software using various deployment strategies such as canary or blue-green releases. This allows developers to deploy code frequently, giving them peace of mind that the new revision has been tested. Typically, trunk-based development is used in CD strategies as opposed to feature branching or pull requests.

Continuous integration (CI)

CONCEPT

WHAT IT IS

Continuous integration, often abbreviated as CI, is the practice of integrating code changes as regularly as possible. CI is a prerequisite for continuous delivery (CD). Traditionally, the CI process begins when code changes are committed to a source control system (Git, Mercurial, or Subversion) and ends with a tested artifact ready to be consumed by a CD system.

PROBLEM IT ADDRESSES

Software systems are often large and complex, with numerous developers maintaining and updating them. Working in parallel on different parts of the system, these developers may make conflicting changes and inadvertently break each other's work. Additionally, with multiple developers working on the same project, any everyday tasks such as testing and calculating code quality would need to be repeated by each developer, wasting time.

HOW IT HELPS

CI software automatically checks that code changes merge cleanly whenever a developer commits a change. It's a near-ubiquitous practice to use the CI server to run code quality checks, tests, and even deployments. As such, it becomes a concrete implementation of quality control within teams. CI allows software teams to turn every code commit into either a concrete failure or a viable release candidate.

DevOps

CONCEPT

WHAT IT IS

DevOps is a methodology in which teams own the entire process from application development to production operations, hence DevOps. It goes beyond implementing a set of technologies and requires a complete shift in culture and processes. DevOps calls for groups of engineers that work on small components (versus an entire feature), decreasing handoffs – a common source of errors.

PROBLEM IT ADDRESSES

Traditionally, in complex organizations with tightly-coupled monolithic apps, work was generally fragmented between multiple groups. This led to numerous handoffs and long lead times. Each time a component or update was ready, it was placed in a queue for the next team. Because individuals only worked on one small piece of the project, this approach led to a lack of ownership. Their goal was to get the work to the next group, not deliver the right functionality to the customer — a clear misalignment of priorities.

By the time code finally got into production, it went through so many developers, waiting in so many queues that it was difficult to trace the origin of the problem if the code didn't work. DevOps turns this approach upside down.

HOW IT HELPS

Having one team own the entire lifecycle of an application results in minimized handoffs, reduce risk when deploying into production, better code quality as teams are also responsible for how code performs in production and increased employee satisfaction due to more autonomy and ownership.

DevSecOps

CONCEPT

WHAT IT IS

The term DevSecOps refers to a cultural merger of the development, operational, and security responsibilities. It extends the DevOps approach to include security priorities with minimal to no disruption in the developer and operational workflow. Like DevOps, DevSecOps is a cultural shift, pushed by the technologies adopted, with unique adoption methods.

PROBLEM IT ADDRESSES

DevOps practices include continuous integration and continuous deployment and accelerate application development and release cycles. Unfortunately, automated release processes that fail to represent all organizational stakeholders adequately can exacerbate existing issues. A process that rapidly releases new software without considering security needs can degrade an organizations' security posture.

HOW IT HELPS

DevSecOps focuses on breaking down team silos and promotes the creation of secure, automated workflows. When selecting security applications, organizations must take advantage of automated CI/CD workflows and policy enforcement that empower the developer. The goal is not to be a blocker but to enforce security policies while giving users accurate information on how to move their project forward. When properly implemented, an organization will gain better team communication and reduce security mishaps and associated costs.

Distributed systems

CONCEPT

WHAT IT IS

Canary deployments is a deployment strategy that starts with two environments: one with live traffic and the other containing
A distributed system is a collection of autonomous computing elements connected over a network that appears to users as a single coherent system. Generally referred to as nodes, these components can be hardware devices (e.g. computers, mobile phones) or software processes. Nodes are programmed to achieve a common goal and, to collaborate, they exchange messages over the network.

PROBLEM IT ADDRESSES

Numerous modern applications today are so big they'd need supercomputers to operate. Think Gmail or Netflix. No single computer is powerful enough to host the entire application. By connecting multiple computers, compute power becomes nearly limitless. Without distributed computing, many applications we rely on today wouldn't be possible.

Traditionally, systems would scale vertically. That's when you add more CPU or memory to an individual machine. Vertical scaling is time-consuming, requires downtime, and reaches its limit quickly.

HOW IT HELPS

Distributed systems allow for horizontal scaling (e.g. adding more nodes to the system whenever needed). This can be automated allowing a system to handle a sudden increase in workload or resource consumption.

A non-distributed system exposes itself to risks of failure because if one machine fails, the entire system fails. A distributed system can be designed in such a way that, even if some machines go down, the overall system can still keep working to produce the same result.

Immutable infrastructure

PROPERTY

Immutable Infrastructure refers to computer infrastructure (virtual machines, containers, network appliances) that cannot be changed once deployed. This can be enforced by an automated process that overwrites unauthorized changes or through a system that won't allow changes in the first place. Containers are a good example of immutable infrastructure because persistent changes to containers can only be made by creating a new version of the container or recreating the existing container from its image.

By preventing or identifying unauthorized changes, immutable infrastructures make it easier to identify and mitigate security risks. Operating such a system becomes a lot more straightforward because administrators can make assumptions about it. After all, they know no one made mistakes or changes they forgot to communicate. Immutable infrastructure goes hand-in-hand with infrastructure as code where all automation needed to create infrastructure is stored in version control (e.g. Git). This combination of immutability and version control means that there is a durable audit log of every authorized change to a system.

Infrastructure as code

CONCEPT

WHAT IT IS

Infrastructure as code is the practice of storing the definition of infrastructure as one or more files. This replaces the traditional model where infrastructure as a service is provisioned manually, usually through shell scripts or other configuration tools.

PROBLEM IT ADDRESSES

Building applications in a cloud native way requires infrastructure to be disposable and reproducible. It also needs to scale on-demand in an automated and repeatable way, potentially without human intervention. Manual provisioning cannot meet the responsiveness and scale requirements of cloud native applications. Manual infrastructure changes are not reproducible, quickly run into scale limits, and introduces misconfiguration errors.

HOW IT HELPS

By representing the data center resources such as servers, load balancers, and subnets as code, it allows infrastructure teams to have a single source of truth for all configurations and also allows them to manage their data center in a CI/CD pipeline, implementing version control and deployment strategies.

Kubernetes

TECHNOLOGY

WHAT IT IS

Kubernetes, often abbreviated as K8s, is a popular open-source tool for modern infrastructure automation. It's like a data center operating system that manages applications running across a distributed system (just like the OS on your laptop that manages your apps).

Kubernetes schedules containers across nodes in a cluster. It bundles several infrastructure constructs, sometimes referred to as "primitives," like an instance of an app, load balancers, persistent storage, and others together in a way that they can be composed into applications.

Kubernetes enables automation and extensibility, allowing users to deploy applications declaratively in a reproducible way. Software products and projects in the Kubernetes ecosystem take advantage of that automation and extensibility to extend the Kubernetes API. This enables them to leverage Kubernetes' automation and make their tools more accessible to experienced Kubernetes practitioners.

PROBLEM IT ADDRESSES

Infrastructure automation and declarative configuration management have been important concepts for a long time and have only become more pressing as cloud computing has gained popularity. As demand for compute resources increases and organizations feel pressured to provide more operational capabilities with fewer engineers, new technologies and working methods are required to meet that demand. Additionally, the rise of cloud computing was paired with containerization and organizations that were busy automating more traditional infrastructure needed a mechanism to automate the configuration and deployment of their containers.

HOW IT HELPS

Kubernetes helps with automation in a manner similar to traditional infrastructure as code tools but has the advantage of working with containers that are more resistant to configuration drift than virtual or physical machines. Kubernetes works declaratively, which means that instead of operators providing the instructions about how to do something they instead describe, usually as a YAML document, what they want to be done; Kubernetes will take care of the "how" on its own. This results in Kubernetes being extremely compatible with infrastructure as code.

Microservices

CONCEPT

WHAT IT IS

Microservices are a modern approach to application development that takes advantage of cloud native technologies. While modern applications, like Netflix, appear to be a single app, they are actually a collection of smaller services – all closely working together. For instance, a single page that allows you to access, search, and preview videos is likely powered by smaller services that each handle one aspect of it (e.g. search, authentication, and running previews in your browser). In short, microservices refer to an application architecture pattern usually contrasted with monolithic applications.

PROBLEM IT ADDRESSES

Microservices are a response to challenges posed by monolithic applications. Generally, different parts of an application will need to be scaled separately. An online store, for example, is going to have more product views than checkouts. That means you'll need more copies of the product view functionality running than the checkout. In a monolithic application, those bits of logic can't be deployed separately. If you can't scale the product functionality individually, you'll have to duplicate the entire app with all other components you don't need – an inefficient use of resources. Monolithic applications also make it easy for developers to succumb to design pitfalls. Because all the code is in one place, it is easier to make that code tightly-coupled and harder to enforce the principle of separation of concerns. Monoliths often require developers to understand the entire codebase before they can be productive.

HOW IT HELPS

Separating functionality into different microservices makes them easier to deploy, update, and scale independently. By allowing different teams to focus on their own small part of a bigger application you also make it easier for them to work on their apps without negatively impacting the rest of the organization. While microservices solve many problems, they also create operational overhead—the things you need to deploy and keep track of increase by order of magnitude or more. Many cloud-native technologies aim to make microservices easier to deploy and manage.

Observability

PROPERTY

Observability is a characteristic of an application that refers to how well a system's state or status can be understood from its external outputs. Computer systems are measured by observing CPU time, memory, disk space, latency, errors, etc. The more observable a system is, the easier it is to understand how it's doing by looking at it.

The observability of a system has a significant impact on its operating cost. Observable systems yield meaningful, actionable data to their operators, allowing them to achieve favorable outcomes and less downtime. Note that more information does not necessarily translate into a more observable system. In fact, sometimes the amount of information generated by a system can make it harder to identify valuable health signals from the noise generated by the application. Observability requires the right data to make the right decisions.

Reliability

PROPERTY

From a cloud native perspective, reliability refers to how well a system responds to failures. If we have a distributed system that keeps working as infrastructure changes and individual components fail, it is reliable. On the other hand, if it fails easily and operators need to intervene manually to keep it running, it is unreliable. The goal of cloud native applications is to build inherently reliable systems.

Scalability

PROPERTY

Scalability refers to how well a system can grow. That is increasing the ability to do whatever the system is supposed to do. For example, a Kubernetes cluster scales by increasing or reducing the number of containerized apps, but that scalability depends on several factors. How many nodes does it have, how many containers can each node handle, and how many records and operations can the control plane support?

A scalable system makes it easy to add more capacity. We differentiate between two scaling approaches. On the one hand, there is horizontal scaling which adds more nodes to handle increased load. In contrast, in vertical scaling individual nodes are made more powerful to perform more transactions (e.g. by adding more memory or CPU to an individual machine). A scalable system is able to change easily and meet user needs.

Self-Healing

PROPERTY

A self-healing system is capable of recovering from certain types of failure without any human intervention. It has a "convergence" or "control" loop that actively looks at the system's actual state and compares it to the state that the operators initially desired. If there is a difference (e.g., fewer application instances are running than desired), it will take corrective action (e.g., start new instances).

Service discovery

CONCEPT

WHAT IT IS

Service discovery is the process of finding individual instances that make up a service. A service discovery tool keeps track of the various nodes or endpoints that make up a service.

THE PROBLEM IT ADDRESSES

Cloud native architectures are dynamic and fluid, meaning they are constantly changing. A containerized app will likely end up starting and stopping multiple times in its lifetime. Each time that happens, it will have a new address and any app that wants to find it needs a tool to provide the new location information.

HOW IT HELPS

Service discovery keeps track of apps within the network so they can find one another when needed. It provides a common place to find and potentially identify individual services. Service discovery engines are database-like tools that store info about what services exist and how to locate them.

Service mesh

TECHNOLOGY

WHAT IT IS

In a microservices world, apps are broken down into multiple smaller services (app components) that communicate over a network. Just like your wifi network, computer networks are intrinsically unreliable, hackable, and often slow. Service meshes address this new set of challenges by managing traffic (i.e., communication) between services and adding reliability, observability, and security features uniformly across all services.

THE PROBLEM IT ADDRESSES

Having moved to a microservices architecture, engineers are now dealing with hundreds, possibly even thousands of individual services, all in need to communicate. That means a lot of traffic is going back and forth over the network. On top of that, individual applications may need to encrypt communications to support regulatory requirements, provide common metrics to operations teams, or provide detailed insight into traffic to help diagnose issues. If built into the individual applications, each one of these features will cause friction between teams and slow down development of new features.

HOW IT HELPS

Service meshes add reliability, observability, and security features uniformly across all services across a cluster without requiring code changes. Before service meshes, that functionality had to be encoded into every single service, becoming a potential source of bugs and technical debt.

Service proxy

TECHNOLOGY

WHAT IT IS

A service proxy intercepts traffic to or from a given service, applies some logic to it, then forwards that traffic to another service. It essentially acts as a "go-between" that collects information about network traffic and/or applies rules to it.

THE PROBLEM IT ADDRESSES

To keep track of service to service communication (aka network traffic) and potentially transform or redirect it, we need to collect data. Traditionally, the code enabling data collection and network traffic management was embedded within each application.

HOW IT HELPS

A service proxy allows us to "externalize" this functionality. No longer does it need to live within the apps. Instead, it's now embedded into the platform layer (where your apps run).

Acting as gatekeepers between services, proxies provide insight into what type of communication is happening. Based on their insight, they determine where to send a particular request or even deny it entirely.

Proxies gather critical data, manage routing (spreading traffic evenly among services or rerouting if some services break down), encrypt connections, and cache content (reducing resource consumption).infrastructure, and changes that occur to systems outside IaC tools persist in the environment. That may lead to unexpected behaviors, security vulnerabilities, and potentially break the automation.

Virtualization

TECHNOLOGY

WHAT IT IS

Virtualization, in the context of cloud native computing, refers to the process of taking a physical computer, sometimes called a server, and allowing it to run multiple isolated operating systems. Those isolated operating systems and their dedicated compute resources (CPU, memory, and network) are referred to as virtual machines or VMs. When we talk about a virtual machine, we're talking about a software-defined computer. Something that looks and acts like a real computer but is sharing hardware with other virtual machines. As an example, you can lease a "computer" from AWS – that computer is actually a VM.

PROBLEM IT ADDRESSES

Virtualization addresses a number of problems, including the improvement of physical hardware usage by allowing more apps to run on the same physical machine whilst still being isolated from each other for security.

HOW IT HELPS

Apps running on virtual machines have no awareness that they are sharing a physical computer. Virtualization also allows the users of the datacenter to spin up a new "computer" (aka a VM) in minutes without worrying about the physical constraints of adding a new computer to a datacenter. VMs also enable users to speed up the time to get a new virtual computer.