emma

Design Principles for Multi-cloud Interoperability

Contents

Introduction

Cloud computing platforms are not one-size-fits-all. That's why an overwhelming majority of organizations utilize more than one kind of cloud and cloud service provider. Ideally, workloads running on any cloud should be able to communicate and collaborate with each other. This ability of different systems and modules to exchange data with each other across clouds is known as multi-cloud interoperability. It's one of the very basic attributes of cloud native systems. Without it, much of the cloud's scalability and agility would be lost.

In our latest white paper, we discuss the key principles for ensuring interoperability in diverse multi-cloud environments, which is not only a prerequisite for optimal cloud utilization but also for fostering an ethical and free market environment.

Siloed vs. Interoperable Multi-cloud: Why Interoperability Matters?

Classically, a multi-cloud was any environment that utilized services from more than one cloud provider. However, modern multi-cloud deployments have an increasingly diverse portfolio of cloud services as well as cloud types, including on-premise, private, and public clouds, all well-integrated to operate holistically as a single entity. This shift from a disjointed multi-cloud to an integrated, interoperable multi-cloud offers numerous benefits:

1. Seamless deployment, migration, and communication across all clouds
2. Localized cloud benefits and cost optimization
3. Horizontal scalability
4. Cross-cloud redundancy and disaster recovery
5. Unified view and control of security and management

emma

Understanding Multi-cloud Interoperability

Before jumping into the design principles of an interoperable multi-cloud, it's essential to establish a clear distinction between interoperability and portability. Often used interchangeably, they are, in fact, distinct concepts with nuanced but significant differences. In the context of a multi-cloud:

1. **Interoperability is the ability of two systems, components, or services running on separate clouds to work together and communicate with each other effectively without significant modifications.**
2. **Portability is the ability of applications and data to migrate from one cloud to another without running into errors or losing functionality on the other platform.**

The concept of holistic interoperability has several layers to it.

Technical Interoperability

The behavior of interconnected applications or components must be consistent and predictable when integrated across clouds. Additionally, the protocols used for data transmission must be reliable and secure across clouds.

Semantic Interoperability

The data exchanged between multi-cloud applications or services must be understood and utilized correctly.

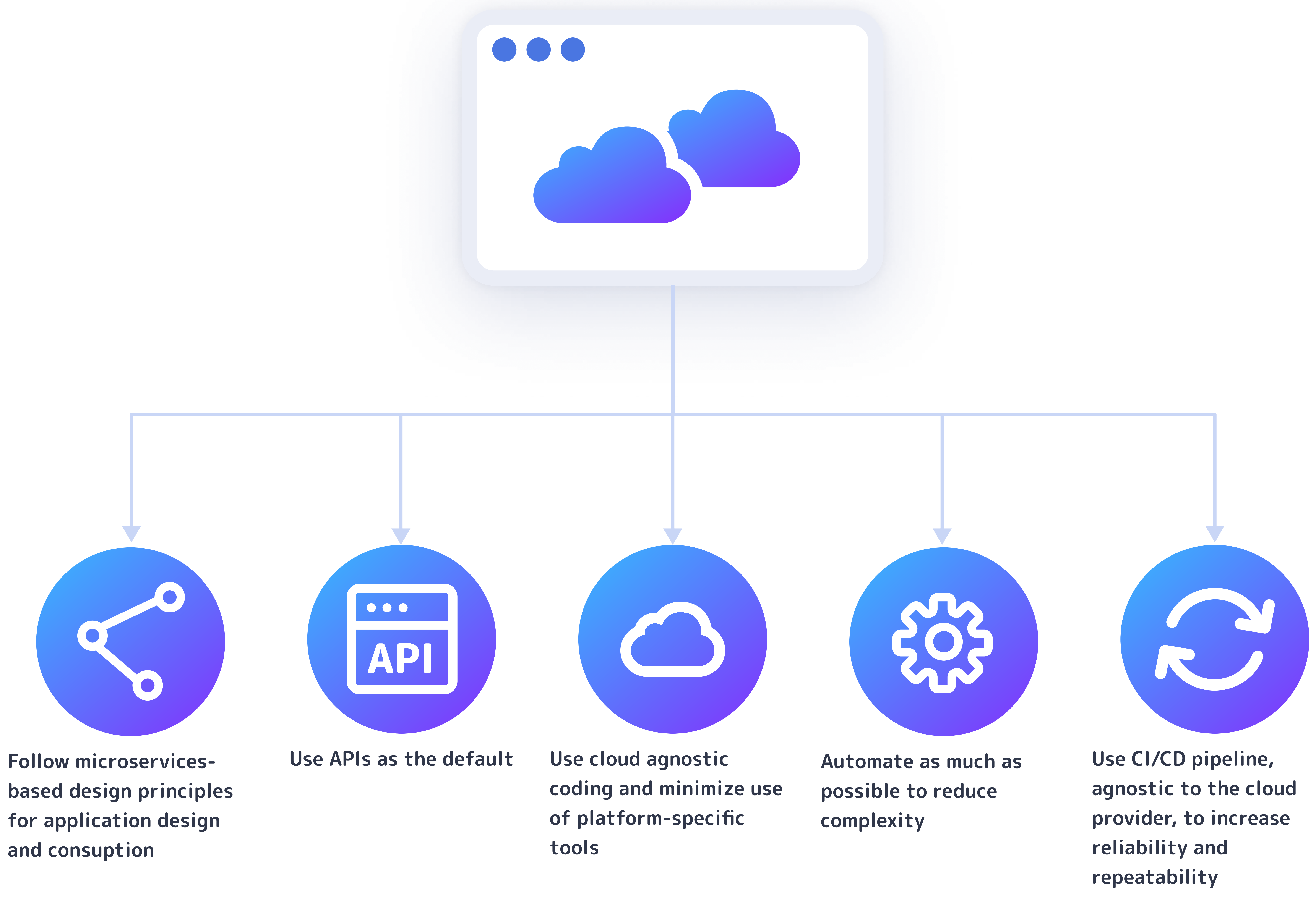
Policy Interoperability

The rules, permissions, and access control policies for using cloud resources must be consistent and compatible across clouds to ensure adherence to organizational policies.

Collectively, these layers help achieve different aspects of interoperability in a multi-cloud ecosystem.

Designing the Multi-cloud for Interoperability

Interoperability is a fundamental attribute of the modern multi-cloud, fostering a culture of fair competition, faster innovation, and social responsibility. Below are the basic principles that ensure interoperability in data, applications, and services. As such, they are all tightly knit and go hand-in-hand in enabling multi-level interoperability.


API
Follow microservices-based design principles for application design and consuption
Use APIs as the default
Use cloud agnostic coding and minimize use of platform-specific tools
Automate as much as possible to reduce complexity
Use CI/CD pipeline, agnostic to the cloud provider, to increase reliability and repeatability

Microservices-based Design

A microservices-based design decomposes complex applications into small, loosely coupled, and modular components, each focusing on a single functionality. Each microservice can be reused, deployed, and scaled individually, reducing dependencies, complexity, and the need for rewriting the same code repeatedly. However, there are several strategies and technologies that enable flexible, adaptable, and independent microservices that are interoperable across any cloud.

Standardized Interfaces

Microservices must communicate with each other through standardized and well-defined APIs and protocols. APIs must use standard communication protocols, like HTTP/HTTPS for RESTful APIs or gRPC for efficient remote procedure calls. This ensures that microservices can communicate regardless of their technology stack, programming language, or underlying platform. Standardized APIs promote interoperability by exposing the functionality and data of each service in a consistent way, which allows microservices to understand and use each other's functionality even when hosted on different clouds.

Containerization and Orchestration

Containers complement microservices and enhance their portability and interoperability by encapsulating service code and its dependencies, such as the runtime, libraries, and system tools, into a lightweight, isolated, self-contained, and executable package that can run consistently across any development, test, or production environment on any cloud. All major CSPs offer tailored container deployment and orchestration services, like AWS Fargate, Amazon ECS (Elastic Container Service), and Azure Container Instances (ACI). However, in order to ensure multi-cloud interoperability, containers must be deployed via a standard, vendor-neutral container engine, such as Docker, and orchestrated through platform-agnostic tools like Kubernetes.

Service Meshes

Service meshes, like the open-source and platform-agnostic Istio and Linkerd, provide advanced features and infrastructure abstractions for handling service-to-service communication, service discovery, load balancing, security, and observability between microservices. Service meshes can enhance the interoperability of microservices by assuming responsibility for network-level services and communication logic, thus boosting standardization and consistency in how microservices interact with one another.

Event-driven Architecture

Event-driven architecture is a pattern that enables microservices to communicate asynchronously in response to events. This means that each microservice can operate independently without waiting for the sequential execution of other tasks. Event brokers and platforms like Apache Kafka® and RabbitMQ are responsible for distributing, storing, and managing events, which decouples event-producing and event-consuming microservices. This decoupling enhances interoperability among distributed microservices, as they are no longer directly dependent on other microservices to perform their respective functions.

Application Programming Interfaces (APIs)

Standardized communication is crucial for achieving interoperability in a multi-cloud environment. APIs define how services and applications, with varying underlying languages and technology stacks, communicate with each other consistently. They serve as a layer of abstraction between applications and cloud-specific services — instead of communicating directly with cloud-specific services, applications can have their own standardized and generic APIs, which can then call the cloud-specific API. This abstraction allows applications to communicate with similar services from different clouds without needing significant modifications.

However, there are certain prerequisites for ensuring that the APIs are broadly compatible and interoperable.

Open Standards and Protocols

APIs should conform to widely accepted standards and protocols, for instance, adhering to RESTful principles, which involve utilizing JSON or XML for data representation and using HTTP/HTTPS to define actions (GET, POST, PUT, etc.) on resources. In addition, following practices like defining standardized endpoints and resource-oriented URIs ensures consistency, understandability, and usability of the API across clouds.

API Documentation

APIs must be accompanied by detailed documentation providing information on endpoints, request/response formats, input parameters, and error-handling mechanisms. Well-documented APIs are easier for developers to understand and use across different platforms effectively. Tools like OpenAPI Specification (OAS) or Swagger can help create machine-readable documentation for ensuring better standardization, enabling automated integration, and facilitating consistent and error-free interactions between systems and services across any cloud platform.

API Gateways

API gateways are servers or services that provide a centralized point for managing and controlling the flow of requests and responses between microservices. They standardize inter-service communication by translating between different protocols and data formats. They also handle request routing, security, and authentication, thus abstracting the complexity of the underlying microservices from API consumers. This standardization and abstraction offers compatibility and interoperability between services supporting different technologies and platforms.

API Versioning

Most services with published APIs will need updates and modifications that will require changes to the API. It is imperative to implement versioning in service APIs to ensure consistency and backward compatibility. This ensures interoperability between systems that may be using different versions of the API. It also allows seamless adaptability across multi-cloud platforms where different clouds may have varying update schedules and requirements. Tools like Swagger, OpenAPI, and Postman allow version information in their API documentation. Similarly, many API gateways and management tools offer built-in support for versioning.

Automation

Automation is the use of tools and scripts to streamline workflows and enable actions, decisions, and responses without manual intervention. In a multi-cloud environment, automation requires standardization of infrastructure, configurations, deployments, and operations across different cloud environments. Certain cloud-specific details must also be abstracted in order to maintain a common language and approach to managing applications and resources in diverse cloud environments.

Automation, through standardization, consistency, and abstraction, enables and enhances interoperability in multi-cloud. Several aspects of cloud configuration and management can benefit from automation as it boosts reliability, efficiency, and interoperability.

Infrastructure as Code (IaC)

IaC abstracts the underlying infrastructure details and specifications, defining it in a version-controlled and machine-readable format that is consistent and reproducible across different cloud environments. Tools like Terraform and Puppet enable organizations to define and provision cloud resources consistently across multiple clouds.

Configuration Management

Configuration management tools like Ansible, Chef, and Puppet abstract the underlying cloud-specific details, enabling organizations to define cloud-agnostic configuration playbooks for configuring servers, applications, and infrastructure resources. These playbooks can then be run across any cloud to automate configurations and yield consistent results.

Container Orchestration

Container orchestration tools like Kubernetes automate the deployment and scaling of containerized applications. They abstract cloud-specific details and provide a consistent way to deploy and manage containers across different clouds, making it easier to achieve interoperability in multi-cloud.

Continuous Integration and Continuous Deployment (CI/

CI/CD pipelines are a key part of the DevOps strategy. They automate build, testing, and deployment of applications and services for faster and more reliable software delivery. Carefully implemented CI/CD pipelines support interoperability by promoting consistency and enabling automated, early detection of compatibility issues that can hamper interoperability.

Cloud Agnosticism

Cloud agnosticism ensures that an application or system remains independent of a specific cloud provider. Cloud agnosticism, as a design and operational approach, can significantly boost interoperability through open standards, abstractions, and consistency in deployment. As such, it requires avoiding vendor-specific tools and services and embracing open standards and cloud agnostic tools and technologies at multiple layers of an application and system design, development, and deployment.

Code and Development

At its core, interoperability demands that applications be written using languages and frameworks that are cloud agnostic. This means developers should avoid cloud-specific APIs, SDKs, and services and opt for open-source libraries and tools that work consistently across cloud platforms. Using RESTful APIs, gPRC, and other widely accepted protocols for application interactions promotes interoperability.

Infrastructure

At the infrastructure level, cloud agnosticism involves designing applications to run and communicate consistently on any cloud's virtual machines or containers. Vendor-neutral IaC tools like Terraform and Ansible can be used to define cloud agnostic infrastructure configurations, thus reducing dependencies on cloud-specific features and enabling applications to communicate and collaborate seamlessly in a distributed, multi-cloud setting.

Data Storage

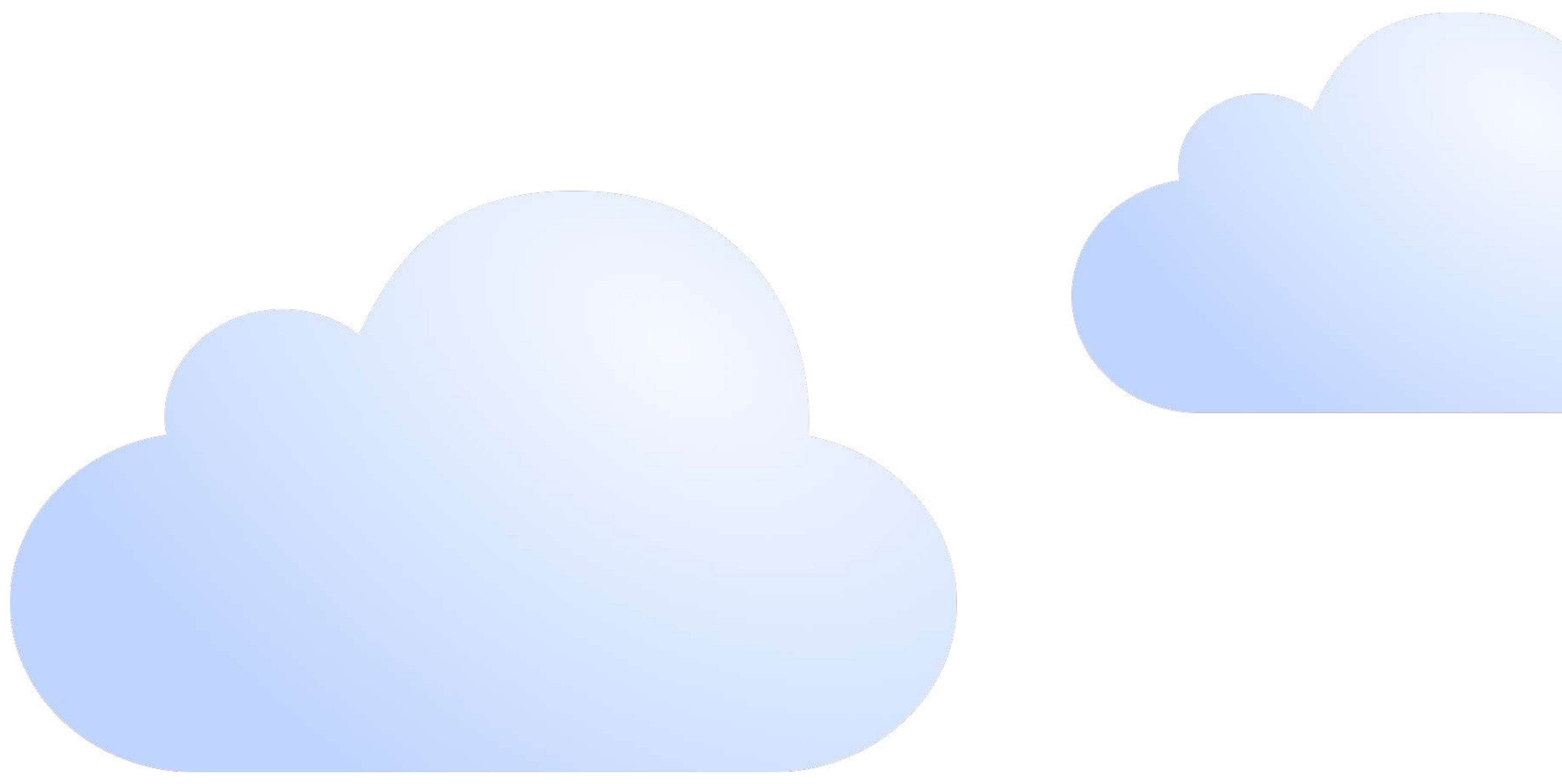
To ensure interoperability, data storage solutions should not be tightly tied to any particular cloud provider. For instance, instead of using AWS DynamoDB as a No-SQL database, organizations should opt for an open-source distribution like MongoDB or Redis, which are widely supported and can interact with applications across different platforms without code refactoring.

Cloud Services

In addition to databases and IaC tools and services, organizations should opt for open-source and/or vendor-neutral versions of any managed offerings they need, for instance, choosing Kubernetes instead of AWS EKS, Azure Kubernetes Service (AKS), or Google Kubernetes Engine (GKE). This ensures that containerized applications across different clouds remain interoperable. In cases that necessitate the use of a proprietary service, the service should be consumed via API abstractions.

Management Tools

Major cloud providers are now offering multi-cloud platforms, like Google Anthos, Azure ARC, and IBM Cloud Satellite, but they often restrict organizations to the clouds and environments they support. For instance, Google Anthos offers limited functionality when it comes to managing clusters in Microsoft AKS and Amazon EKS. To fully use Anthos, administrators must migrate their workloads to Google's Kubernetes engine. Such limitations make multi-cloud management platforms from hyperscalers less interoperable as opposed to cloud agnostic multi-cloud management platforms, like the emma platform. The emma platform supports various containerization technologies and extends container management and orchestration capabilities to all environments — any cloud, edge, or on-premise.



Vendor-Neutral CI/CD Pipeline

In keeping with the principle of vendor agnosticism, it’s important to embrace open-source and vendor-neutral tools throughout the CI/CD pipeline. The idea is that organizations must plan for interoperability right from the early stages of development. The CI/CD pipeline itself has several stages, each serving a specific purpose in the software development and delivery process and utilizing its own set of tools and technologies. A vendor-neutral CI/CD pipeline entails that none of the tools and technologies utilized across the pipeline are tied to any specific cloud or technology vendor. This ensures that the CI/CD process can work with diverse technologies, cloud platforms, and infrastructure configurations.

Below are the key stages of a CI/CD pipeline and how each can be made vendor-independent to ensure consistency and interoperability when different phases of the pipeline are executed on different clouds or when different components of the same application are deployed across different clouds:

Source Code Repository

The source code must be stored on a cloud-agnostic version control system like Git that integrates seamlessly with any cloud platform or tools needed for the subsequent stages.

Continuous Integration

Tools like Jenkins, GitLab, or CircleCI are needed to build and test the application continuously. They connect seamlessly with Git and work across multiple cloud providers.

Deployment

Packaging applications into containers using Docker, an open-source containerization platform, maintains portability. It goes hand-in-hand with Kubernetes, which is vendor-neutral itself and automates the deployment and coordination of Docker containers in complex, large-scale containerized applications distributed across multi-cloud clusters.

Enabling Interoperability in Multi-cloud with the emma Platform

Enabling an interoperable multi-cloud requires careful planning from the get-go. However, ensuring interoperability by design may not be feasible for many enterprises in different phases of their multi-cloud journey. emma is a cloud-agnostic, no-code multi-cloud management platform that takes the complexity out of multi-cloud interoperability. It enables different environments, legacy on-premise, private, public, and multi-cloud, to integrate and work together seamlessly through abstractions, single-pane-of-glass visibility, and intercloud connectivity. These attributes go beyond technical interoperability to foster semantic and policy interoperability as well, creating a cohesive and interoperable multi-cloud.

Here’s how the emma platform facilitates and enables organizations to embrace the seemingly complex principles of multi-cloud interoperability:

Microservices Architecture

The emma platform facilitates seamless communication between microservices across clouds by abstracting the complexities of different cloud interfaces. Its no-code approach allows admins to deploy virtual machines and containers across multiple clouds in just a few clicks.
The platform’s load-balancing capabilities that span the entire multi-cloud ecosystem, allowing efficient traffic distribution and optimal resource utilization across all clouds.

Cloud-specific APIs

The emma platform ensures consistency in deployment mechanisms when dealing with cloud-specific APIs and configurations. It abstracts the complexities and variations of individual cloud APIs and provides a unified and standardized deployment framework. The platform incorporates a powerful API gateway that serves as a centralized hub, enabling smooth communication across diverse systems and cloud providers.

Automation

The emma platform's Kubernetes capabilities allow automated container orchestration across multiple clouds. In addition, users can create policies and rules to automate cloud tasks and workflows, such as infrastructure resource provisioning, cross-cloud application scalability, compliance checks, and backups and disaster recovery.

Cloud Agnosticism

The emma platform abstracts cloud-specific interfaces to enable users to deploy workloads on any cloud platform in a cloud agnostic manner. The platform itself is cloud agnostic, which means it integrates seamlessly with all public, private, and on-premise cloud deployments. Organizations can manage and integrate their applications and infrastructure regardless of where they are hosted, all through a single platform.

Vendor Agnostic CI/CD Pipelines

The emma platform supports automated testing, deployment, and integration mechanisms through integrations with various CI/CD tools. Organizations can integrate their CI/CD tools, such as Jenkins and ArgoCD, to to ensure consistent deployment pipelines, automation workflows, and version control in a multi-cloud environment.