LTIMindtree

Container-Native Engineering: Future Proof Your Cloud Journey

LTIMindtree

Cloud technologies revolutionize how businesses share, consume, and manage data. Tools and processes created by these technologies make industries, as well as society as a whole, more efficient. Between 2009 and 2020, enterprise spending on cloud and data centers increased exponentially, directly reflecting the increased demand for cloud services infrastructure.

One notable area of growth within the cloud-native space is the increasing popularity of containerization. Today, containerized applications represent 30% of production applications and are expected to make up 75% of production applications by 2022. This aligns with the general trend towards cloud-based technologies. Are these figures alone enough to inspire business leaders to act? Probably not right away.

LTIMindtree

A Framework for Understanding Containerized Engineering

While current trends may help spark an interest in cloud-native technologies and engineering for others, they don't give curious organizations a clear idea of where to begin. For that reason, business leaders generally agree that these technologies are the future yet drag their feet when it comes to timely adoption.

The best way for leaders to navigate a fast-paced and rapidly evolving environment is to have a deep understanding of these technologies, as well as what they can offer their business.

LTIMindtree

Containerized engineering has four crucial, yet under-addressed, benefits:

1 Flexible Modernization

Containers offer flexible modernization options, allowing users broad control over the cloud technology journey, particularly regarding the rate and degree of technology adoption.

2 Targeted Scalability

When individual parts of the application are packaged as containers, those microservices can be scaled independently instead of scaling the entire application.

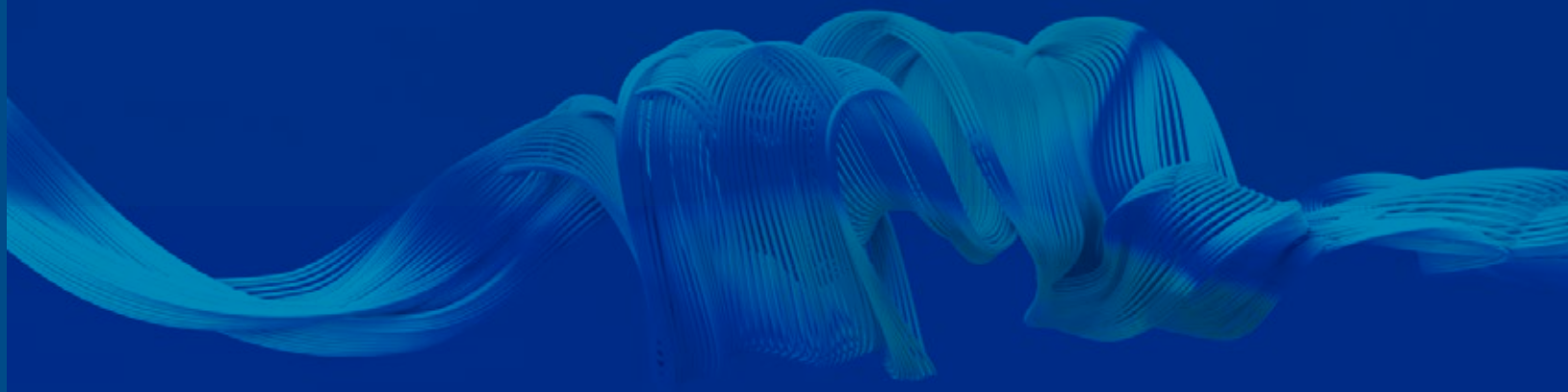
3 Improved Reliability & Application Density

By allowing developers to define an ideal environment for applications, containers ensure they run consistently regardless of the infrastructure used at deployment. Additionally, they do not host a full-fledged operating system, making them far more reliable and lightweight than Virtual Machines.

4 Simplified Operations

With containers, the particulars of applications are abstracted using container images, resulting in simplified operations. A growing suite of tools is making their use a standard.

This article will explore these benefits in-depth, along with some common challenges, offering a more holistic understanding of the context and utility of containers and containerized engineering.



LTIMindtree

1

Flexible Modernization

A container is a unit of application code that comes complete with everything necessary to run that code. This creates endless possibilities for their use, allowing applications to be more agile, portable, and rapidly scalable, regardless of when they were created or what business requirements arise.

There are four primary options when considering containerization for existing workloads

I

Monoliths on Containers

The first is monoliths on containers, a quick, efficient, and low-cost option for modernizing small to medium-sized applications. The result is a highly portable solution that carries no scaling requirements for individual parts of the application. While this does provide some benefits of containerization, those benefits are limited. However, it is still suitable for tech stacks and tightly coupled for critical legacy applications.

II

VMs on Containers

By containerizing Virtual Machiness (VMs), businesses can leverage container orchestration platform features like local access on cluster networks and service mesh. This eliminates the need to manage additional VMs and technology stacks. Containers are the fastest option for deploying VMs using a lift-and-shift option. It is also best for combining virtualized workloads and container workloads.

VM images are supported on KubeVirt. Containerization of VMs allows for heavy interactions with other containerized applications in the cluster. It is also a great option when there is a need to access the same secrets, configurations, and volumes as other containers. This creates a low-effort and cost-effective option but is not necessarily quick. It also is constrained in terms of portability and Continuous Integration (CI)/Continuous Deployment (CD) considerations. Moreover, it is less tolerant of post-migration updates and patching.

LTIMindtree

III

Cloud-Native Containers

This presents the possibility of a cloud-native technology stack and can be refactored to scale parts of the application independently. A modular approach to application development results in applications that run more reliably, regardless of the deployment environment.

IV

Serverless Containers

Serverless containers are an excellent option for microservices or applications with occasional or sporadic usage. This solution is predominantly used to handle asynchronous communication requests and can be dramatically more cost-effective.

Each option carries its own pros and cons, but all involve some combination of more targeted scalability, improved application density, heightened reliability, and simplified operations. And the possibilities for containerized technologies are always improving, especially when working with a cloud provider that offers excellent support for containerization ex. AWS.

LTIMindtree

2

Targeted Scalability

Until recently, organizations relied exclusively on the monolith as their infrastructure model for application development. That’s because in the early days, the development of web applications centered around a carefully constructed codebase. They had simpler architectures and weren't as in-demand, so maintaining them wasn't as challenging.

Now that applications have become more complex and more users access them, it becomes necessary to rethink how they are built, deployed, scaled, and monitored. That’s because with traditional applications, scaling a part of an application requires scaling the entire application, which forces organizations to modify functions even when it is not beneficial. This problem is entirely separate from the codebase and has more to do with runtimes.

The simple truth is that monoliths are not well-suited for situations where business needs are constantly in flux. Containerized engineering improves traditional application development by offering a modular alternative that provides more targeted scalability.

By packaging and deploying applications on containers, organizations have more control over how they scale, what they scale, and for how long. Based on workload, capabilities can expand elastically, which would not be automatic or instantaneous with a traditional workload. The result is more cost-control and efficiency.

3

Improved Reliability & Application Density

The benefits of containers go beyond maintaining a competitive edge. In fact, statistics show that containerized deployments are responsible for a 36% gain in IT efficiency, on average, and that number is even higher using managed Platform-as-a-Service (PAAS) providers.

Container First Engineering

Container-first engineering is a mindset that prioritizes containers as the default option when developing an application. Along with their flexible modernization and improved scalability, containers also lend quite a bit of portability, greater efficiency, and less overhead.

Because containers are a complete unit of application code, they have very low dependence on the environments in which they run. Gone are the days of developing an application in one environment, only to watch it fail when transported into another!

Unlike virtual machines, containers share a single kernel, whereas VMs share a server. This enables containers to run multiple applications on the same hardware but in total isolation, achieving the highest possible application density on available hardware.

Serverless and Serverless First

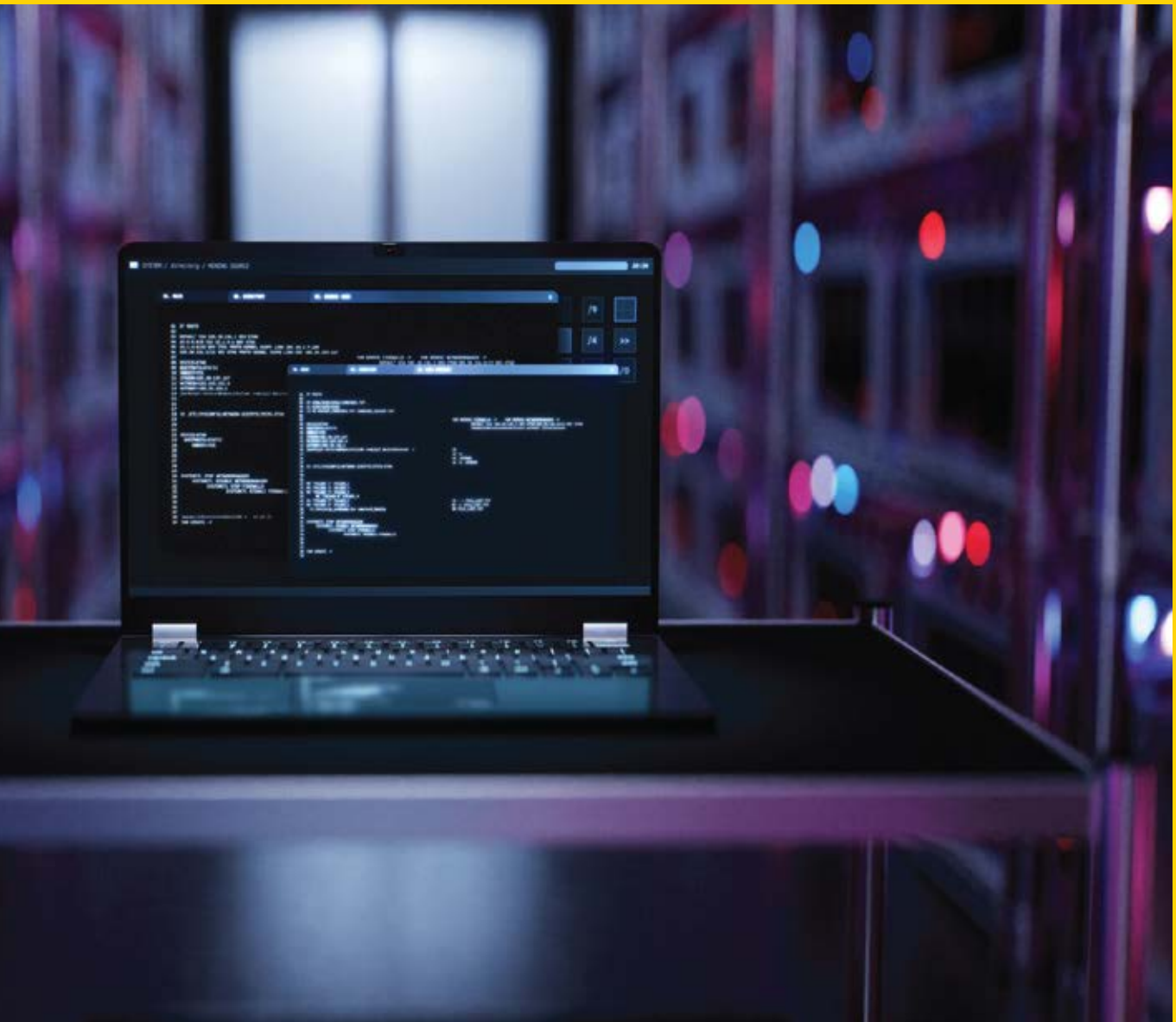
Serverless currently represents the fastest-growing cloud service model with a growth rate of around 75%. Containerization allows organizations to take a serverless approach to scale for business needs and plays a role in the serverless-first mindset. It recognizes the opportunity costs in a tradeoff between the added expense of using a managed service versus the benefits of faster iterations.

Serverless for on-prem and hybrid cloud offers additional benefits for containerized workloads, allowing for an even greater application density over the same set of resources. This means that when no users access a certain application in a serverless container, that application gets scaled down to zero and thus consumes no resources. This occurs while remaining a regular application instead of Function-as-a-Service (FaaS) like AWS Lambda is written specifically for that purpose.

Knative Serverless with Kubernetes or OpenShift is suitable for small monolithic applications, microservices, or even functions. This involves strong internal couplings between objects and services and is the best option when accessing the same environmental variables, secrets, and volumes as other containers. They can access cluster-hosted services, including Redis cache, Kafka, ELK, and DB.

They can also function as a service and are best suited for atomic functions that are headless and reusable. Since there is some request-level latency, serverless containers should only be used when they will not produce a business impact. There are also upcoming technologies relating to serverless containers. You can have serverless containers powered by OpenShift Serverless on any public cloud or Kubernetes, plus Knative on any public cloud, such as AWS or on a private cloud.

Serverless containers are not as modularized as AWS Lambda, which functions as a service but is still equipped to handle occasionally used microservices. There are also upcoming technologies relating to serverless containers.



LTIMindtree

4

Simplified Operations

So much of the innovation around containerized engineering relates to the use of container images, which constitutes the basic, non-alterable code of a container's design. Abstracting the particulars of an application opens the door to simplified operations. And with so many benefits and increasing levels of adoption, a growing market of tools and services is emerging around containerized engineering.

The current suite of frameworks and technologies, microframeworks, and runtimes available to maximize the benefits of containerization is already a long list, so only a few items can be highlighted.

LTIMindtree

Micronaut and Quarkus

Micronaut and Quarkus are two newer software frameworks that have become invaluable in creating containerized modular and lightweight cloud-based applications and services. They tackle some common problems with REST services in Java, such as startup time and memory consumption. These runtimes cut down on the integration testing time, making the process much easier for developers. Micronaut is specifically designed for building modular and lightweight cloud-based applications and services. It analyzes microdata and builds its dependency injection when the application is compiled, instead of at the run time.

One of the key benefits of using Micronaut is that startup time and memory consumption do not factor into the size of the application's codebase. It also incorporates microservice patterns, which helps developers reduce their need for the host of different tools and frameworks upon which they previously relied. It is also incredibly flexible and adaptive, designed for compatibility with serverless functions for cloud providers like AWS Lambda.

Quarkus is designed explicitly for building Kubernetes-native Java applications, optimizing them for container deployment. It serves as a reliable tool for cutting through compatibility issues, making Java a key development framework for serverless environments. Because users no longer have to worry about disruptions in the application development process, Quarkus significantly boosts the productivity and efficiency of Java developers and reduces operational costs.

It also doesn't occupy much memory, can quickly and easily be deployed for reactive event-driven applications, and therefore rapidly modernizes existing applications or facilitates the creation and deployment of new microservices and serverless workloads.

Helidon

Helidon is a lightweight and fast open-source framework, which offers a collection of Java libraries designed specifically for writing microservices. This framework makes it even easier for Java developers to be competitive in the creation of microservices-based applications.

Currently, Helidon has two versions: Helidon SE and Helidon MP. Both have incredibly compact footprints but carry slight differences from one another.

Service Mesh

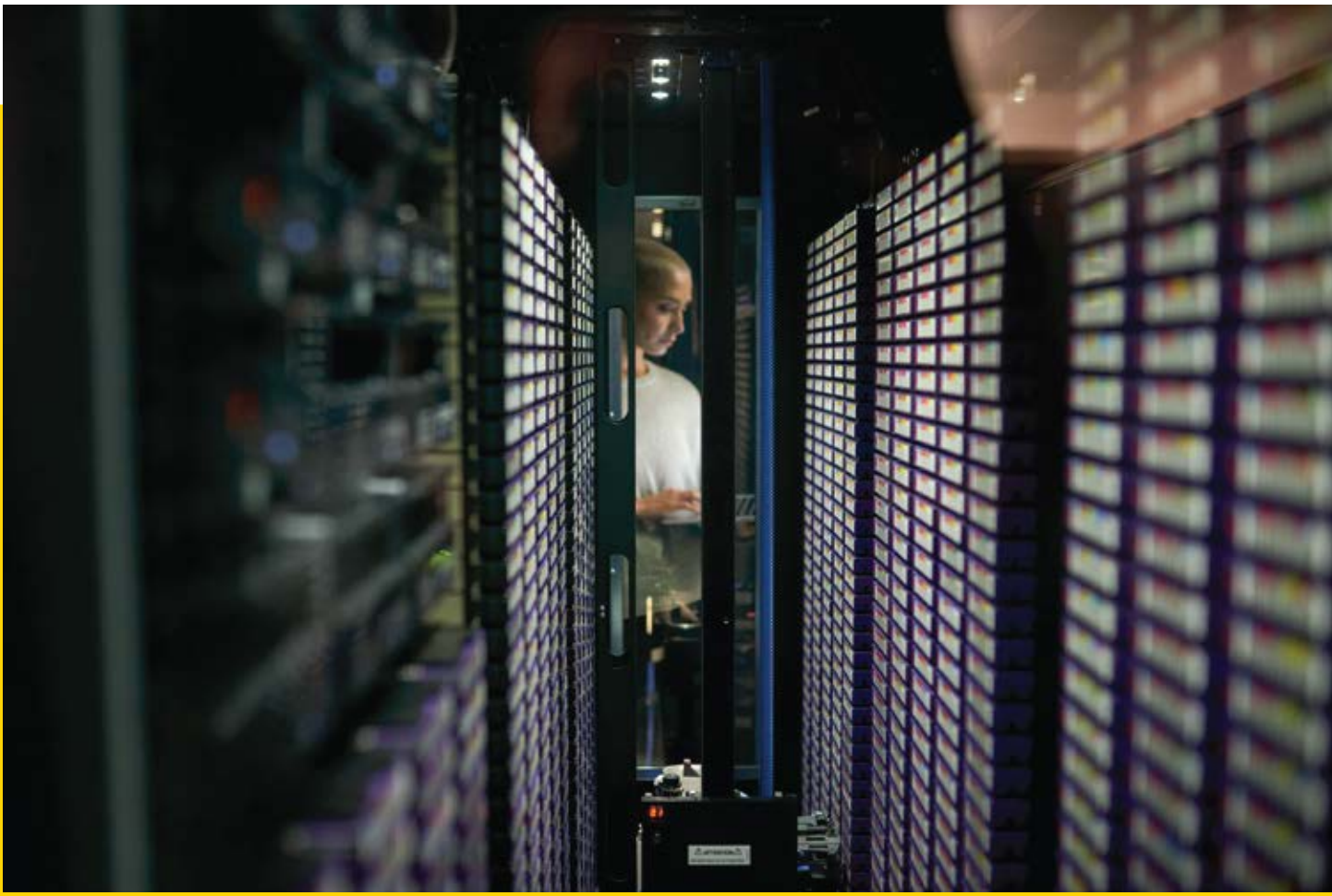
Since microservices are modularized, dedicated attention must be paid to communication within these containerized environments to ensure that they run efficiently. Therefore, there needs to be a service mesh, which provides users with a custom-configured infrastructure layer to oversee the applications and monitor how the various microservices and parts interact with one another.

Businesses often accomplish this through a sidecar, a proxy instance paired to each service instance. By taking responsibility for communications, monitoring, and security, a service mesh allows developers to effectively transfer those responsibilities to operations teams. This then frees development teams to focus their attention on the codebase.

Operators for Stateful applications

Container orchestration platforms like Kubernetes and Enterprise Kubernetes platforms like OpenShift, Rancher, Tanzu can manage and scale stateless applications (such as web apps, mobile backends, and API services) without requiring any additional knowledge about how these applications operate. The built-in features of Kubernetes are designed to handle these tasks easily.

However, stateful applications, like databases and monitoring systems, require additional, domain-specific knowledge that Kubernetes doesn't have. This advanced knowledge is needed to scale, upgrade, and reconfigure stateful applications. By encoding this specific domain knowledge into Kubernetes extensions, Kubernetes operators make it possible to manage and automate an application's life cycle.



LTIMindtree

Common, Yet Avoidable Challenges in Containerized Engineering

Containers offer a modular alternative to the monolithic architecture models traditionally used to create and run applications. They provide users with more flexibility and help businesses operate more efficiently, speeding up time-to-market while cutting costs.

Aside from offering portability, scalability, consumption-based infrastructure, intent-based provisioning, and unified data experiences, it also promotes a simple, straightforward, and cost-effective solution for businesses. The benefits are even greater when your cloud provider has industry-leading containerization support, which is currently offered by AWS.

However, containerized engineering poses some challenges that can wreak havoc if not addressed early on:

Standardization

Standardizing is a challenge when it comes to containerization. Containers are named according to labels; whatever command an administrator provides will always be on top of the labels given to a container image and must be standardized.

Labels, names, and other commonly shared objects must be standard when using containers. Containers are built on top of images, but developers cannot use any container image as a base. Additionally, you should make sure that the image is whitelisted, vulnerability checked, etc., and that those factors are standardized.

Standardization is also necessary when you think about versioning. One prime benefit of containers is that they allow you to roll back what you've already deployed. This is difficult with a traditional setup. In a traditional setup, if you have a newer version on the production system, you have to keep a backup and restore the backup. In a container-based world, you can instead roll back to the minus-one or minus-two versions. By correctly naming all your images that are deployed, you can avoid the mess that might otherwise occur.

A prime example of the need for standardization is when it comes to database user IDs and passwords. If over a period of time, your developers are creating connection names and passwords (like 10 copies talking to the same database) and the password changes, you may forget a few of the passwords causing your entire system to crash in production. Instead, it would be more efficient to keep those database user IDs and passwords stored in a commonplace. This means each developer creates their own connection name and password for the same database. These considerations must be addressed as methodically as possible for a well-functioning system.

Eliminate Technological Redundancy

The way you structure the platform itself can cause specific issues. Imagine a situation in which one team uses a certain technology for one function and another team uses another technology for another function. There will soon be a handful of logging frameworks, a handful of caching systems, and so on.

That's why it's vital to create a landing zone architecture so that you can whitelist certain technologies for each of those services. When an application gets deployed, it is understood that it will use one of these whitelisted services, instead of arbitrarily selecting another option.

Minimize Container Image Sizes

Containers are all about flexibility and portability so that you can handle fluctuations in workloads. If you have large images, of 3GB or 4GB, it will take a while to kickstart those images and move them from one mission in the cluster to another. That becomes difficult to manage and control by a central cloud council for various reasons. In this sense, smaller images tend to be more flexible.

Follow Security Best Practices

Security concerns are frequently raised about containers, but many of these concerns are outdated. In fact, there are many options for boosting container security. For example, signing containers with certificates enables tamper-proofing for the highest possible level of protection.

LTIMindtree

LTIMindtree: A Leader in Container Native Engineering

Conversations on containerized engineering tend to be vague, brief, and one-sided. The result is a skewed and oversimplified view of containers, which causes confusion and reluctance. This needs to change.

Cloud technologies should be discussed and communicated in a way that empowers business leaders to take a step forward, not retreat. It is important to be open-minded when it comes to new technologies, acknowledging both their benefits and limitations.

Container services have grown rapidly and gained popularity for their simplicity, agility, and portability. Enterprises use these features to grow their business at speed and at scale. By combining containers with the public cloud, they have been able to provide both orchestration platforms and serverless solutions.

With all visions of the future pointing to the cloud, LTIMindtree has the expertise and commitment to customer service that keeps you confidently grounded in the present. Cloud Container Services are amplified with a fit-for-purpose framework encompassing the end-to-end lifecycle of the migration process – assess, migrate, deploy, operate, and optimize. Cloud migration without optimization yields no real benefits. We deliver results with speed and scale with our comprehensive tools, pre-built artifacts and templates, and cloud-native architectures and technology partnerships.

The results? Our clients and their customers can reduce overall IT costs, ensure faster time-to-market, improve compliance and security posture, enable more rapid migration, and significantly improve ROI.

LTIMindtree

More about LTIMindtree

LTIMindtree is a top global technology consulting and digital solutions company based in India but rapidly expanding its presence and operations in the United States. We are an industry leader with IPs to better serve our customers.

About the Author

Sakthivel Sabanayagam

Cloud Native Engineering & Consulting Head,
LTIMindtree

Follow him on

Sakthi is a Principal Cloud Strategist and a hands on Solution Architect at LTIMindtree, with in-depth knowledge and execution of Cloud Advisory services and Cloud Native engineering across technologies and business domains.

LTIMindtree is a global technology consulting and digital solutions company that enables enterprises across industries to reimagine business models, accelerate innovation, and maximize growth by harnessing digital technologies. As a digital transformation partner to more than 700+ clients, LTIMindtree brings extensive domain and technology expertise to help drive superior competitive differentiation, customer experiences, and business outcomes in a converging world. Powered by nearly 90,000 talented and entrepreneurial professionals across more than 30 countries, LTIMindtree — a Larsen & Toubro Group company — combines the industry-acclaimed strengths of erstwhile Larsen and Toubro Infotech and Mindtree in solving the most complex business challenges and delivering transformation at scale. For more information, please visit www.ltimindtree.com.