Architecting the Future

Cloud Native Applications

Architecting the Future

Cloud Native Applications

Content

Cloud native has become one of the biggest trends in the software industry. It has already changed the way we think about developing, deploying, and operating software products.

Organizations across every industry want to remain competitive, and their leaders realize that it is necessary to maintain at least two technology dispositions. Of course, there is a basic need to "keep the lights on" while supporting product and service innovations. The other disposition must be forward-looking—to grapple with fast-changing technology and which technologies to embrace. This includes CI/CD, containerization, and microservices, to name a few. The pressing need is to secure the right amount of infrastructure flexibility and performance elasticity to manage unpredictable usage volume and geographic dispersion. At many companies, there is a strong sense of urgency: adapt quickly, or become irrelevant.

As architects and engineers have sought to exploit cloud computing, the essential drivers of cloud native architecture have become the following:

Resilience

It's risky to assume that your deployment environments and the networks are permanent. They will eventually change—and substantially so. Your critical applications may not receive any warning to accommodate a smooth shutdown. Therefore, it's vital that you design for failure and assume that some services on which you depend could disappear at any time.

Discovery

Services that support your applications must be readily locatable and accessible by other services. Services are often built to scale dynamically and change locations. Therefore, software must have the ability to readily discover the locations of other components and services—and communicate with them.

Scalability

Though you may manage your applications in the cloud, these apps won't achieve the efficiencies of a cloud native architecture if they are unable to scale horizontally.

Robust, reliable cloud native computing is driven by sound cloud native architecture.

To better prepare for the future, it's important to get a solid understanding of this rising technology trend. In this e-book, we examine cloud native architecture, look back at the rise of cloud native app development, and explore the future of cloud native on the entire software ecosystem.

While cloud native architecture may never be understood by typical consumers, it is steadily making headway in innovative companies and continues to have more influence. Indeed, there is already a well-established industry group, The cloud native Computing Foundation. Industries as diverse as online retail, investment banks, banking and various consumer industries are embracing cloud native architecture as a major foundational underpinning of business strategy. It's important to understand this new technology movement.

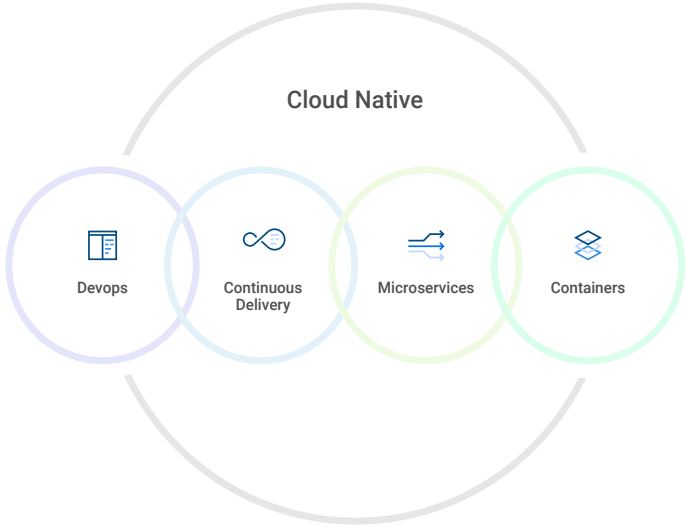
What is a Cloud Native Architecture?

Let's begin with a general definition:

> **A cloud native architecture is a model for building and running applications on a platform that is distinct from infrastructure dependencies—exploits the major aspects of the cloud computing model.**

Adopting cloud native architecture is much more than merely moving some workload over to a public cloud vendor. It is an entirely new and different approach to building infrastructure, developing applications, and structuring your teams.

In more detail, a software application that exhibits a cloud native architecture is built extensively with microservices. Typically, a cloud native app runs on a containerized and active-orchestration platform that leverages various cloud computing assets such as Kubernetes.



Cloud native architecture consists of four major aspects: Microservices, DevOps containerization, and continuous delivery.

Microservices

In cloud computing, this is a sub-architecture in which applications utilize sizeable collections of small services—each of which performs a very specific function. Typically, a microservice runs as an independent process that implements a specific business or technical capability. A microservice communicates by means of APIs, or a messaging protocol. Good design principles call for microservices that are deployable, upgradable, scalable, and restartable. Each microservice should be independent of other services, and also free of dependence upon the calling application. This highly resilient architecture enables frequent updates to live applications with no impact to application end users.

DevOps

DevOps is a tightly woven collaboration among software developers and IT operations. For cloud native DevOps teams, the difference from conventional teams is that these apps are built to exploit the adaptability and resiliency of cloud computing technologies. The goal is to continuously and progressively deliver high-quality software that meets customer requirements and expectations. On the best teams, DevOps supports a culture in which software is frequently built, tested, and released with a high degree of consistency.

Continuous Integration and Delivery

Enabled by Agile product development practices, continuous delivery (CD) is a practice that frequently and automatically moves small batches of updated software through a development pipeline to final delivery into a production environment. CD is most effective with the support of continuous integration (CI), which is a development practice requiring developers to integrate new code into a common repository several times daily. During an automatic build process, each code check-in is verified. This enables teams to respond much sooner to any problems that are found. The major goal of CI/CD is to bring software releases to the point of uneventful and error-free reliability. Using platforms such a Jenkins or CircleCI, many organiza-

tions that have built a mature CD pipeline can deliver frequently with low levels of risk—and get quick feedback from end users.

Containerization

Essentially, a container is a software package containing everything necessary to run an application. For example, many simple containers consist of an application server, a virtual machine (VM), and the application itself. A container can run in a virtualized environment, while isolating the application residing within from its environment. The main benefits of this architecture include environmental independence and high portability. It's easy to move the same container among several environments: development, test, or production. If your application has a horizontally scalable design, you can start multiple instances of a container to accommodate additional user demand. You can setup automatic termination of these instances with a decrease in demand.

Currently, Docker is the most widely used container implementation. Containers offer greater efficiency and performance in comparison with VMs. Using OS-level virtualization, a single operating system instance can dynamically support multiple isolated containers—each having its own distinct, writable file system and resource profile. Virtualization technology is combinable with container orchestration to achieve a high degree of flexibility and responsiveness. Only a small amount of overhead is necessary to create and terminate containers dynamically. Combining this with ability to densely pack containers into virtual machines means that containers are a very attractive compute vehicle for deploying individual microservices.

The Journey: How We Got to Cloud Native Applications

Over the past decade, modern software development has been abandoning use of on-premise physical servers in favor of cloud computing infrastructure. When cloud computing initially arose, many IT departments tried to merely transfer their systems to the cloud without making any changes to application architecture. Multi-tier applications were simply migrated from local physical servers to a virtualized cloud environment. Over time, cloud system engineers have made many innovative improvements in cloud platforms and infrastructures. These initiatives have spawned several engineering trends that continue to this day. One such trend is the move from monolithic systems toward microservice architectures, which has enabled the cloud native movement we see exploding today.

There are reciprocal benefits available to those that invest in the cloud native movement. The pursuit of more efficiency in the use of cloud resources continues to affect cloud native architectures, and the pursuit of better architecture further improves efficiency. Overall, this efficiency comes in the form of better customer service and faster product and service development.

In addition to the cloud native characteristics that we mention above, these are some of the dominant software development trends that are presently shaping cloud native architecture:

REST APIs

REST-based APIs provide scalable and pragmatic communication. This approach relies heavily on long-standing Internet protocols, infrastructure, and well-defined, well-established standards.

State Isolation

Stateless components are much easier to horizontally scale—either up or down. Though stateful components aren't entirely avoidable, they should be reduced to a minimum.

Loose coupling

With loose coupling, service composition occurs largely through events or data. More specifically, event coupling relies on messaging solutions such as the AMQP standard. Data coupling often relies on scalable yet eventual consistent storage solutions—which are often NoSQL databases.

Elastic Platforms

Elasticity is the extent to which a system accommodates workload changes with automatic provisioning in real-time. Elastic platforms such as Kubernetes, Swarm, or Mesos are often seen as critical unifying middleware for broadly elastic infrastructures. These platforms greatly expand and maximize resource-sharing. They also increase utilization for the underlying compute, storage, and network resources for standardized deployment units. Learn more about cloud native elasticity below.

Cloud Modeling Languages

Modern cloud computing supports a high degree of automation in service provisioning, which enables cloud-service customers to dynamically acquire services for deploying cloud applications. Specialized cloud modeling languages seek to provide flexible approaches for managing the increasing diversity of cloud computing features. These new languages aim to support different scenarios—such as migration of existing applications to the cloud, new cloud app development, and optimization.

Serverless

This is a cloud-app architecture that depends heavily upon external third-party services. These are integrated using small event-based trigger functions in an architecture known as Function-as-a-Service (FaaS). FaaS uses time-sharing to exploit and maximize resource-sharing within these elastic platforms. Read more on this elsewhere in this paper.

Microservices: The Heart of Cloud Native Design

Many business applications have been built in the last 15 years using various types of service-oriented architecture (SOA), and this has eventually led to the rise of cloud native architecture. Service-oriented computing is a computing paradigm for managing the complexity of distributed systems and integrating disparate software applications. In essence, one service provides functionality to other services—primarily through a messaging scheme. Importantly, an SOA service design decouples its interface(s) from its core implementation, and specialized workflow languages orchestrate the actions among all of the services.

Historically, many technologists have been optimistic that these service-oriented applications are redeployable into cloud environments with only a modest level of reconfiguration. Indeed, many would describe such apps as cloud-ready or cloud-friendly. However, cloud-system engineers commonly point to a big problem. Though conventional SOA applications consist of distributed services—their deployment does not consist of such service distribution! From a deployment perspective, such "distributed" applications are effectively monolithic applications.

The trouble is that any update or service release for a legacy distributed application must be redeployed in its entirety. This leads to the common practice in which monolithic applications are simply repackaged into a single, large virtual machine image. But the consequence, in many cases, is significant downtime for end users.

These legacy monoliths also have limited scalability—effectively unable to handle substantial variation in computing workload. Such architectures may be acceptable for some operations, such as batch processes running in the wee hours of the morning. But, it is unworkable for any modest-load daytime operation.

The need to move beyond such constraints and limitations has been driving systems architects toward the adoption of microservice architecture, which is a major characteristic of cloud native computing.

The Evolution Toward Greater Elasticity

Cloud computing supports cloud native architecture. Though it has taken many years, senior system engineers have come to better understand and exploit the elastic potential of modern computing technology to adequately support cloud native applications. Today, a number of innovative system designs specifically focus on elastic cloud infrastructures. This directly supports horizontal scalability, which is essential for cloud native architecture. Using containers, microservices, and serverless architectures, these systems significantly increase the utilization of underlying computing infrastructures. This architecture is now commonly known as cloud native.

To count as an advancement over conventional architectures, cloud infrastructures and platforms must be highly elastic. Elasticity is the extent to which a system accommodates changes in workload by automatically provisioning and terminating resources—in real-time. Lacking essential elasticity, cloud computing is typically unfeasible. Indeed, it is both financially and operationally impracticable.

The figure below depicts a definitive trend that has been developing over the last decade. Machine virtualization made it possible to consolidate large arrays of bare-metal machines and increase utilization of physical boxes—and it forms the technology backbone of cloud computing. However, while virtual machines have a smaller resource footprint, the image size of VMs are still quite large.

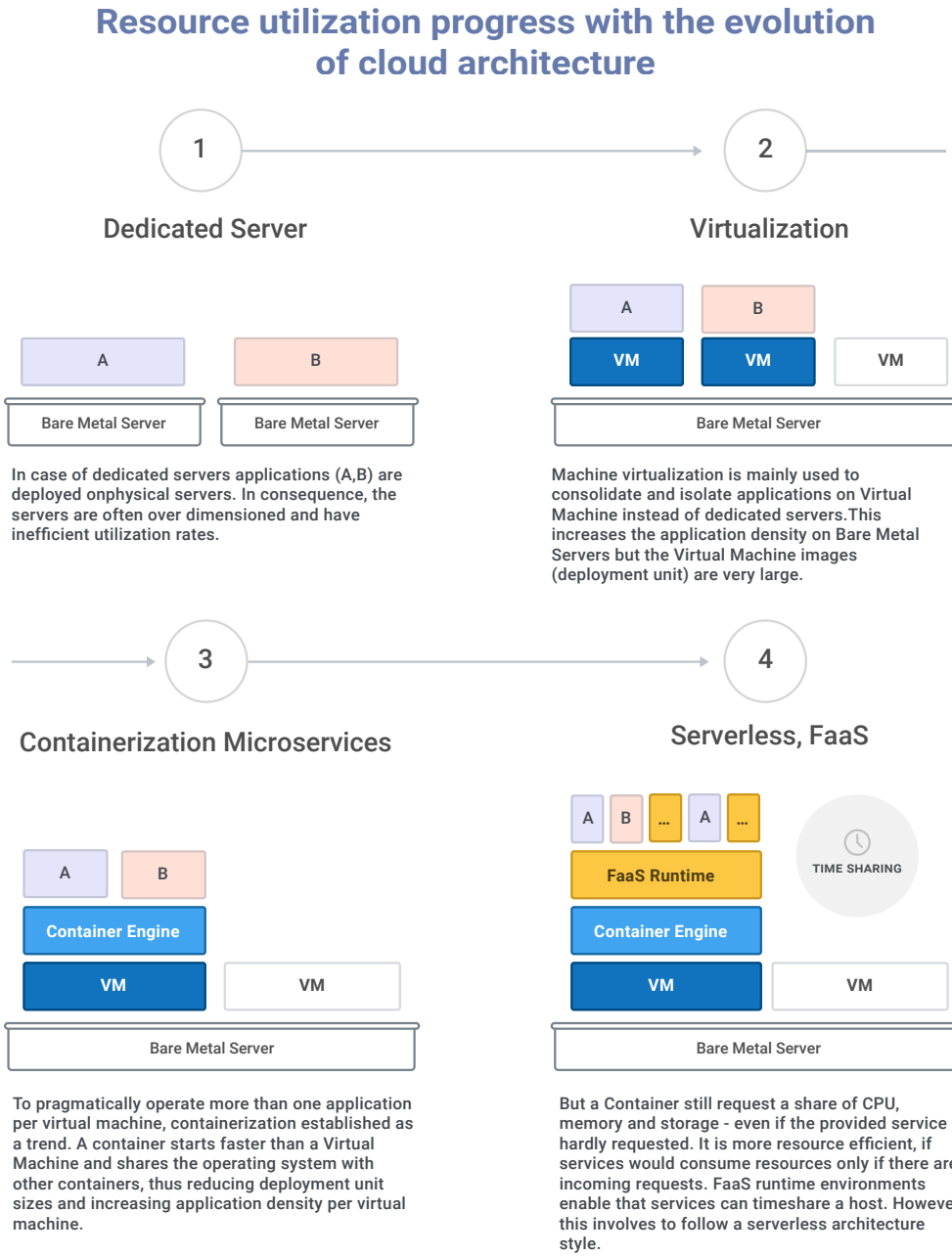


Figure 2. The cloud architectural evolution from a resource utilization point of view, adapted from “A Brief History of Cloud Application Architectures,” by N. Kratzke, 2018, Applied Sciences

Container technology came along to improve and simplify standardized deployments, while also increasing VM utilization. There is a trade-off, however. Although containers can quickly scale, they are always on—and always consuming considerable resources. To mitigate this, FaaS technology arose and made it possible to time-share containers on specialized container platforms. It's now feasible to employ a FaaS-only architecture to execute any deployment unit that presents a request for processing. This time-sharing capability is quite feasible on the same hardware, so FaaS can support a scale-to-zero capability. This greatly improves resource efficiency, which is fairly easy to justify financially. Recently, the technology stack has come to a point at which it can manage a complex array of resources in the cloud—and run significantly more workload on the same physical infrastructure.

Serverless Architectures

Microservice architectures offer modern solutions that can scale computing resources at a scale that is not achievable with monolithic architectures. However, any microservice architecture faces other challenges, such as deploying all microservices, operating them in a cloud computing environment, and scaling them as necessary.

This gave rise to serverless architectures and FaaS platforms in the cloud computing ecosystem. AWS Lambda is perhaps the most prominent, but there is also Azure Functions, Google Cloud Functions, OpenWhisk, and Spring Cloud Functions—among many others. All commercial platforms appear to follow the same basic principle: providing very small services (often consisting of only one stateless function each) that are billed on a per request or per-API call runtime-consumption model.

Serverless design supports event-driven applications, in which lightweight processes are triggered in response to an event. It is more intricate than microservice architecture, and facilitates the creation of a multitude of functions. These small functions are sometimes

called nanoservices—easily deployable and automatically scalable. Done properly, there is plenty of potential to reduce operations and infrastructure costs.

Cloud Native Architectures Combine Microservice and Serverless Designs

Microservices are deployable in various ways. They are often found in combination with a serverless architecture, residing in containers, or built with using PaaS. The benefits of microservices are applicable to local app development. But, the big advantages of microservices are seen most clearly within the context of a cloud environment—exploiting containers or in a serverless architecture.

The difference between a microservice and a serverless function is that a microservice is larger and has more capability than a function. Typically, a function is a relatively small bit of code that performs a single action in direct response to an event. A microservice may be equivalent to a serverless function in many cases, or it may consist of many functions.

Serverless microservices reside in a serverless infrastructure and only operate when a call is made by an application. A serverless computing provider provides access to its microservices or functions. Subscribers can write and deploy code that interacts with these services—without any concern about the supporting infrastructure. Most providers offer automatic scaling to support demand fluctuations. An organization that has access to such services is charged according to compute time. Typically, this results in lower costs since there is no longer a need to reserve a fixed amount of capacity that is often underutilized (and sometimes quite inadequate).

Drivers of the Cloud Native Movement

To see the forces that are behind the cloud native movement, let's compare cloud native applications with conventional enterprise apps.

Cloud Native Applications	Conventional Enterprise Applications
High predictability. Cloud native applications are compatible with a framework that maximizes resilience with uniform and predictable behaviors. Highly automatable cloud infrastructures attract better, more scalable application designs.	**Inconsistency and inflexibility.** Conventional applications simply aren't capable of achieving most of the benefits of running in a cloud native platform. These apps require more effort to build, deploy in infrequent batches, and only scale incrementally.
Optimal capacity. A cloud native platform automates infrastructure configuration and provisioning. It dynamically allocates resources in the deployment—according to the real-time demands of hosted applications. Cloud native architectures scale readily and exhibit better resource utilization.	**Excessive, wasteful over-capacity.** Conventional IT architects build dedicated, hard-wired infrastructure that delays app deployment. Often, the solution is over-sized to accommodate worst-case capacity estimates—but with little capability for scaling to meet future increase in demand.
Collaborative buildout. Cloud native architecture and methodology facilitate better DevOps, resulting in a closer collaboration across the team. It improves delivery speed and production quality.	**Silos.** Conventional IT deploys an over-the-wall release taken from developers—directly into production. Organizational priorities remain higher than customer value and service. The result is internal conflict, faulty delivery, and low team morale.

CI/CD. Development teams release an individual software update the moment it's ready. Such organizations release more frequently and get user feedback much more quickly. This is continuous integration and delivery, which works best with other methodologies such as test-driven development.	**Waterfall development.** IT teams release software occasionally, weeks or months apart. Some independent, release-ready components sit waiting for many weeks. Features that customers need now are delayed. The business misses opportunities to be more competitive and increase revenue.
Automatic, systemic scalability. Cloud infrastructure automation at scale greatly minimizes downtime that results from human error. Process automations consistently apply the same rules across any deployment scope or size. Cloud native goes well beyond the ad-hoc semi-automation that is layered atop conventional VM-oriented orchestration.	**Limited, manual scaling.** Manual infrastructure requires that human operators manually configure and manage the server, storage, and network configurations. At larger scales, operators can't manage the complexity, so they improperly diagnose issues and fail to implement correctly.
Abstraction from the OS. Cloud native architecture gives developers a platform that enables them to abstract away from underlying infrastructure dependencies. Rather than configuring, maintaining, and patching operating systems, teams can place their full attention on building the application.	**OS dependency.** Conventional application architecture restricts developers to building many hard dependencies among the application, the hosting OS, storage, hardware, and backend services. These constraints limit the ability to migrate and scale the application to new infrastructure.

High independence. Microservices architecture enables teams to build applications that consist of small, loosely coupled, independent services. The buildout of these services maps to smaller development teams, which have the independence to create frequent updates that scale and recover without affecting other services.	**Heavy dependence.** A monolithic architecture bundles many disparate functions and services into a single deployment package. This establishes many unnecessary interdependencies among services, resulting in a lack of flexibility for development—and deployment.
Rapid recovery. Both the container runtime and orchestrator provide dynamic virtualization for virtual machines, and this is highly beneficial for hosting microservices. Orchestration can dynamically manage container placement across a VM cluster to provide elastic scaling and recovery-restart if there is any failure.	**Sluggish recovery.** Simple VM infrastructure is quite a slow and inefficient foundation for microservice applications. This is because individual VMs take too long to start or terminate. Also, VMs require too much overhead—even when no application code has been deployed yet.

Benefits of a Cloud Native Architecture

Cloud native applications are purpose-built for the cloud model. These applications—built and deployed in a rapid cadence by small, dedicated feature teams to a platform that offers easy scale-out and hardware decoupling—provide organizations with greater agility, resilience, and portability across cloud environments.

Cloud native architecture offers many benefits.

Competitive advantage

Cloud native architecture reflects a shift in the view of how computing architecture should affect business goals. The change in thinking has evolved from a focus on merely reducing costs to building the engines that will grow the business. In this ever-expanding age of software, organizations that build enduring success will be those that quickly deliver applications in response to current and anticipated customer needs.

Increase in flexibility

While cloud providers offer impressive services at reasonable prices, most enterprises aren't yet willing to choose only one infrastructure. Flexibility is important because it's risky to depend on a single vendor. Maintaining flexibility also preserves the ability to quickly adopt new technologies and enables developers to choose which tools are the best for their needs.

Enable teams to focus on resilience

Services are likely to suffer when legacy infrastructure fails. In a cloud native world, teams can focus tightly on building for resilience to failure or sluggish performance. Cloud native architecture enables system designs that remain online despite any anomalies that may occur anywhere in the environment.

Align operations with business strategy

By automating IT and delivery operations, companies can become leaner as its technology teams align more closely with business priorities. The risk of failure due to human error is mitigated when all staff can focus on automation that replaces inefficient administrative tasks. Automated upgrading and maintenance releases can happen at all layers of the tech stack, which virtually eliminates downtime and manual intervention from ops experts.

Though cloud native architecture has many benefits, there are also many challenges that practitioners face. Let's explore that next.

Challenges of Moving Toward Cloud Native Computing

A huge mistake that some practitioners make is lifting and shifting old, on-premise apps directly to the cloud. The attempt to migrate a monolithic, legacy application onto a cloud infrastructure will not result in any benefits from cloud native features. It is far better, instead, to decompose or refactor the legacy app into a cloud native architecture or consider development of new cloud native applications into a new cloud native environment.

Moving into the future, it will become increasingly important to dispense with old development paradigms and methodologies. The waterfall model is effectively outmoded, and classic Agile may be inadequate. To enjoy the benefits of cloud computing, it is essential to adopt new, cloud native application development approaches. These include minimum viable product (MVP) development, rapid iteration, multivariate testing, and close integration across organizational boundaries by implementing a modern DevOps and CI/CD methodology.

There are many aspects to cloud native design, development and deployment. This includes virtualization, containerization, orchestration, microservices architecture, infrastructure services, automation, and observability. Embracing each of these will require assimilating new approaches, and this in turn requires setting aside old habits. To be successful, it's important to avoid being overzealous. Proceed at a steady pace.

The Impact on Practitioners

Let's briefly examine a number of key insights about cloud native practitioners. All these points significantly influence how practitioners build cloud native architectures that are fully and expressly designed exclusively for cloud computing.

Cloud native practitioners:

- Prefer to transfer platforms, not applications.
- Want to have more than one platform option.
- Prefer declarative and auto-adjusting approaches—rather than workflow-based approaches—to deployment and orchestration.
- Are compelled to efficiently use cloud resources as many more systems are being migrated to cloud infrastructures. This continues to increase operation costs.
- Value pragmatic solutions much more highly than full-feature coverage of cloud platforms and infrastructures.

Considerations for Practitioners

Distributed computing is not easy. Indeed, it's complex—if only for the reason that microservices must communicate. The challenge facing container technology users is that a container running a microservice has a network IP address. This address requires network management. To build cloud native applications upon a microservices architecture, other containers need the IP address of a container to communicate with it.

To support networking, each microservice container must provide network security, a firewall, messaging queuing, load balancing, and other basic network services. Managing all of this is a major challenge in the next evolution of cloud native computing. Also, network elements in a cloud-first architecture are often fragile. Since robust

networking is a significant unsolved problem of cloud computing, it's essential that cloud native architectures are built to be resilient.

Cloud native architecture is heavily dependent on microservices. And yet, despite all the excellent benefits, the microservices approach doesn't solve all problems. While mitigating many of the issues that inhibit monolith applications, microservices present some entirely new challenges. What it offers in agility and speed of development comes at a cost of increase in operational complexity with the multitude of services to manage. There are many more moving parts than a comparable monolithic application.

Employing a microservices architecture is likely to increase operating overhead. An overall deployment burden may require significantly more resources simply because there are a large number of successive deployments. The result is likely to be the need for more effort in creating the infrastructure. Most services need clustering—both for resilience and failover. A typical system has dozens or more separate components. As the team continues to add new features, the system becomes increasingly complex. In contrast to a comparable monolithic application system, a cloud native microservices application may employ many types of services—each running many instances processes. It's vital that you address this additional overhead with automation and add DevOps staff that possess skills in infrastructure automation.

How Cloud Native Will Affect the Entire Ecosystem

Cloud native is not yet a dominant software development paradigm, but it won't be long. Look closely. In a recent Cloud Foundry survey, over 75 percent of about 600 IT decision makers are evaluating or using Platforms-as-a-Service (PaaS). 72 percent are evaluating or using

containers; 46 percent are evaluating or using serverless computing. More than one-third are employing some combination of all these technologies. It is in those companies using all three technologies that cloud native computing is gaining momentum.

We now live in a multi-platform world. So, it is unsurprising that—in gaining comfort with disparate tools and platforms—technical decision makers continue to search for a suite of technologies that work well together. They seek technologies that integrate well with existing platforms—to address current needs. But they also want the flexibility to accommodate future needs.

Increasingly, it seems best to meet such challenges by combining tools or refactoring into cloud native applications. Over half of the Cloud Foundry respondents report that their companies do both: building new, cloud native applications and refactoring existing applications. That's an increase of nine percentage points from the same survey results only a few months earlier (in late 2017).

New Cloud Native Application Development is on the Rise

How exactly are companies using cloud native technologies? It is true that many have been using it to refactor legacy applications. But that is changing. Today, many cloud native practitioners indicate that they are primarily building new cloud native applications. Fewer are using the tech to refactor older applications. An increasing number of companies are developing new cloud native applications while many companies are deploying on deploy PaaS more broadly.

Idea to App in the Shortest Amount of Time

The biggest advantage of a cloud native architecture? You can go from idea to app in the shortest amount time. No other app development paradigm is more efficient.

Cloud native architecture involves much more than merely getting your apps to run in a cloud-computing environment. It goes much deeper, to a major change in how you plan the supporting infrastructure and design your apps around microservices. And, if you're going to make foundational changes to infrastructure, you'll need a new toolset that is purpose-built for cloud native operations.

Team dynamics must also change to be smaller, more agile, multi-functional, and enabled to make decisions that directly affect the services they manage. All of these efforts combine to create serious momentum—that leads to much faster releases and more scalable applications. Yes, the move to cloud native architecture starts with a significant amount of effort, but it is clearly the way forward in an increasingly competitive cloud-computing world.

Konghq.com

Kong Inc.
contact@konghq.com

251 Post St, 2nd Floor
San Francisco, CA
94108 USA