Chapter 1. What Is Cloud Native?

Cloud native is more than a tool set. It is a complete architecture, a philosophical approach for building applications that take full advantage of cloud computing. It is also complex, both conceptually and in practice.

In this chapter we will take a look at the major components of a cloud native system—the five principles—and how they work together. Understanding these core concepts helps newcomers see how the true value of cloud native lies in harnessing the ecosystem of extraordinarily sophisticated services now available to all enterprises, not just the tech giants.

Cloud Native Is Not "The Cloud"

Though the terms are often confused, cloud computing and cloud native are two entirely separate entities.

Cloud computing—often referred to simply as "the Cloud"—is the on-demand delivery of infrastructure (hardware/servers), storage, databases, and all kinds of application services via the internet. Frequently these are delivered by a cloud services platform like Amazon Web Services, Google Cloud, or Microsoft Azure, with metered pricing so you pay only for the resources you actually consume.

Cloud native is an architecture for assembling all of the above cloud-based components in a way that is optimized for the cloud environment. *It's not about the servers, but the services*. So cloud native is also an organizational *destination*: the current goal for enterprises looking to modernize their infrastructure and process, and even organizational culture, carefully choosing the cloud technologies that best fit their specific case. (At least, the goal for now—eventually, even quite soon, cloud native will be replaced by another paradigm that once again completely changes our way of doing things).

There! That was easy.

Perhaps too easy, actually. After all, there are innumerable paths for reaching your cloud native migration destination. Identifying, provisioning, and then deploying the just-right combination of services to best take advantage of this new, rapidly evolving world among the clouds can take very different forms, depending on the needs of a particular organization. It's easy to get lost.

For enterprises ready to undertake their own cloud migration, staying on track means focusing on the architecture: understanding and prioritizing design before jumping into full-on implementation and deployment.

Over the course of five years spent guiding enterprises onto the cloud, Container Solutions engineers have learned a thing or two (or three) about helping each company find its own optimal route. We are most definitely not prescribing any "top-down" one-size-fits-all solution. We have by now, however, through observation and experience, collected enough data to identify some landmarks necessary for mapping that route.

Developing a cloud native pattern language is the next step in drawing a useful, and reusable, roadmap. A shared language for identifying common contexts and discussing tools, techniques, and methods is essential for developers to be able to discuss, learn, and apply the best practices in cloud native—even as they continue emerging.

But first let's take a quick, basic look at how cloud native works.

A Cloud Native Primer

Let's begin with the closest thing to an official definition for "cloud native":

Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization.

This comes from the Cloud Native Computing Foundation (CNCF), the entity that oversees and coordinates the emergence of open source technologies that support cloud native software development. CNCF

emphasizes open source technologies, but there are also important cloud native tools offered by commercial providers.

Essentially, cloud native is the name of a particular approach to designing, building, and running computer applications. The architecture rests upon Infrastructure-as-a-Service, combined with new operational tools and services like continuous integration, container engines, and orchestrators. The objective, usually, is to improve speed. Companies of all sizes now see strategic advantage in being able to move quickly and get to market fast—putting a new idea into production within days or even hours, instead of months.

In fact, most enterprises migrating to cloud native these days cite velocity as their primary motive.

How Do I Know Cloud Native When I See It?

The fundamentals of cloud native are most often described as container packaging, dynamic management, and a modular distributed architecture.

We, however, believe cloud native is actually about adopting five architectural principles (which is hard) plus two cultural ones (which is even harder):

- **Containerization:** Encapsulating applications together with their dependencies/operating environment, all in a single package. This makes them easy to test, move, and deploy.
- **Dynamic management:** Using cloud-based servers that can be flexibly provisioned on demand; if on a public cloud, which is typical, companies pay only for resources when they are actually used.
- **Microservices:** Designing applications as a collection of small, decoupled component services. Each microservice can be deployed, upgraded, scaled, and restarted independent of other services in the application, and with no impact on the end user. Microservices increase velocity by allowing teams to develop in parallel, working on their components simultaneously yet independently, thanks to the elimination of dependencies and the coordination efforts that come with them.

- **Automation:** Replacing manual tasks, like maintenance and updating, with scripts or code so they happen seamlessly and reliably.
- **Orchestration:** Tying it all together by automating the deployment, scaling, and management of containerized applications. Specifically, using Kubernetes (or another orchestration tool) to control and automate tasks such as the availability, provisioning, and deployment of containers, load balancing of containers across infrastructure, and scaling up/down by adding/removing containers as needed.

The two cultural principles are:

- **Delegation:** Offering individuals the tools, training, and discretion they need to safely make changes, then deploy and monitor them as autonomously as possible (i.e., without needing to hand off to other teams or seek permission through a slow management approval process).
- **Dynamic strategy:** Communicating strategy to teams, but allow them to modify that strategy in response to their results. That is the ultimate purpose of the fast, experimental deployment that cloud native provides: there's no point running experiments if you don't make use of what you learn.

Ultimately, **cloud native** *is about how we create and deliver, not where*. So, when you see an application built and deployed in rapid iterations by a squad of independent, compact feature development teams—and those teams are collaborating via an integrated platform that decouples infrastructure while providing automated monitoring and testing—that is when you know you are looking at the cloud native approach in action.

We are not considering the cloud native approach because it's the current hot tech (though this is undeniably true). Our motivation is pragmatic: cloud native works well with fast, modern software delivery methods like continuous delivery to provide faster time to value; it scales horizontally and effortlessly; and it can be very efficient to operate. cloud native architecture also allows us to create complex systems by dividing them into smaller components built by independent teams. This differs from traditional monolithic application complexity,

which is limited by the ability of the developers and architects to fully understand it even as it chains them together to deliver in unison.

Most importantly, cloud native can help reduce risk in a new way: going fast but small. So how and where do we start building?

It's All About Services

The heart of cloud native is cloud-based services. This is the platform upon which we build, launch, and operate our distributed, containerized, and automated modular application empire. There are different types of services available from public cloud providers:

- **Infrastructure-as-a-Service:** This is the obvious one, and it includes off-premises hardware, data storage, and networking. Hiring infrastructure rather than owning it allows you to maximize the creativity of each team instead of limiting it to the capabilities of a central architecture team.
- **Platform-as-a-Service:** This can be used to manage and maintain all that virtualized infrastructure, greatly reducing the load on your Ops (or Platform) team.
- **Software-as-a-Service:** Allows you to pick and choose component applications, everything from traditional business software (think MS Office 365 or Adobe Creative Cloud) to virtual infrastructure management tools, all delivered via—and operated over—the web. The provider ensures security, availability, and performance.
- **Container-as-a-Service:** Lets you hand over container engines, orchestration, and all underlying compute resources for delivery to users as a service from your cloud provider.
- ***-as-a-Service:** If you can dream it, if your business requires it, there is probably a service for it. If it doesn't exist right now, just wait a month or two. Backend-as-a-Service, Functions-as-a-Service—these once pie-in-the-sky services are now crossing the chasm into full enterprise introduction even as we write this book.

All cloud services arrive pre-built and ready to wire up with any other services, so you can get right to work more or less instantly. However, in order to use them effectively, you must use the right architecture.

Understanding the Principles

Cloud native is a lot to wrap your head around: it's an architecture, a tech stack, an approach to software development and delivery, and a complete paradigm shift all rolled into one! To confuse things even more, cloud native implementations vary widely between enterprises thanks to the sheer number of tools available as well as the complexity they offer. It's dangerous to go alone, so take this: the five principles of cloud native architecture. They are the defining elements that appear (or at least very much ought to appear) in every cloud native implementation, no matter what your company's size or sector. Recognizing these five principles, and understanding how they interrelate and support each other, will give you a working knowledge of cloud native fundamentals.

To reiterate, the five principles consist of:

- Containerization
- Dynamic management
- Microservices
- Automation
- Orchestration

Containerization: Once you've defined your service-based architecture, it only makes sense (for just about everybody, everywhere) to containerize things. Containers are lightweight, standalone executable software packages that include everything required to run an application: code, runtime, system tools, libraries, and settings. They are a sort of "standard unit" of software that packages up the code with all of its dependencies so it can run anywhere, in any computing environment. You can link containers together, set security policies, limit resource usage, and more.

Think of them as scalable and isolated virtual machines in which you run your applications. (We know this statement has historically launched a thousand flame wars, so let's at least agree that containers are simply much faster, OK?). Containers isolate an application and its dependencies, even its own operating system, into a self-contained unit that can run on any platform, anywhere. This means you can host and

deploy duplicate containers worldwide (thanks to your Infrastructure-as-a-Service!) so your operations are flexible, reliable, and fast.

Dynamic management: This is where your new system absolutely shines. In short, dynamic management means making optimum use of the benefits conferred by your new cloud platform. Compute, network, and storage resources are provisioned on-demand, using standardized APIs, without upfront costs—and in real-time response to real business needs.

Dynamic management takes away the costs typically involved in capacity planning and provisioning of hardware resources. Instead, a team of engineers can start deploying value to production in a matter of hours. Resources can also be de-allocated just as quickly, closely mirroring changes in customer demand.

Operating compute, network, and storage resources is traditionally a difficult task that requires specialized skills. Obtaining these skills is often time-consuming and expensive. Even more important, though, is speed: humans are never going to be able to respond as quickly to cycle up and down as demand surges and sinks. Letting your chosen cloud platform run things dynamically means resource life cycles get managed automatically and according to unwaveringly high availability, reliability, and security standards.

Microservices: Microservices (microservice architecture) is an approach to application development in which a large application is built as a suite of modular components or services. Each service runs a unique process and often manages its own database. A service can generate alerts, log data, support UIs and authentication, and perform various other tasks. Microservices communicate via APIs and enable each service to be isolated, rebuilt, redeployed, and managed independently.

They also enable development teams to take a more decentralized (non-hierarchical) and cross-functional approach to building software. By using microservices to break up a monolithic entity into smaller distinct pieces, each team can own one piece of the process and deliver it independently. Ideally, some of these parts can even be acquired as an on-demand *-as-a-Service from the cloud.

Think about the companies setting the bar for everyone else in terms of performance, availability and user experience: Netflix, Amazon, the instant messaging platform WhatsApp, the customer-relationship management application Salesforce, even Google's core search application. Each of these systems require everything from login functionality, user profiles, recommendation engines, personalization, relational databases, object databases, content delivery networks, and numerous other components all served up cohesively to the user. By breaking all this functionality into modular pieces and delivering each service separately and independently, you increase agility. Each microservice can be written in the most appropriate language for its particular purpose, managed by its own dedicated team, and scaled up or down independently as needed. And, unlike in a tightly coupled monolithic application, the blast radius from any change is contained within that microservice's footprint.

Automation: Manual tasks are replaced with automated steps in scripts or code. Examples are automated test frameworks, configuration management, continuous integration, and continuous deployment tools. Automation improves the reliability of the system by limiting human errors in repetitive tasks and operationally intensive procedures. In turn, this frees up people and resources to focus on the core business instead of endless maintenance tasks.

Simply put, if you are trying to go cloud native but don't have automation then you are rapidly going to get yourself in a mess. Enterprises come to the cloud to deploy more quickly and frequently. If you haven't fully automated your deployment processes, then suddenly your Ops staff are spending all that time they save by no longer managing those on-premises servers to instead manually deploy your new, expedited production cycle. More frequent deployments also mean more opportunities to screw up every week; putting things into production faster and scaling them faster also means generating bugs faster. Automated deployment takes the grunt work out of constant implementation, while automated testing finds problems before they become crises.

Orchestration: Once microservices architecture is in place and containerized, it is time to orchestrate the pieces. A true enterprise-level application will span multiple containers, which must be deployed

across multiple server hosts that form a comprehensive container infrastructure, including security, networking, storage, and other services. An orchestration engine deploys the containers, at scale, for required workloads while scheduling them across a cluster while scaling and maintaining them—all the while integrating everything with the container infrastructure.

Orchestration encourages the use of common patterns when planning the architecture of services and applications, which both improves reliability and reduces engineering efforts. Developers are freed from solving lower-level abstractions and get to focus on the application's overall architecture.

This is where Kubernetes comes in, and it is one of the very last things to be done in a cloud native migration. If you implement an orchestrator first, you are fighting a battle on simultaneous fronts. Using an orchestrator effectively is a highly complex endeavor; getting that right often depends on the flexibility, speed, and ability to iterate you have put in place first. Other cloud native principles—cloud infrastructure/dynamic management and automation—must be in place first. Quite often, when experts are called in to work with a company whose cloud migration has gone wrong, what we find is that they have put in an orchestrator before things were otherwise in place.

Use your platform to build your platform—before you start worrying about orchestrating all the pieces!

Fitting Everything Together

The five (technical) principles, constructed in the proper order, are all essential supports in a cloud native architecture. One, however, may be even more important than all the others: microservices.

Microservices occupy a central role among the five principles. In order to get microservices right, you must have a mature approach to all four of the other principles. At the same time, containers, dynamic management, automation, and orchestration are truly powerful only when combined with microservices architecture. Figure 1-1 shows how everything fits together.

Figure 1-1. The relationship diagram between the five principles of cloud native architecture

For example, one of the main advantages of containerization is that it enables heterogeneous applications to be packaged in a standardized way. This is not very compelling if your entire business logic is built inside a monolith. Similarly, you could apply dynamic management to such a classic homogeneous enterprise application on public infrastructure or a Platform-as-a-Service cloud provider. Doing so, though, means wasting the capability of scaling up and down in response to your business needs.

And, yes, while it is true that automation can still be applied to monolithic architecture, at least to a certain degree, the level of automation achievable with microservices is vastly higher given their self-contained, independent nature. Finally, modern orchestration platforms assume that applications will be composed of smaller, containerized services. A "lift and shift" cloud migration of a traditional application to run in containers on top of modern orchestrators is possible. However, it requires a great deal of adaptation and investment—all while failing to capture the significant benefits of microservices architecture.

There are cases where doing a lift and shift of an existing monolithic app to run on a cloud native platform is a reasonable choice. Companies with existing cloud native expertise can find advantages in moving an application to the cloud first, before re-architecting to optimize it to run there. This doesn't get you to full cloud native, but instead jump-starts your journey in two out of five areas as a starting point for further transformation. So, in the right circumstances, there can be some value in this approach.

Experience is essential, however: if you're going to use an experienced partner to guide your cloud native transformation, you may get it right. But the risk of doing it on your own is way too high for the value. We too often see companies with limited cloud knowledge trying to simply shift their existing monolith onto the cloud as an end goal rather than a starting point. This requires a lot of time and resource investment with only limited benefits—at which point many companies will get discouraged and delay, or even cancel, further transformation efforts.

The most compelling argument for placing microservices at the center of cloud native principles is the fact that they encapsulate business logic, the differentiating factor for any enterprise. Microservices are capable of representing the processes by which a business delivers value to its customers. Ultimately, this shortens the distance between strategy definition and execution—serving the need for speed that brings most enterprises to cloud native in the first place.

However they are deployed, these five principles now have techniques and tools behind them that are approaching full maturity and commoditization. The ease of use and robust tooling offered by current containers and orchestration platforms is truly impressive. Once upon a time (four years ago) the world's most advanced technologies were attainable at true production scale only by large companies who could maintain in-house IT teams dedicated to the development, care, and feeding of powerful but still immature innovations like containers and microservices. The competitive advantages these technologies confer are now publicly available, commoditized and available to anyone with an internet connection and a credit card.