ORACLE
Communications

A Cloud-Native Journey for Telecommunications

How to reap the promised benefits of cost savings, rapid deployment and customer empowerment

ORACLE COMMUNICATIONS WHITE PAPER / FEBRUARY 22, 2019

ORACLE

DISCLAIMER

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

AGILITY THROUGH CLOUD

Agility describes the ability to move at speed and change direction, all while maintaining balance and control. Without agility, a business faces disruption, or even worse, commoditization. Although telecom is not an industry known for its agility, and is often thought of as the opposite, it — like many industries — has been transforming through adopting more agile approaches in business strategy, striving to maximize resources and reduce inefficiencies while at the same time sparking innovation. Nothing has been more of a stimulus for agility than the cloud, especially for the webscale companies whose services pose the greatest threat to the telco industry.

Per the National Institute of Standards and Technologies, cloud computing is defined as "…a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[1] However, the definition falls short in revealing why cloud is critical to achieving true agility.

Cloud is significant in that it gives businesses the ability to rapidly develop, test and launch software applications that drive business growth at a rate and scale not seen before — and all while eliminating the need to buy and maintain equipment. Virtualization enables the abstraction of the IT stack so that one physical resource is used logically like many, creating more flexibility and scalability, and opening the door to new service models. The addition of automation further allows the focus on other issues such as core business logic, security, and analysis, as opposed to just provisioning and maintaining the resources. Finally, when the new service models are coupled with the cloud's transparency of resource utilization, new business models start to emerge. The "anything-as-a-service" paradigm of cloud drives a mostly static and transactional world into a more dynamic and subscription-based economy.

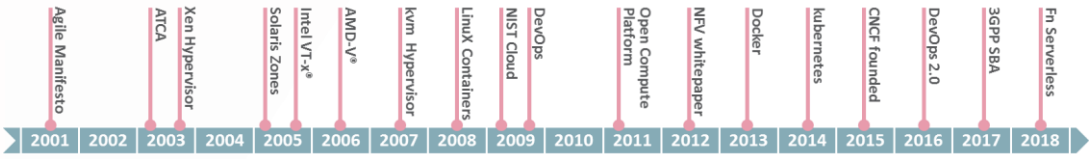

Figure 1: the modernization of infrastructure is continuous in nature.

Cloud was a major modernization undertaking for the IT industry, sending it on a journey that is still unfolding today. Meanwhile, the telecommunications industry was undergoing its own modernization effort, shifting to all-IP networks and connecting data centers filled with commercial off-the-shelf servers that run open source operating systems. Nevertheless, the thinking became that there was a need to adopt a more "authentic cloud" where the industry was more likely to reap the promised benefits of cost savings, rapid deployment and customer empowerment.

'CLOUDIFICATION' WITH NFV

Network Function Virtualization (NFV) soon became the predominant way the telecommunications industry sought to leverage cloud. The NFV goal is to offer Software-as-a-Service (SaaS), but the building blocks are Network Functions (NF), something not traditionally thought of as software in the same sense as enterprise applications in a SaaS context. Network equipment vendors seek to "cloudify" their products by porting the underlying software from dedicated bare metal into virtual machines (VM), allowing these Virtual Network Functions (VNFs) to take advantage of the basic cloud tenants.

[1] NIST SP 800-145

Most of the resulting VNFs still have dependencies between the lifecycle of the service and the underlying software's lifecycle. Some even have dependencies on underlying infrastructure resources. The core architecture for NFV has to incorporate multiple layers of hierarchical control through orchestration to manage the extensive and intertwined dependencies. Instead of dynamic elasticity, orchestrators manipulate fixed-size templates to manage capacity. The use of 1+1 redundancy schemes between VMs to ensure high availability sometimes makes the resource consumption worse off than the dedicated appliances they are moving away from. Configuration becomes so complex that the orchestration engines have to be augmented with custom VNF Managers from the vendors, further complicating interoperability. NFV is not achieving the promised benefits; rather, it adds extra layers of complexity.

THE PEOPLE AND PROCESS STORY

Technology, or how it is realized in the case of NFV, is not the only factor in achieving cloud agility. Before cloud there had been a widely held notion that development of software is a different activity than its operation. As organizations sought to become more agile with the development of software for cloud, the resistance to change in favor of stability from operations became more apparent. The relationship where the development team "throws code over the wall" to operations led to the classic battle of: "it's not my machines, it's your code," vs "It's not my code, it's your machines." Coined by a presentation at O'Reilly Velocity 2009[2], this resulted in the need for a new process approach for cloud – *DevOps*. It became evident that the relationship between development and operations needed to improve as both are indispensable for a successful cloud. There was now an apparent need to have more operational awareness in development while having a more developer mentality in operations. This was to be achieved by having software developers support what they built while having operation engineers introduce software engineering practices into the platform — to better understand and adopt the notion that everything, including process, is code.

An agile business must embrace change, as that is fundamental in bringing new services to market. Both the software developer who adds new features and the operations engineer who keeps the services stable are essential to enabling this business objective. However, the notion of "change"—often embraced by development but shunned by operations—can be associated with the root cause of most outages. The business must make a choice. The first option is to discourage change in the interest of stability. The second option is to change as often as needed by mitigating the risk of change through tools and culture. Needless to say, the latter has been the DevOps path to a successful cloud business.

Automation is one of the most effective tools for agility. To do automation successfully though, it needs to incorporate both development and operations. Build and deploy should be a one-step process, based on a shared, version-controlled repository so that everyone knows where to look. Likewise, shared metrics and data flows enable everyone to be talking about the same thing and oriented to the same goal. Finally, both should share the same communication channels to establish a platform for more automated closed loop processes and ensure there is no misunderstanding.

If there is only one cultural change an organization makes, it needs to be instilling respect between development and operations. Fundamental to gaining respect is a having trust in the job each team does, that the job is done with the best intent for the business and with full transparency. Once there is trust and the organization is aligned to the same goals, it can create the environment where harder issues such as failure can be addressed. Often organizations develop an unhealthy focus on preventing failure, sometimes at the cost of not being able to respond effectively when it does happen, which then can lead to appointing blame and eroding the hard earned organizational trust.

Time for DevOps 2.0

DevOps 1.0 has been mostly centered on harmonizing the interplay of development and operations with the goal of institutionalizing continuous delivery. In DevOps 2.0 we see the emergence of adaptive feature delivery and the broadening of scope to non-technical teams.

Business adopts cross-functional methods to ensure that software is iterated on a continuous cadence complementary to marketing and sales campaigns. Decoupling feature rollout from code deployment with feature flags gives marketing, design, and business teams control of targeted visibility and testing without consuming engineering resources or compromising the application's integrity.

Last but not least, DevOps 2.0 elevates security as a fundamental element from beginning to end with the mindset everyone is responsible for security. This is sometimes referred to as DevSecOps. By thinking of security as code, security specialist are given the tools to contribute value with less friction.

[2] "10+ Deploys per Day: Dev and Ops cooperation at Flickr" link to: video, slides

One way to deal with failure is to minimize its impact through smaller iterations, which typically bring about a decreased Mean Time to Resolution (MTTR). By shipping the main trunk with feature flags[3] instead of branching the code, several failure testing and containment techniques are enabled, such as dark launches and canary testing. These techniques release a change to only a subset of the customers to benchmark a code change from either a performance or customer perception perspective. The flags also allow for the administration of private betas and in dire circumstances can be used as circuit breakers.

Finally, DevOps should be treated like a journey — one that can always be improved. Learning from lean manufacturing, an organization should be continuously evaluating how to further automate and simplify its processes. The most attention should be given to things which are queued, usually because they are awaiting human interactions. Reviews, approvals, and testing, for example, are often the larger bottlenecks requiring both tooling and cultural changes to effectively improve.

BETTER SOFTWARE FOR A BETTER CLOUD

Taking NFV as an example, having an underlying cloud technology like virtualization did not mean that software was able to be developed to maximize its capabilities. Even with the drastic improvement DevOps made for deployments, the way software was developed still prevented reaching agility at scale in the cloud. In order to better understand the next shift in cloud, it is necessary to reflect back on Service Oriented Architecture (SOA), which also came into prominence around the same time cloud made its first appearance in the IT industry.

At its core, SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. In SOA, services are the mechanism by which needs and capabilities are brought together through interactions between logical representations of a repeatable business activities with specified outcomes.

The SOA philosophy has had a profound impact on how present-day developers approach the concept of as-a-service. Its architecture introduced many benefits, including:

– loose coupling of components with relationships that minimize dependencies
– abstraction whereby services hide their logic, encapsulated within their implementations
– discoverability using metadata through which services can be effectively discovered
– implementing sophisticated and complex operations using composability of services as building blocks

Typically SOA manifested itself as multiple business services running in Application Servers, connected through an enterprise bus enabling some level of heterogeneous interoperability. It was an architecture that fit the waterfall development of the time. The philosophy was right, but the methodology did not quite fit into the agile paradigm of cloud. Through adjustments from the community a manifesto appeared, serving as the foundation for a microservices architecture, known as the *Twelve-Factor App*[4].

RESTful State

The rise of RESTful interfaces in telecommunications with their statelessness has caused a bit of consternation; therefore, it helps to understand what this really means. Looking back at Roy Fielding's thesis[5] on the topic, he cites three important properties that are induced by statelessness:

- **Visibility** – every request to a service contains all context necessary to understand it. Thus looking at a single request is sufficient to visualize the interaction with the service.
- **Reliability** – failure of one request does not influence others because the request stands on its own.
- **Scalability** – the server does not have to remember the application state, which is the state from the perspective of the consumer in the interaction, enabling it to serve more requests and to have multiple instances of the server handling those requests.

These are all very desirable properties and fundamental to the success of web-scale services. This does not mean that the service cannot be acting on a stateful resource, which is persisted data within the services. Only that the interaction itself is not aware of stateful concepts like "next", but rather must contain a representation of the resource with controls that would lead the server to a "next" action.

[3] Feature Flags or Feature Toggles have multiple categories and are catalogued extensively by Martin Fowler and Pete Hodgson. You can read how Oracle Integration Cloud uses Feature Flags in this blog post

[4] https://12factor.net/

TWELVE-FACTOR APPLICATION	
1.Codebase	one codebase tracked in revision control; many deploys
2.Dependencies	explicitly declare and isolate dependencies
3.Configuration	which varies between deployments should be stored in the environment
4.Backing services	all treated as attached resources, attached and detached by the execution environment
5.Build, Release, Run	delivery pipeline should strictly consist of build, release, run
6.Processes	applications deployed as stateless processes with persisted data stored on a backing service
7.Port binding	self-contained services should make themselves available to other services by specified ports
8.Concurrency	is advocated by scaling individual processes
9.Disposability	fast startup and shutdown are advocated for a more robust and resilient system
10.Dev/Prod parity	all environments should be as similar as possible
11.Logs	applications should produce logs as event streams and leave the execution environment to aggregate
12.Admin Processes	should be kept in source control and packaged with the application

Table 1: the Twelve-Factor Application

As opposed to SOA, applications built with this methodology follow the "single responsibility pattern," having a more granular and small nature to their service components, thus leading the term microservices to be used for the architecture. Microservices expose their capabilities via an API layer as opposed to relying on heavy middleware like SOA. This allows for a minimal coordination between development teams of service components and minimizes the impact of changes to the codebase. Most importantly, twelve-factor apps take into account these characteristics and are able to scale up without significant changes to tooling, architecture, or development process, hence truly delivering the full promise of cloud.

[5] Architectural Styles and the Design of Network-based Software Architectures

CLOUD NATIVE WITH ORACLE COMMUNICATIONS

The microservice architecture and concept of a twelve-factor application gives developers a pattern to create software that is considered truly native to the cloud, while DevOps becomes paramount for managing the agile cloud within which they would run. Nevertheless, technology continues to progress. This is best illustrated by the move from hypervisor-based virtual machines to container virtualization directly in the operating system, which is lighter weight and better for microservices. However, questions like how to orchestrate large quantities of containers and how to manage containers on multiple clouds exacerbates the need for the industry to chart a path through the landscape of ever increasing technologies to help developers make sense of it all. This is where the Cloud Native Computing Foundation (CNCF) comes into play, serving as an open source software foundation dedicated to making cloud native computing universal and sustainable. Oracle is a platinum member of the CNCF in the realization that its vision of an open, cloud native and standards approach is well aligned.

With a rich heritage in telecommunications, Oracle Communications has always had a deep understanding of service reliability. Developing applications in this space has placed an emphasis on creating foundational platform services whereby applications could be built consistently to these expectations. There has also been a keen awareness of the operational aspect of these applications. This has become even more pressing as the applications move to SaaS-based cloud delivery. It is this space, where Oracle's leadership with cloud technologies like those of the CNCF, which has given Oracle Communications new perspectives on what core principles are really necessary:

Figure 2: eight principles for successful cloud native operations

System Immutability: Everything, software and configuration, is code. All changes are made through CI/CD where they are deployed as immutable artifacts. No manual configurations or customizations are allowed. This makes it easier to implement the principle of least privilege as there is no need to run scripts in the production environment. Also, any changes not coming though the delivery pipeline can be considered malicious.

Automate Everything: All aspects of build, test, verification, and deployment are automated. This includes activities such as backup, recovery, password/key rotation, etc. Fully automating the DevOps pipeline (including verification and testing) removes much of the potential for human error, allows changes to be applied to the environment with confidence, and provides for rapid repair.

Disposability: All services are transient and treated as short lived. Instead of focusing on never failing, services are designed to go up and down quickly without service interruption. Regular repaving (re-deployment) of the environment ensures failed or failing services are removed and new ones deployed.

Externalized Configuration: Configuration (including passwords, credentials, location of backing stores, etc.) is decoupled from the software image, and like software, can be treated as a build artifact in a controlled and versioned manner. Versioned configuration enables development and production parity as an artifact, and can eliminate costly operational errors.

Logs as Event Streams and Constant Telemetry: Everything needed to debug or diagnose any functional, operational, or security issue must be in logs, traces, or metrics data. These are treated as a stream of time-ordered events and stored in a centralized collector outside the system where better threat monitoring, forensics, and diagnostics can occur through event correlation or analyzing the aggregated and holistic view.

Delegated Governance: Some shared aspects of the environment are centrally managed like networking, identity management, or infrastructure, but in a true DevOps fashion, teams delivering a service are responsible for operating the service. This is allowed only with strict governance, enforced through checks in the Continuous Delivery pipelines, giving greater control over rate of change. Business agility is improved as applications have better visibility into the operations of their service with tighter feedback loops that ultimately improves quality.

Independent Lifecycle: Independently upgrading, scaling and deploying each Microservice is paramount for supporting other cloud native principles as well as minimizing the amount of change in the system at a given time. Furthermore, such decoupling makes other principles such as repaving easier as well as promoting easier isolation of issues

Oracle Communications has realized that only with an operational framework that embraces these eight principles can it achieve an agile cloud with an environment enabling its developers to innovate true cloud native solutions. Furthermore, Oracle Communications has found that this also provides the ideal context in which to implement a broader view of the CNCF landscape, giving developers a robust Platform-as-a-Service of backing services beyond just containers.

5G EMBRACES CLOUD

The idea that an application can be built from a collection of loosely coupled components acting as producers and consumers to each other, like in SOA, has been one of the most prevalent paradigms found in application development. Recently, practitioners have these components interact through what is known as RESTful interactions, which are self-contained messages sent with a stateless protocol using a uniform interface. This predominately hails from the “web services” world and has been a significant portion of recent SaaS growth.

As a mental model, this paradigm can be challenging for network-oriented telecommunication applications. Historically, network applications have been defined using tightly coupled components which interacted through long-lived, stateful protocols. However, with 5G, 3GPP has now embraced the patterns found with REST through defining its next generation core with a Service Based Architecture (SBA). In this architecture, control plane capabilities are exposed as discoverable services with RESTful APIs. This will enable a much faster pace of innovation as new capabilities can be introduced and existing ones can be easily consumed in new ways without the need for defining new point-to-point interfaces or making difficult changes to existing protocols stacks and state-machines.

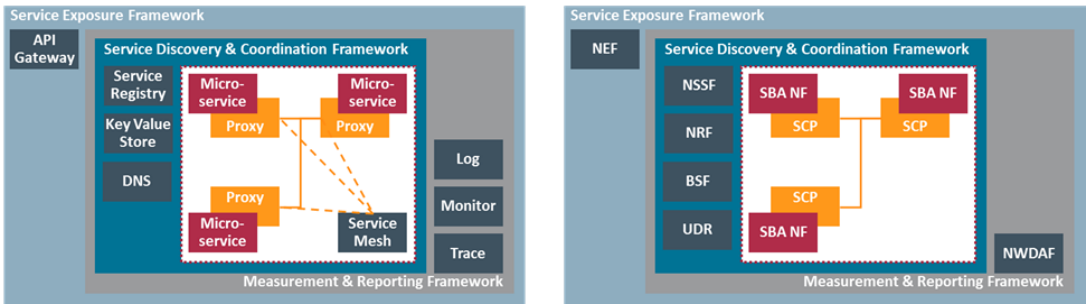

Figure 3 CNCF Service (left) versus SBA Services (right)

Service Mesh

There would be little value to a producer if a consumer was not able to reliably utilize its services. In a Cloud Native Environment, that ability to have reliable communications between services is where the Service Mesh comes into play. It is typically implemented as an array of lightweight network proxies deployed alongside application code, without the application needing to be aware. A service mesh can perform the following tasks:

- **Service discovery:** What are all of the upstream/backend service instances that are available?
- **Health checking:** Are the upstream service instances returned by service discovery healthy and ready to accept network traffic?
- **Routing:** Given a REST request from the local service instance, to which service cluster should the request be sent?
- **Load balancing:** To which upstream service instance should the request be sent? With what timeout? With what circuit breaking settings? Should it be retried?
- **Authentication and authorization:** For incoming requests, can the caller be cryptographically attested and is the caller allowed to invoke the requested endpoint?
- **Observability:** For each request, detailed statistics, logging, and distributed tracing data should be generated.

Intuitively, each component in this paradigm can be a seen as a microservice running in a container, but there are a lot of other moving parts behind the scenes that makes this come together. In the twelve-factor application, these are known as backing services and they make up a majority of the CNCF's landscape. A few notable ones include:

Key Value Store: A centralized service for maintaining information, naming of the information and providing distributed synchronization of the information. Often used for configuration and state.

Service Registry: Service discovery uses a registry to keep a list of services, their location, and their health. Services query the registry to discover the location of services so they then can connect directly.

Service Mesh/Proxies: A secure and reliable network between every microservice, which provides policy driven management of the traffic with full visibility.

API Gateway: Safely expose services to the outside via secure programming interfaces that can be managed and monitored.

Monitoring/Logging/Tracing: These are tools that collect data and events, correlate them, analyze them, search them, and otherwise manipulate them so that a business can have better operational awareness.

There is enormous benefit to having these readily available as services which application developers can utilize. Otherwise, each time they implement one of these there is effort to build and debug. Because these are not the core business of the application and can be difficult to implement, quality is often lacking, making the solution fragile and difficult to manage. Because of this value, it should be of no surprise that many of the services defined in the 5G SBA have an uncanny similarity to CNCF backing services (see figure 3), thus bringing SBA into the realm of agile cloud. Realizing this, Oracle Communications contributed to 3GPP the addition of service mesh and proxy concepts as an enhancement to SBA, becoming the basis of the Service Communications Proxy function[6].

FUTURE OF CLOUD FOR TELECOM

The mass media hype around Cloud Native has clearly captured the attention of the telecommunications industry, but does the industry really understand what it means? A pedantic view of the phrase leads to meaning nothing more than building something that is fit for the cloud. But which cloud? We have seen the cloud evolve from a definition of five basic characteristics to something that is now an entire landscape of technology with an accompanying operational philosophy. To pin that down today it would have to at least have the following as illustrated:

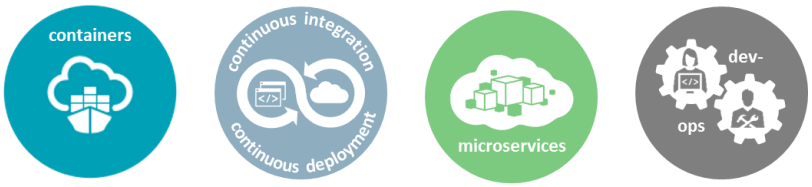

Figure 4: modern cloud native characteristics

[6] Original contribution S2-187627 to specification 23.742 - Study on Enhancements to the Service-Based Architecture. This has now progressed to normative Change Request 706 to specification 23.501 - System architecture for the 5G System for Release 16

None of these can really stand on their own, nor are any of these trivial in their own right. Containers is more than virtualization. It implies the rich ecosystem of services to orchestrate, schedule, network, deploy, and maintain the containers. Microservices are more than smallish applications. They require a different way of thinking about the service, how resources are used, how state is persisted and working in a distributed environment. DevOps is a cultural shift of unprecedented magnitude and not a token gesture between two organizations. Finally, Continuous Integration and Continuous Delivery are more than having some cookbook and collection of scripts. They are a pipeline that spans all aspects of creating and running a service, embracing automation at every step, and strict governance – literally turning the business into code. Most importantly, CI/CD is a singular codebase, always deploying main trunk, so there are not multiple versions of the cloud.

FINAL THOUGHTS ON DELIVERING SAAS AS CLOUD NATIVE

Bringing this all together is not trivial. It literally takes a platform in of itself to enable the delivery of SaaS as cloud native on an agile cloud. Oracle Communications has devised a Cloud Native Environment Platform-as-a-Service (PaaS) to capture all four of these aspects and more. On top of a robust bare-metal-as-a-service infrastructure there is a Kubernetes container management environment. Additionally there are key backing services so developers can concentrate on business logic instead of the tedium of logging or persistence framework. There is also a pipeline that facilitates developers designing, testing and building code all the way through to delivery, on a continuous basis. Furthermore, each of these are literally services in their own right, with their capabilities adhering to all of the principles discussed so far. In other words, the configuration for the pipeline is itself versioned code. The backing services are containerized microservices deployed using the same pipeline. In fact, over the span of a typical month, it is not uncommon for there to be several hundred changes propagated through this system across all of the layers from OS to platform to application, into a global cloud without service interruption. Finally, it is worth mentioning that there is an inherent difference in the philosophy in which some of the SaaS software of today is designed than that of traditional telecommunications NFs of the recent past. Some of the difficulty inherently comes from functional architecture of many network domains. Examples of such issues include:

- complex dependencies between configurations for multiple functions, which then have dependencies on component lifecycles
- application endpoints which cannot be decoupled from underlying resource lifecycles
- interfaces that don't lend themselves to service discovery, load balancing or otherwise enable the elimination of configuration dependencies

The root of these issues lies in the fundamental view that a service is something created by the configuration of distinct functions. A NF does not exhibit a service but rather tends to convey some set of capabilities. With some non-trivial amount of forethought, they can be configured and assembled to create services. In contrast, web services typically start from the service definition itself. This does not mean that the service cannot be decomposed into an architecture of constituent parts; rather, it means it does not start with parts that may not have been designed with a full understanding of the services in which they are to serve. The telecommunications industry now finds itself grappling with this very issue of a service-first mentality and that will drive the future of what agile cloud for telecommunications looks like.

Having a service-first view does not necessarily mean having solutions that are not interoperable. Having a well-defined API, not unlike what SBA has done, is a fundamental step towards this. Oracle Communications has approached its SaaS based services with this new way of thinking, designing around what it means to have a slice-as-a-service, built entirely from microservices that make sense to support the service while still maintaining the external interfaces per 3GPP standards. Most importantly, Oracle has learned from this that pursuit of an agile cloud, especially in this context, is an always evolving, continuous journey.

Delivery vs Deployment

While continuous deployment may not be right for every company, continuous delivery is an absolute requirement of DevOps. What really then differentiates the two? Fundamentally, delivery is having the pipeline contain all the mechanisms and practices to ensure the software is fit for production by placing every change into a production-like environment with rigorous automated testing. This gives the business the confidence that based on its criteria every change could then be "push-button" deployed to the production environment. If regulatory and business conditions allow, the goal of the business should be to have continuous delivery into production for every change.

ORACLE CORPORATION

Worldwide Headquarters
500 Oracle Parkway, Redwood Shores, CA 94065 USA

Worldwide Inquiries
TELE + 1.650.506.7000 + 1.800.ORACLE1
FAX + 1.650.506.7200
oracle.com

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com. Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com/oracle-communications/

https://www.linkedin.com/showcase/oracle-comms/

Integrated Cloud Applications & Platform Services

White Paper **A Cloud-Native Journey for Telecommunications**Cloud NativeA Cloud Native Journey for TelecommunicationsA Cloud Native Journey For Telecommunications
February 2019
Author: Todd Spraggins
Contributing Authors: Shirin Esfandiari

Oracle is committed to developing practices and products that help protect the environment