ORACLE
Communications

Cloud Native Monetization

Benefits of Deploying Oracle Communications Billing and Revenue Management on Cloud Native Infrastructure

An Oracle whitepaper

October 2020 | Version 1.00

SAFE HARBOR STATEMENT

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. 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, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

This document is provided for information purposes and should not be relied upon in making a purchasing decision. The contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose.

THIS DOCUMENT IS NOT PART OF A LICENSE AGREEMENT NOR CAN IT BE INCORPORATED INTO ANY CONTRACTUAL AGREEMENT WITH ORACLE CORPORATION OR ITS SUBSIDIARIES OR AFFILIATES.

Failure to adhere to these benchmarks does not constitute a breach of Oracle s obligations. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document.

TABLE OF CONTENTS

INTRODUCTION

The Importance of Cloud Native Monetization

Modern monetization systems for 5G converged charging, communications, media, cloud and digital goods and services markets will need to take optimal advantage of compute, network, and storage infrastructure to operate and scale efficiently and grow as the business demands. These requirements translate into the need for monetization systems to support a cloud native containerized, orchestrated and multi-service deployment architecture. Diving a little deeper into these terms:

- Containerized – portable, lightweight application components that can be rapidly spun-up and down, taking advantage of core Linux kernel capabilities (namespaces and control groups) that are much more efficient than Hypervisor-based virtualization technologies.
- Orchestrated – managing the lifecycle of the multiple containers that comprise the application by abstracting the underlying infrastructure and providing built-in scalability and resiliency through the definition of a declared state of deployment that is maintained by the orchestration engine. Container orchestration technology enables simpler application management and dramatically reduces operational complexity.
- Multi-service – an architecture where application components representing specific functional concerns are deployed in separate containers to aid scalability, resiliency, and observability. Core business functions should be deployed as multi-replica containers.

The above-mentioned cloud native characteristics will be essential to take advantage of today's DevOps aligned Continuous Integration/Continuous Delivery (CI/CD) toolchains. This will help to minimize time-to-value, scale efficiently, and improve the overall operational quality of deployments. Some business benefits of cloud native deployment include:

- Significantly reduced vanilla installation time
- Rapid environment replication for development, testing and faster root cause analysis of potential issues
- Self-healing capabilities for greater service availability
- Simpler updates with less downtime
- Efficient scaling that takes maximum advantage of the available compute resources (nodes)
- Faster launch of reliable market offerings by taking advantage of CI/CD toolchain integration and automation

Oracle Communications Billing and Revenue Management (BRM)

Oracle Communications Billing and Revenue Management (BRM) is a proven, reliable, modern monetization solution that is foundational to the digital commerce operations of leading telecommunications and enterprise customers. BRM provides converged, real-time charging as part of an end-to-end revenue management solution for supporting the key business processes of generation, capture, collection, and analysis of revenue.

- Flexible service, industry and partner-enabled business model support.
- Faster innovation: rapidly launch digital offers with design-time flexibility.
- IT agility: modern cloud native deployment model with low total cost of ownership.

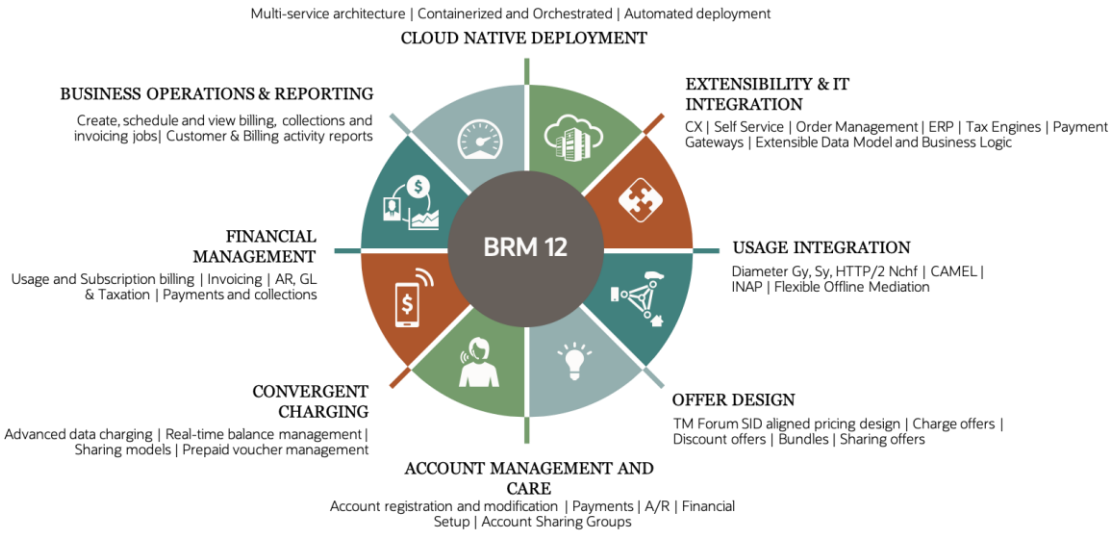

Figure 1 – BRM feature coverage

Oracle Converged Charging System

BRM Release 12 has been designed with the 5G future in mind supporting the 5G service-based architecture while offering a feature rich converged charging, billing and revenue management system that is both network grade and extensible (figure 1). BRM provides a foundation for monetizing 5G Non-Standalone Architecture based services and Standalone Architecture slice-based offerings, available in a DevOps aligned cloud native deployment model to significantly reduce costs and accelerate innovation

BRM's Elastic Charging Engine (or ECE) is an advanced, network grade and high performance in memory converged charging grid with built-in resilience. It includes a comprehensive set of southbound core network integration points including support for Diameter and the 3GPP 5G HTTP/2 REST charging message, as well as flexible offline mediation capabilities.

ECE is integrated into the core BRM revenue management functionality (shown in the upper half of the diagram) which provides flexible and extensible business logic across customer management, service management, subscription management, re-rating, billing, invoicing and accounts receivable.

The high-level functional architecture is shown in figure 2. Shown in the lower half of this diagram we have BRM's converged charging capabilities, supporting integrated online and offline charging in a single architecture.

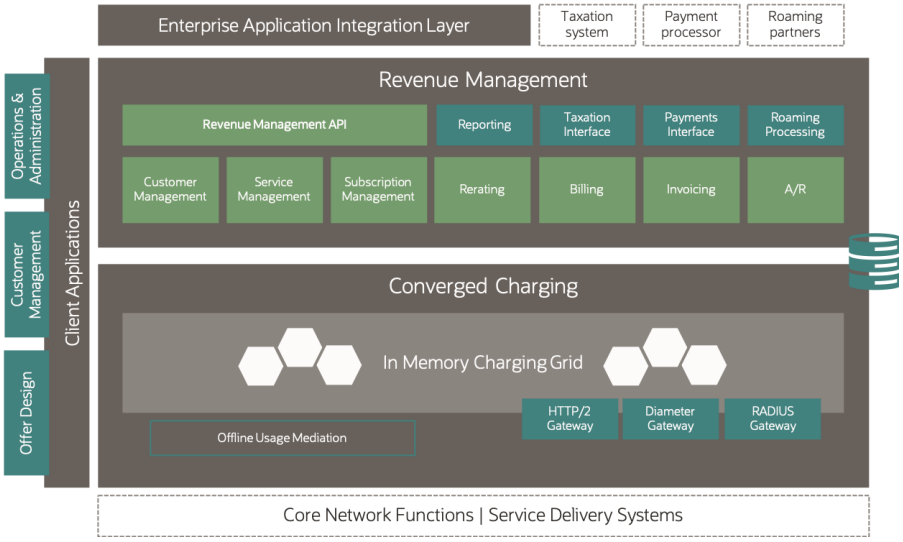

Figure 2 – BRM Functional Architecture

BRM exposes a large number of functions with a documented SDK to aid integration into northbound enterprise business systems. Integration methods include web services and via the BRM JCA resource Adapter. In addition, BRM has data managers supporting integration with external taxation databases and payment processing systems.

Supporting the core functionality, BRM has a comprehensive set of client applications covering operations and administration, customer management and offer design.

DEPLOYING BRM ON CLOUD NATIVE INFRASTRUCTURE

The Features and Extensibility of BRM, the Agility and Efficiency of Cloud Native

BRM can be configured to run as a cloud native application in a containerized and orchestrated deployment architecture, taking advantage of cloud native infrastructure and DevOps CI/CD tooling to enable service providers to design, test and deploy services more quickly, operate more efficiently, and grow with the business (figure 3).

The features and extensibility of BRM, the agility and efficiency of cloud native

Figure 3 – BRM Cloud Native container images uses standard cloud native techologies

BRM has a multi service architecture with each service provided as a Docker image for deploying as a run time container in a Kubernetes cluster on cloud infrastructure. This deployment option takes advantage of industry accepted cloud native technologies such as Docker for the container runtime, Kubernetes for container orchestration, Helm for packaging and deployment, and the EFK stack, comprising of ElasticSearch, fluentd and Kibana for logging. BRM's cloud native deployment retains the features and extensibility of BRM whilst taking advantage of the agility and efficiency of cloud native infrastructure and tooling.

A high level logical view of BRM's cloud native deployment option is shown in figure 4 below.

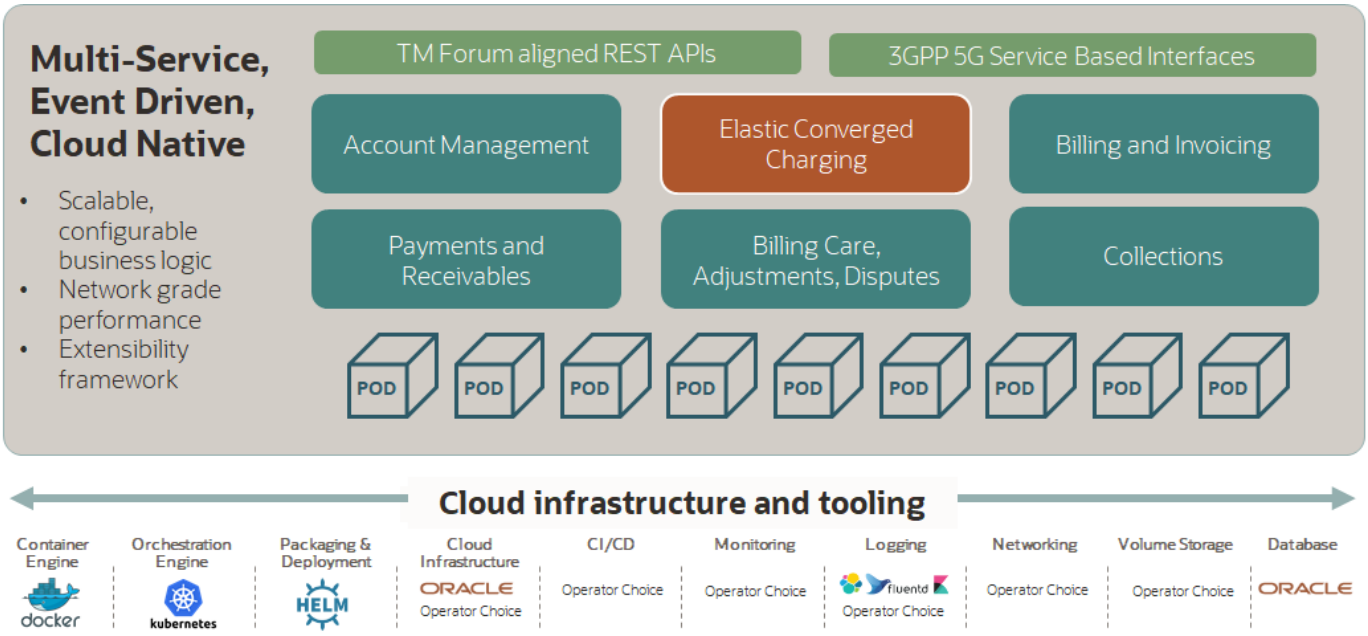

Figure 4 – BRM Cloud Native deployment high level logical view

BRM Multi Service Architecture

Figure 5 shows BRM's multi-service architecture. Each BRM service, running as a Docker container, is deployed as a Kubernetes POD, the fundamental building block of Kubernetes. Many of the core BRM services can be deployed and managed as multiple replicas within a Kubernetes replica set, which enables efficient scaling and aids resiliency. Kubernetes operates on the principle of *declared state* – you tell it how many instances of a POD you want running and it will ensure that the number of replicas matches the declared state.

Figure 5 – BRM Cloud Native Multi-Service Architecture

The BRM application Kubernetes cluster requires that cloud native infrastructure and tooling is in place. This includes cloud compute infrastructure, such as Oracle Cloud Infrastructure (OCI), a CI/CD automation pipeline, monitoring and logging software, Kubernetes cluster networking and volume storage, which is accessed by PODs via a persistent volume claim (PVC). Note that the Oracle database sits outside of the BRM application Kubernetes cluster, typically deployed in a separate subnet.

Example Cloud Native Deployment topology in Oracle Cloud Infrastructure (OCI)

Cloud native BRM has been designed to be deployed in both public and private cloud infrastructure. An example deployment model is shown in figure 7, which shows BRM deployed in Oracle Cloud Infrastructure (OCI). Note that not all details are shown in this diagram and it is intended to be for illustrative purposes only, rather than represent a specific deployment architecture. Deployment approaches will be dependent upon specific business requirements. For deployment in OCI, it is recommended to configure BRM application worker nodes in three different fault domains (FDs) within an Availability Domain (AD). The diagram shows an Oracle RAC cluster in a dedicated subnet, using Active Data Guard for replication with a secondary database in another AD, depending upon business requirements. From an ECE perspective, Coherence federation is used to ensure that the charging grid cache is replicated to a secondary availability domain (not shown).

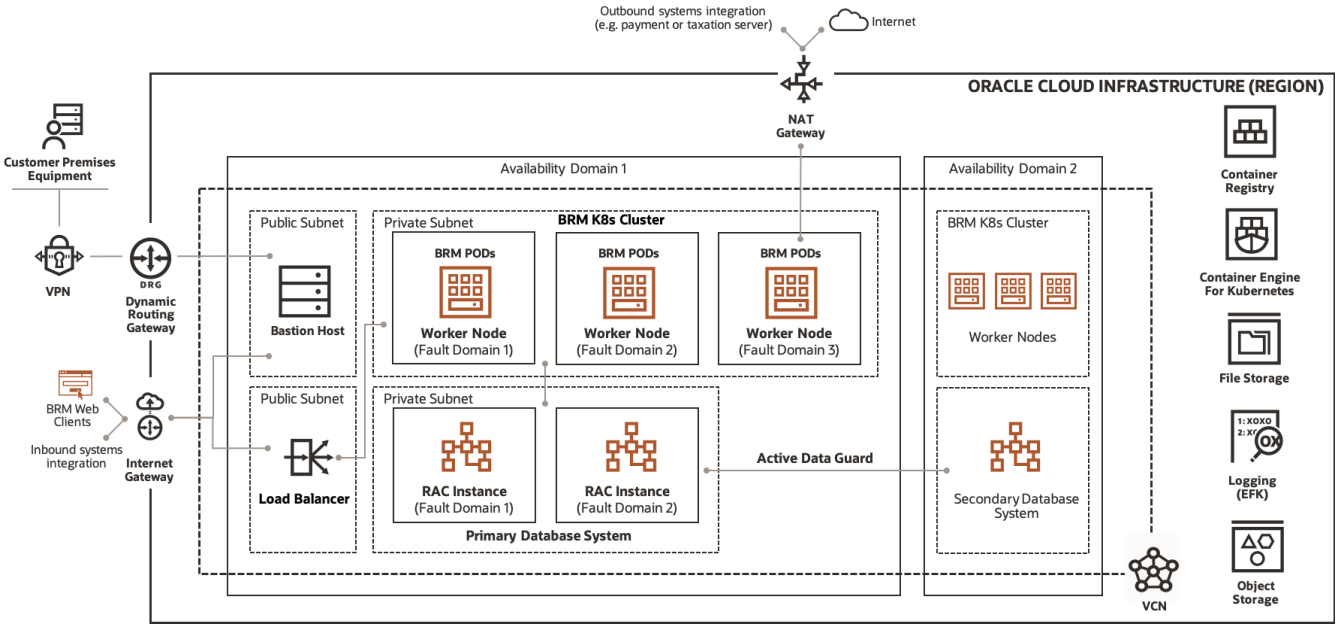

Figure 7 – Example BRM Cloud Native OCI Deployment Model

A bastion host is typically configured in a public subnet to allow access to the BRM worker nodes from the customer's network (for example via SSH). The BRM web clients and inbound integrations connect to the load balancer through the Internet gateway. Outbound integrations, for example payment or taxation server integration, are managed through the NAT gateway, which is also used by the worker nodes for outbound access to the Internet (for example to perform `yum` updates).

BRM CLOUD NATIVE BENEFITS

Cloud native deployment of BRM has the potential to provide significant benefits with regards to initial deployment, ongoing operations and reduced time to market to launch and monetize new products and services. This section provides a high-level summary of these benefits. It is important to understand that the degree of savings depends on specific deployment architecture, workload, degree of product customization and overall adoption of DevOps CI/CD and cloud native tooling across the service provider's broader IT estate. Figure 8 summarizes indicative deployment and operational efficiencies of cloud native BRM when compared to a traditional BRM deployment model.

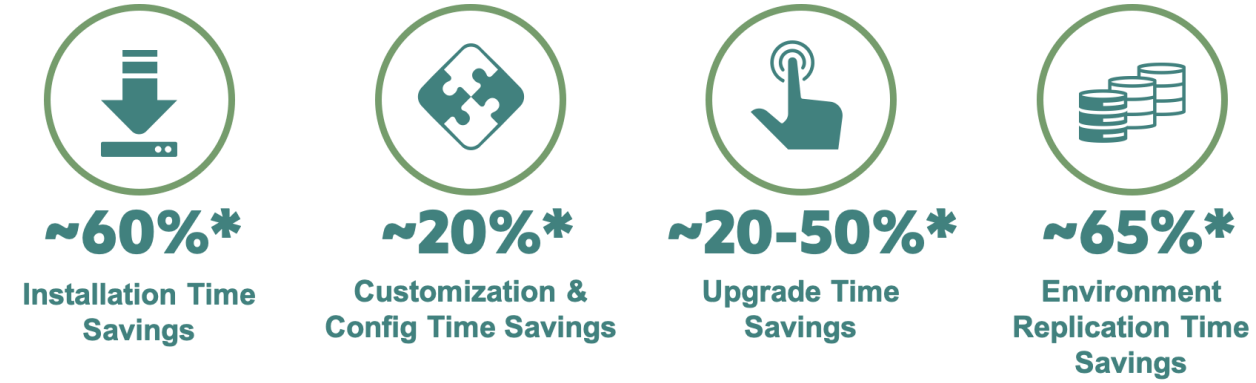

*Figure 8 – Indicative operational savings compared to traditional on-premise BRM deployment**

Deployment Benefits

Faster Installation

The initial deployment time for BRM in a cloud native environment is significantly reduced through the use of Helm (the package manager for Kubernetes applications). A separate Helm chart is provided for the database schema deployment flexibility for those customers that already have a database in place. Helm unifies the installation experience, eliminating the need for customers to manually download software technology dependencies, such as compilers, the JDK, and Perl scripts to a compute server. The reduction in installation steps can provide significant time and cost savings for customers when setting up test and production environments.

Table 3 describes the Helm charts that are included in the BRM cloud native deployment package.

HELM CHART	DESCRIPTION
oc-cn-init-db-helm-chart	Initializes and upgrades the database schema for the BRM server functions.
oc-cn-op-job-helm-chart	Creates WebLogic server domains for Billing Care and Business Operations Center (BOC).
oc-cn-helm-chart	Deploys the core BRM server components, Pricing Design Center (PDC) and Pipeline Configuration Center (PCC).
oc-cn-ece-helm-chart	Deploys the Elastic Charging Engine (ECE) and its services. Sets the connection with the BRM server functions and Pricing Design Center. Configures the sharing of persistent volumes with the BRM server functions.

Table 3 – Helm Charts included in the BRM cloud native deployment package

From the perspective of BRM's Elastic Charging Engine (ECE), cloud native deployment simplifies the Coherence cluster deployment, eliminating the need for customers to define the primary and secondary driver machines, sync software binaries across the server nodes or VMs and configure the cluster topology (in `eceTopology.conf`).

The customer would define the number of replicas by overriding the default values from `values.yaml` with adding custom values in `override-values.yaml` and then run the Helm install, which ensures that all components are started in sequence and places the system into a usage processing state.

ECE high availability is configured by default, with each Elastic Charging Server node consisting of three POD replicas and each protocol gateway node consisting of 2 POD replicas.

BRM's WebLogic applications, such as Pricing Design Center (PDC), Billing Care (BC) and Business Operations Center (BOC) are significantly faster to install as WebLogic is already baked into the base Docker images.

Based on indicative Oracle lab metrics, and assuming the readiness of underlying cloud native infrastructure and the database, vanilla BRM application **installation time could be significantly reduced by approximately 60%.**

Operational Benefits

Environment Replication

Once initial deployment has taken place, separate environments can be rapidly replicated by simply overriding the Helm configuration with environment specific values in the `values.yaml` override file and performing a Helm install. This is a rapid process that can offer significant savings, particularly to those customers that require large numbers of environments to be spun up within their DevOps CI/CD processes.

Rapid environment replication can also have significant benefits to aid root cause analysis of any potential issues observed in production, by replicating the production environment in a dev/test environment for rapid problem resolution through investigation, patching and validation.

Based on indicative Oracle lab metrics, BRM environment **replication time could be reduced by as much as 65%,** depending on the specific customer DevOps processes and toolchain adoption.

Customization

As discussed earlier a key benefit of the BRM cloud native architecture is the ability to integrate into a service provider's CI/CD workflow to support automated build, test and deployment processes. Figure 9 shows the high-level flow involved when extending BRM through new customizations, showing Oracle Cloud Infrastructure as the continuous delivery and deployment infrastructure. BRM's cloud native deployment option supports BRM's extensibility and customization capabilities. Custom C and Java code can be used to extend the core business logic capabilities, develop new client applications and extend the Billing Care UI.

A customer or partner would take the base BRM images, store them in their registry of choice, and then create a new image with the customizations using Docker layering. After the build and test process these new images will be pushed to the container registry for deployment into test and production under the co-ordination of the DevOps CI/CD pipeline tooling.

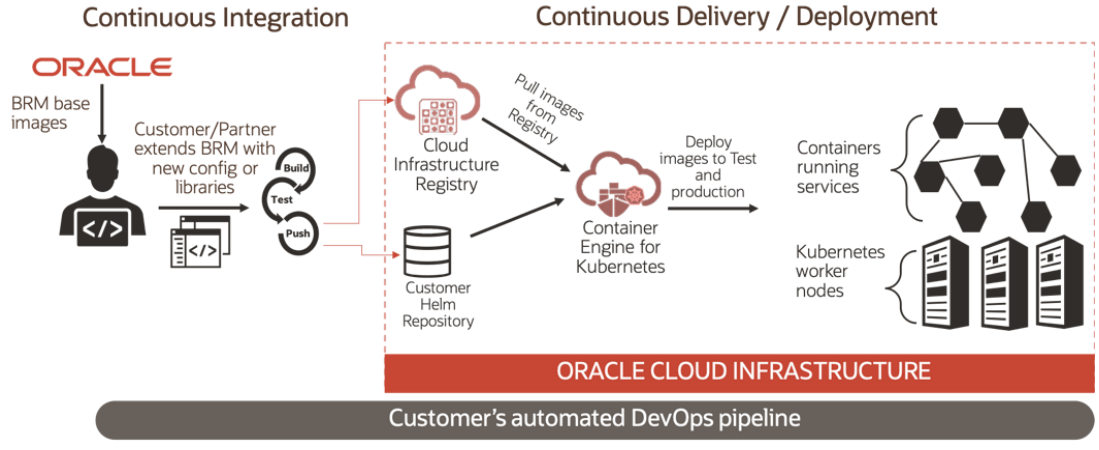

Figure 9 – Customizing and extending BRM via CI/CD toolchain integration

The time for customization bug detection, fixing, testing and deployment can be significantly reduced when deploying BRM alongside a CI/CD toolchain and supported by DevOps automation. It is anticipated that deploying BRM in an automated cloud native environment could offer **customization implementation efficiencies of 20% or greater.**

Patching

The benefits of CI/CD automation significantly reduces the patching cycle (whether for bug fixes or security updates) from patch updates on a dev and test environment, automated testing and validation, through to deployment to staging and final production environments. **Depending upon the nature of the patch (e.g. whether a one-off patch or patchset, degree of data model changes etc), it is anticipated that the efficiency savings achieved for patching could be as high as 50%.**

Patchset Upgrades

Patchset upgrades in a BRM cloud native deployment are faster and more efficient. The upgrade process consists of running the `init_db` container in upgrade mode, which performs the database schema upgrade, and then perform a Helm upgrade with the new set of images. During the upgrade, the same port and IP (via the Kubernetes service proxy) continues to service requests.

For ECE, Kubernetes will take care of rolling restart of the ECE pods and each pod will automatically check the Coherence cluster status to make sure partition rebalancing is complete during the restart.

Configuration Changes

Core BRM business logic configuration changes are very straightforward and require near zero service downtime as a rolling upgrade of the BRM server-side components is possible.

Each BRM application service has its configuration externalized via a Helm chart and Kubernetes ConfigMaps. *Externalized configuration* is an important characteristic of cloud native applications, decoupling the Docker images from the configuration (which can be stored in a version control system). Kubernetes makes rolling out (and rolling back) new configuration changes easy through the use of Helm, which creates and updates deployments and services (figure 10).

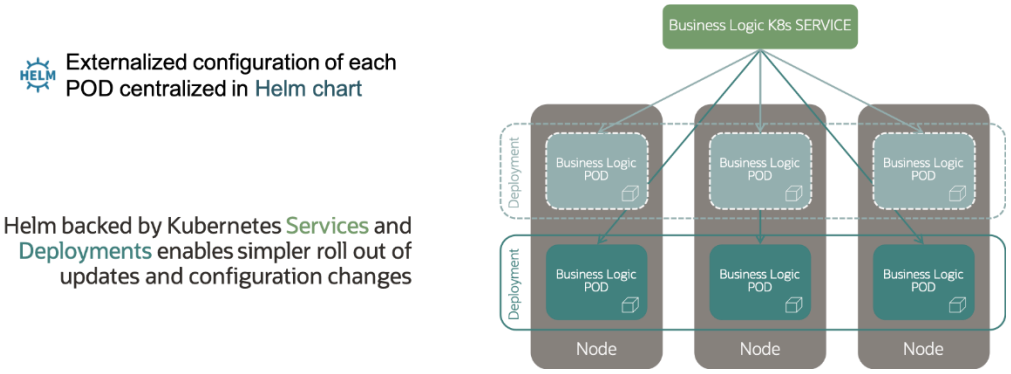

Figure 10 – Near zero downtime configuration changes

For example, changes in the `BRM pin.conf` file can be achieved by invoking `helm upgrade` specifying the updated `override-values.yaml` file. Similarly, after invoking changes in `config_business_params` for example, by running helm upgrade, the new PODs will read their configuration from the database with no overall service loss due to the rolling upgrade process supported by Kubernetes.

Whenever a BRM ConfigMap or Secret file is changed, BRM supports the configuration of a POD to automatically update its deployment specification and hence support a rolling deployment. This is achieved via the use of annotations in the Kubernetes deployment YAML descriptor.

A business logic configuration change that would have typically needed a system restart is now supported by a rolling update of the CM or DM, providing near zero BRM ecosystem downtime.

Batch Job Invocation

Kubernetes jobs are provided for key revenue management batch applications, such as billing jobs, invoicing, and import/export pricing utilities, making configuration and invocation efficient and straightforward, and adhering to cloud native best practices by avoiding the need to `exec` into the POD containers to directly run the jobs.

Logging and Regular Operational Benefits

The use of industry standard cloud native tooling provides improved efficiency for regular day to day BRM operations to ensure smooth running of the system. For example, all BRM PODs use the ElasticSearch, Fluentd and Kibana stack for centralized logging and visualization (figure 11).

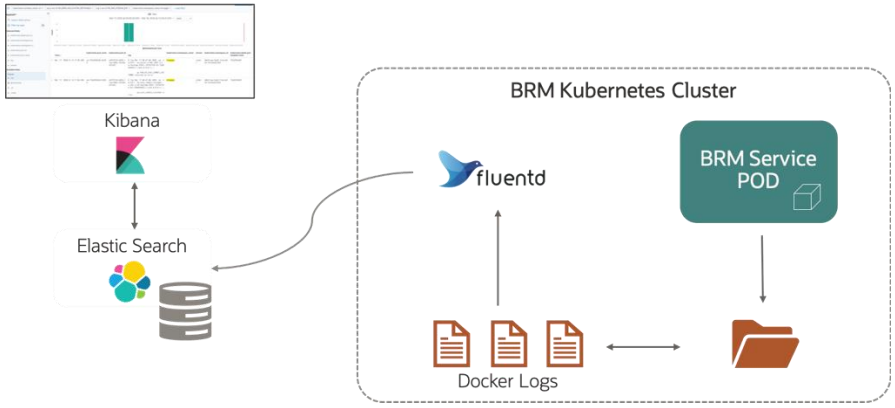

Figure 11 – BRM cloud native logging environment using the EFK stack

Each fluentd agent pod is deployed within a Kubernetes DaemonSet, which deploys one pod for each node in the BRM application cluster. The BRM application PODs write logs to `stdout`, which are collected by fluentd and persists them into the ElasticSearch object store, ready for visualization by Kibana.

Efficient Scalability

A key principle of cloud native applications is the ability to efficiently scale multiple service instances, whether that is "up" to support new capacity demands for example charging operations during busy seasons or "down" to free up capacity when certain applications are not required to be running (for example for communications service providers, billing and invoicing jobs may only be required to be run during certain periods in a month).

The core BRM business logic services, for example the connection managers and data managers, are multi-replica (managed by a ReplicaSet). Load balancing across multiple CMs and DMs is inherent and is maintained by the Kubernetes service proxy, which abstracts the client service requests from the specific instances of the backend serving PODs. This provides significant efficiency improvements when load sharing multiple CMs and DMs across workloads. Figure 12 illustrates the efficient scaling of BRM's multi-replica PODs. The number of desired replica's can be specified via configuration, with Kubernetes perfoming scaling to meet this state. From the command line business logic replicas can be scaled with the command:

```
kubectl scale deployment/cm – replicas=4
```

Changing the number of replicas will create or shutdown pods without client disruption.

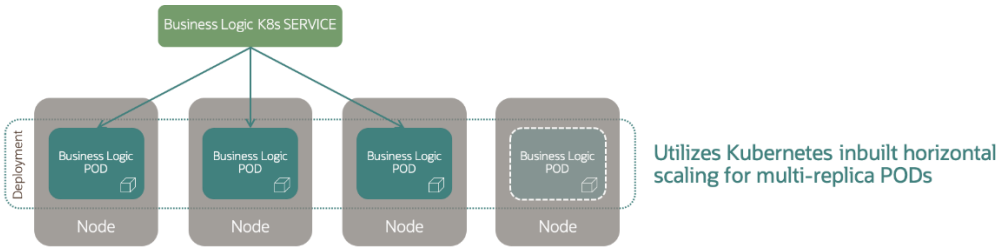

Figure 12 – Efficient BRM business logic scalability using Kubernetes replicas

Self-Healing

Kubernetes is inherently self-healing, always operating on the principle of declared state and ensuring the correct number of POD replicas are running. The BRM business logic and data management services will gracefully terminate if Kubernetes decides to reduce the number of running PODs.

Future Upgrades

It is anticipated that deploying BRM in a cloud native environment could offer **future upgrade efficiencies of 20% or greater,** depending on the specific deployment context and project scope (for example the degree of customization) and the extent to which cloud native and DevOps is embraced.

Future minor and major upgrades will follow the same high-level process as described previously for patchset upgrades (specific times and upgrade process details will be release specific).

Time To Market Benefits

Deployment of BRM on cloud native infrastructure within a DevOps CI/CD toolchain can contribute to overall time to market benefits beyond the deployment, operational and upgrade savings described above. When deployed alongside an automated DevOps toolchain and supporting business processes, and in combination with adoption across a service provider's broader IT estate, cloud native BRM can contribute to the faster launch of new services and resultant earlier revenue generation. The combination of BRM's inherent flexibility and extensible business logic with the efficiency gains from a fully automated Continuous Integration/Continuous Deployment (CI/CD) toolchain and broader cloud native IT adoption across the enterprise, can shorten the time from concept to launch, which will be critical as we enter the "digital native" and 5G era.

SUMMARY

Oracle Communications Billing and Revenue Management is a modern monetization solution that provides real time converged charging for any business model. It is available in a cloud native deployment option, supporting a Kubernetes-orchestrated containerized multi-service architecture to facilitate continuous integration / continuous delivery and DevOps practices. Best deployed on Oracle's Cloud Infrastructure with its autonomous capabilities, adaptive intelligence and machine learning cyber security, BRM cloud native has the option of being deployed in public or private cloud infrastructure environments that provide standard cloud native tooling and can support the Oracle database.

BRM's cloud native deployment option enables BRM to take advantage of the efficiencies of modern cloud compute infrastructure and automated toolchains, supporting easier and faster installations, agile service development and launch, simplified configuration updates, rapid environment replication and efficient scaling. As a result, BRM can enable service providers to design, test and deploy services faster, improve operational savings and scale with the business.

Figure 13 – Summary of cloud native BRM Benefits

GLOSSARY OF TERMS

AD	An Availability Domain in Oracle Cloud Infrastructure
CI/CD	Continuous Integration / Continuous Delivery
CM	The BRM Connection Manager service
DM	The BRM Data Manager service
ECE	BRM's Elastic Charging Engine
EFK	ElasticSearch, Fluentd, Kibana – a popular logging stack
FD	A Fault Domain in Oracle Cloud Infrastructure
Helm	A package and deployment manager for Kubernetes applications
OCI	Oracle Cloud Infrastructure
POD	Kubernetes deployment unit
PVC	Persistent Volume Claim
RAC	Oracle Real Application Clusters

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 facebook.com/oracle twitter.com/oracle

Cloud Native Monetization
October, 2020
Authors: Richard Hallett