MATRIXX Cloud Native Implementation

The MATRIXX Digital Commerce Platform (DCP) is a cloud native, real-time monetization engine delivering industry-compliant rating and charging functionality alongside a rich array of digital commerce capabilities, including subscription management, event streaming and management, personalization and digital payments.

At its heart is a 3GPP-compliant Online Charging System (OCS) supporting 2G, 3G and 4G cellular networks, and a Converged Charging System (CCS) supporting 5G, both delivered from the same platform.

The MATRIXX DCP can be implemented in Network Edge, Telco Cloud, Private and Public Cloud environments. It is deployed as a cloud native, containerized application, orchestrated and managed by Kubernetes. It can be delivered on 'bare metal' or virtualized infrastructure.[1]

With 5G driving the need for flexible performance and elastic scalability, cloud native deployments are becoming the primary choice for most organizations, supporting:

- **Platform Abstraction** – The ability to take advantage of the practically-infinite scale of the cloud, or to deploy within a secure private data center, the cloud native architecture benefits from the portability and ubiquity of the Kubernetes ecosystem.
- **Container-Based Deployments** – Easily orchestrated, fast start up times and lightweight implementation that enables high utilization of the underlying hardware.
- **Loosely-Coupled Microservices** – Optimally sized autonomous business functions that independently scale, are easily upgraded, ephemeral and easily replaced.
- **DevOps** – Highly-automated orchestration allowing for continuous integration and delivery.

[1] Refer to Cloud Native Infrastructure Requirements for details, page 6

Deployment Benefits Summary

MATRIXX DCP provides market-leading, real-time monetization and is architected for 5x9's availability. The components of the platform are based on a loosely-coupled processing pipeline where each element can be separately scaled and tuned for resilience and performance. MATRIXX DCP and Kubernetes together deliver stateful, cloud native monetization in a fully containerized environment.

MATRIXX Digital Commerce Platform Cloud Native Architecture

MATRIXX DCP architectural layers include API gateways, routing agents, processing logic, in-memory database nodes and event publishing. Each layer is separately scalable and can be tuned for the desired level of resilience. They are enabled by efficient communications between each node, with data volume reduced by localized databases and the co-location of critical business logic.

Kubernetes takes these concepts and applies them to any cloud environment.

KEY BENEFITS OF MATRIXX DCP CLOUD NATIVE DEPLOYMENT

MATRIXX enables a high-performance, consistent and replicated in-memory database across a cluster of containers. Typically, containers and microservices imply a focus on stateless web applications. The deployment of stateful applications and databases is less straightforward and requires additional steps to operationalize. Stateful containers typically require much greater coordination of cluster management, failover, replication and ordered upgrades.

MATRIXX DCP is architected for high performance and 5x9's availability. The lightweight and high-performance software is fully aligned to a microservices environment. Containers should be lightweight, easily managed and ephemeral — easily created and destroyed without consequence. This is all supported by MATRIXX DCP's high performance, loose coupling and fully replicated database. With Kubernetes, MATRIXX DCP delivers a cloud native, stateful application in a fully containerized environment.

MATRIXX DCP is deployed on Kubernetes using 100% native controls without plugins or an external provisioning overlay. This is made possible by the microservices design of the platform, even including the stateful in-memory database components, and achieved through the use of Deployments and the Operator Framework.

Deployment

Kubernetes is an open source container orchestration engine designed to manage reliable, distributed, containerized applications at scale. The smallest and simplest object in Kubernetes is a pod, but most applications are formed of multiple types and instances of pods. Kubernetes provides various out-of-the-box controllers to manage different types of pods. The DCP makes extensive use of a Deployment and ReplicaSet to ensure that for a particular function in the platform a required number of pods are available at any point. The Deployment is a controller that also includes support for 'rolling upgrades' which is important in this context so that pod version changes are introduced in a managed fashion.

Configuration

Helm is a tool used in cloud native environments to help manage the complex set of configurations typical in Kubernetes deployments. It provides a mechanism for deploying Kubernetes manifest files in a controlled manner. This includes the ability to upgrade and rollback an installation. The Kubernetes manifest files are packaged as a *Chart*. A Chart packages the manifest files required for an application but also provides templating to allow these to be customized for the given deployment.

Helm Charts are used to define a given deployment sub-domain and Engine topology along with the number of pods for all the stateless pods to be created under the various Deployment ReplicaSets. All pods within the reference architecture come with a Helm chart along with higher level items such as "engine," "sub-domain" and a top level "matrixx" chart, allowing for simple "helm install matrixx" type commands.

By providing a values.yaml file per environment, Helm makes it very easy to spin up and tear down various environments such as dev, test, preproduction and production.

Helm also manages upgrades by specifying container versions that any given environment should use. When a new version of a given container is available and is to be deployed to a given environment, the values.yaml file is altered to state the new version for that container and a "helm upgrade" performed.

MATRIXX Software supports the use of Helm for managing various deployment configurations and performing upgrades. This simplifies the spinning up of new deployments with different pod configurations and topology, for test as well as production.

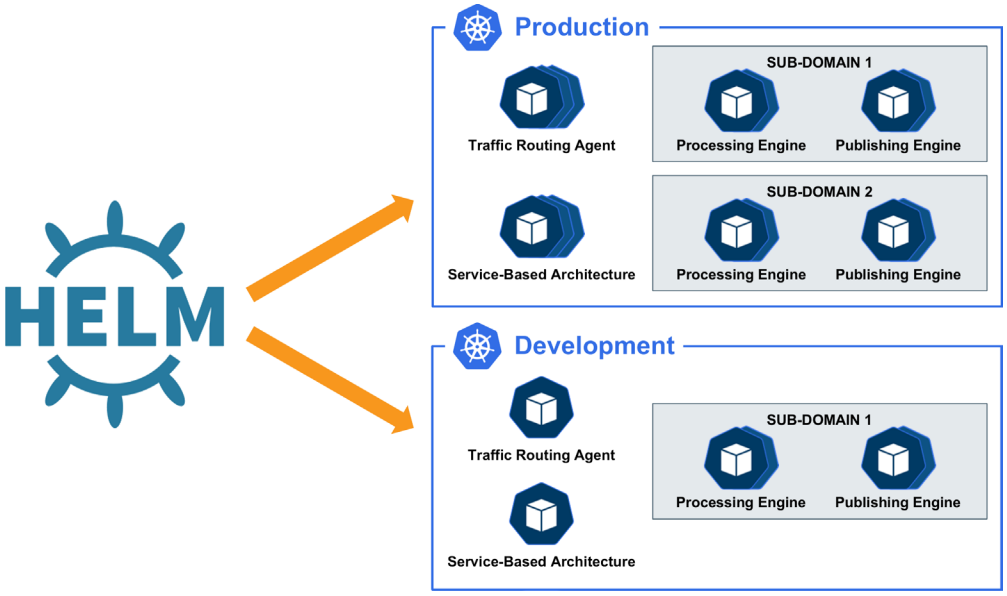

Figure 2: The role of Helm in deploying MATRIXX DCP Software

CI/CD Pipeline

The move to a Cloud Native architecture brings challenges along with many benefits. For example, the DevOps process is impacted by the increased complexity of managing several microservices instead of a singular application. The number of code pipelines to be managed could grow exponentially. The build and release process therefore has to adapt to manage dozens of microservices, and the various versions and compatibilities between them, with a quicker time-to-market. This makes automation essential.

As containers are immutable, all changes to MATRIXX DCP software are managed as part of an automated CI/CD process which allows for complete traceability. MATRIXX recommends the use of a git-based and Jenkins orchestrated pipeline for managing all changes to production systems via automated test environments.

For example, deploying updated pricing or a new engine configuration in addition to deploying new software versions should all be managed via a pipeline to remove manual error.

When it comes to the option of continuous deployment or continuous delivery, a majority of telcos are likely to practice continuous delivery, and to manage their own testing and production software updates, given the 5 9's environment they operate in.

Allowing multiple vendors to directly change production software will likely be too risky for most without a robust coordination process. Telcos will still expect code changes to be managed and automated all the way into production via a pipeline which they will expect to own and manage and which will be utilized across the various cloud native vendors and applications in their environment.

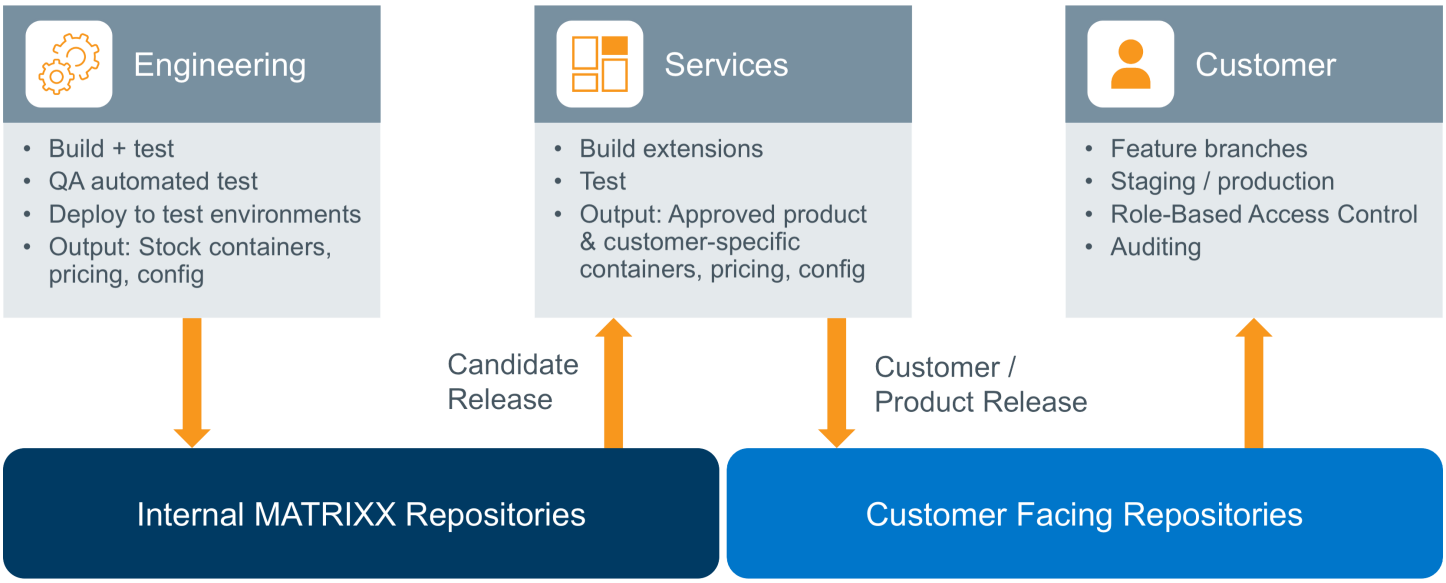

Figure 3: MATRIXX Software Reference CI/CD Pipeline

CLOUD NATIVE INFRASTRUCTURE REQUIREMENTS

- **Kubernetes Distribution** – A production grade, fully supported Kubernetes distribution for public or private cloud such as Amazon EKS, Google GKE, Redhat Openshift or VMware Tanzu Kubernetes Grid. K8s 1.18 or above.
- **Worker Nodes** – Linux x86 based public or private cloud or bare metal worker nodes.
- **Persistent Volumes** – For local storage solutions (EBS, local SSD, etc), small space for host OS and container images only. For shared storage solutions (EFS, NAS, NFS/SAN etc.), MATRIXX DCP is 'ReadWriteMany (RWX)' and requires fast shared storage for transaction log files. Standard shared storage is used for archiving transaction log files, checkpoints and events.
- **Ingress Controllers/Load Balancers** – Production-grade cloud platform network load balancer for non-http ingress traffic (such as diameter) along with an appropriate application load balancer or separate ingress controller and network load balancer for HTTP traffic (5G Service Based Architecture and Business API Gateway).
- **Networking** – Standard Kubernetes Container Network Interface for networking. Optionally, Single Root I/O Virtualization (SR-IOV) networking for the processing pods for improved network performance can be used.

ABOUT MATRIXX DIGITAL COMMERCE

MATRIXX Digital Commerce is the industry's leading cloud native rating and convergent charging solution. Architected with the performance and resiliency of a network function and the configurability of an IT application, MATRIXX DCP unifies IT and Networks to provide operators the agility to operate at web scale. With its API-first design, lightweight, no-code configuration, and microservices architecture, MATRIXX Digital Commerce is easy to configure, fast to deploy and capable of serving multi-network environments from a single, extensible platform. Massively scalable and highly efficient to operate, MATRIXX DCP enables operators to successfully automate operations, monetize new services and hyper-scale offerings, all at web-speed.

matrixx.com