Reference Implementation for a Cloud Native Digital Enterprise

Version Winter-2020

Original Author

- Lakmal Warusawithana | Senior Director - Technology Evangelism | WSO2, Inc | lakmal@wso2.com | @lakwarus

> *This paper presents a reference implementation for a cloud native digital enterprise architecture described in the Reference Architecture for a Cloud-Native Digital Enterprise. We will focus on an implementation using Kubernetes and WSO2's API-led integration platform.*

Introduction

The recent constraints on businesses have pushed organizations to accelerate their plans for moving operations to the digital world—often shrinking timelines from years to months. In the process, APIs have emerged as the products of the 21st century. As discussed in the Reference Architecture for a Cloud Native Digital Enterprise, the combination of cloud native technologies and an API-led integration platform significantly increases productivity by enabling agility, flexibility, and scalability through automation and services.

This paper discusses a reference implementation for a cloud native digital enterprise using two industry-leading, open-source technology stacks: Kubernetes and WSO2's API-led integration platform.

Reference Implementation: A Cloud Native Digital Enterprise

Figure 1 - Cloud Native Digital Enterprise

In this reference implementation, Kubernetes provides the cloud native platform capabilities, and WSO2's API-led integration solution offers the required integration and API management capabilities for a digital enterprise.

Kubernetes can install on top of any private, public, or hybrid cloud infrastructure. WSO2's API-led integration platform (WSO2 API Manager and WSO2 Enterprise Integrator) can be installed on top of Kubernetes with native support through the WSO2 Kubernetes API Operator. This native integration provides the necessary automation, scalability, and operations as well as giving API-led integration capabilities.

Kubernetes

Kubernetes, an open-source cloud native orchestration platform or a framework, provides a complete set of cloud native abstractions and a toolkit to build a scalable, flexible solution that aligns with business growth. It is a mature open-source project that is governed by the Cloud Native Computing Foundation (CNCF). Kubernetes' rich cloud native abstractions help define the scalable architecture. Its toolkit helps to orchestrate containerized applications into a multi-host, multi-cloud distributed platform by providing the necessary infrastructure and automation.

The Kubernetes architecture consists of a control-plane and a set of worker nodes. Typically, control-plane components deploy in master (control-plane) nodes, and work-loads deploy on worker nodes. The control-plane is responsible for storing information regarding nodes, monitoring the nodes, container workload scheduling, etc. If you are new to Kubernetes, I would recommend reading the Kubernetes Architecture appendix section.

Kubernetes' rich cloud native abstractions give the flexibility to create a wide variety of cloud native platforms. *PODs, Services, Deployments, ConfigMaps, Labels,* and the *Horizontal POD Autoscaler* are some of the key cloud native abstractions. The Kubernetes cloud-native abstractions section discusses details of the Kubernetes abstractions used in the API-led integration platform to build a cloud native digital enterprise.

Custom resources and custom controllers

A custom resource is an extension of the Kubernetes API; it does not ship with the default Kubernetes installation. A Kubernetes cluster admin can add/remove custom resources independently of the cluster itself. Once a custom resource installs, users can create and access its objects using kubectl, just as they do for built-in resources such as *PODs*.

Custom controllers help keep the current state of Kubernetes objects (defined in CR) in sync with the desired state. The controller interprets the structured data as a record of the user's desired state and continually maintains it. So, custom resources provide a dedicated declarative API when you combine a custom resource with a custom controller.

Custom resources and custom controllers give flexibility and an immense power to architects to build cloud native platforms on top of Kubernetes.

WSO2's API-led integration platform

WSO2's API-led integration platform is an open-source, market-leading enterprise solution that supports full API lifecycle management and integration. The platform was named a Leader in The Forrester Wave™: API Management Solutions, Q3 2020 report.

WSO2's offering comes with an API designer and publisher, a developer portal, a key manager, an API analytics server, an API gateway, an enterprise integrator, and a Kubernetes API operator.

Full API lifecycle management

In an era of digital transformation, APIs are a strategic investment to any organization. They play a significant role as both technical enablers and business drivers. WSO2 API Manager provides state-of-the-art web interfaces, e.g., WSO2 API Designer and Publisher, for API development and management. It is 100% compliant with the Open API Specification, helping API creators to develop, document, scale, and version APIs, while also facilitating more API management-related tasks such as publishing, monetizing, and promoting APIs.

Figure 2 - WSO2 API Designer and Publisher

In addition to the graphical interface, WSO2's API-led platform has a command-line tool, i.e., apictl, which automates full API lifecycle management functionalities.

The communication hub for an API ecosystem

The API developer portal is a hub to discover and onboard developers with low friction experiences. WSO2's developer API portal enables developers to find APIs, test them before subscription and consumption, calculate monetization with specific metrics, view feedback, and feature requests from consumers through forums, and more.

Figure 3 - WSO2 API Developer Portal

The developer portal's web UI is a single page React application written on top of well-defined APIs. Organizations can customize the web UI to align with organization needs or build their own branded developer portal by consuming these developer APIs.

Enabling QoS through traffic regulation

When productizing APIs, it is vital to have API traffic regulation capabilities to provide consumers with different service levels. WSO2 Traffic Manager helps to define throttling policies and enforce them in API gateways. WSO2 Traffic Manager comes with a dynamic throttling engine that processes throttling policies in real-time and enables rate-limiting of API requests to prevent APIs from being overwhelmed, allowing API owners to enforce a limit on the number of requests.

Business intelligence

Obtaining meaningful business insights on how APIs are behaving is critical in every business. WSO2 API Analytics Server is capable of generating all kinds of business intelligence. In addition to statistical graphs, its real-time event processing engine can identify abnormalities and alert users about potential malicious attacks.

API security and anomaly detection

Security is paramount when exposing business capabilities via APIs. WSO2 Key Manager manages all clients, security, and access token-related operations. It supports OAuth 2.0, JWT, Basic Auth, Mutual SSL, and API-Key-based authentication mechanisms.

Figure 4 - WSO2 API Key Manager

Sometimes, traditional security measures, such as authentication and authorization, alone are not enough. Compromised security tokens can cause a data breach or malicious activities. You can use WSO2's integrated artificial intelligence-based solutions to control these kinds of incidents.

Malicious payloads could come with authorized and authenticated requests, and identifying and stopping them immediately at an early stage is critical. You can set rules to analyze the request payload (e.g., JSON or XML) and prevent them from passing through the gateway to the respective backends.

Anomalies can be identified by analyzing the access patterns to the APIs. Artificial intelligence and machine learning techniques help to identify these kinds of security threats. These advanced techniques help digital enterprises proactively monitor and prevent possible risks and malicious activities.

Composition, integration, and mediation

WSO2 Enterprise Integrator is capable of playing multiple roles in your enterprise architecture. It can be used as an Enterprise Service Bus (ESB), a streaming data processor, and a microservices integrator. WSO2 Micro Integrator supports both centralized (ESB style) and decentralized (microservices, cloud native) architectural styles. WSO2 Streaming Integrator allows you to implement streaming ETL (extract, transform, and load), change data capture (CDC), and process large files and real-time APIs.

WSO2 Micro Integrator provides a unique low code approach to microservices integration. Its rich, full connectors help integrate with a wide range of legacy systems. The offering's easy-to-use code integration approach speeds creating composite integration microservices, while enabling users to reap the benefits of MSA.

Figure 5 - Micro integration

Policy enforcement at scale

The API Gateway is the main policy enforcement point. It supports OAuth 2.0, JWT, Basic Auth, Mutual SSL, and API-Key based authentication mechanisms and enables IT organizations to enforce rate limits and throttling policies.

In addition to the policy enforcements, you can perform edge integrations, like mediation, transformation, etc., depending on your requirement. Suppose your integration is complex or wants to have a highly scalable architecture. In that case, WSO2 recommends using a facade pattern by using an enterprise micro integrator along with the API Gateway.

WSO2 API Micro Gateway is optimized for microservices and supports all three deployment patterns: shared-gateway, private-jet gateway, and sidecar gateway (described in the reference architecture paper).

Figure 6 - WSO2 API Micro Gateway Deployment Patterns

Depending on the API strategy, users can choose an effective API gateway deployment model. In some use cases, we can handle all API traffic by using a shared API Gateway cluster. Users can also group APIs and distribute them in different API gateway clusters. These groupings can be done based on API functionalities or regions that they are accessing. These can be deployed in private jet mode or shared mode depending on scalable requirements.

You can automate gateway deployment by using a Kubernetes platform that deploys across multi regions. Having a scalable gateway cluster is critical if you are looking for a global operational digital enterprise.

Automation through the Kubernetes operator pattern

WSO2 Kubernetes API Operator provides a fully automated experience for cloud native API management. It introduces a set of custom resources to deploy and manage API-led integration artifacts into Kubernetes easily.

Figure 7 - WSO2 Kubernetes API Operator

WSO2 Kubernetes API Operator can create and deploy WSO2 API Micro Gateway and WSO2 Micro Integrator by reading Swagger definitions or integration definitions provided by the API developer/publisher. These gateways and integrators automatically deploy into the defined Kubernetes cluster along with the necessary Kubernetes deployment artifacts.

API observability

Unlike monolith architecture, auditing and tracing are challenging problems in decentralized architectures such as MSA. Performance issues, errors, and exceptions are unfortunate events that may occur in a production environment. To identify such an event, observing the production environment is essential.

Often, microservices do not act alone and they interconnect to each other through the API calls. WSO2 API Gateway can work with cloud native observability tools, such as Prometheus, Jaeger, and

Fluentd, to analyze these captured metrics, statistics, and data to produce meaningful visualizations to understand system behavior.

Prometheus

Prometheus is an open-source system monitoring and alerting toolkit that is governed by the CNCF. Prometheus can set up to work natively with a Kubernetes cluster.

The API gateway exposes a metrics endpoint that gives metrics related to service level data, client-side data, and some process-related data such as memory and CPU usage. These metrics can then feed into Prometheus. In addition to this, users can configure an analytics dashboard, such as Grafana, to visualize Prometheus's metrics.

In addition to observability dashboards, you can use these collected metrics to scale backend services and API gateways by extending the Kubernetes Horizontal Pod Autoscaler.

Figure 8 - Custom metrics-based autoscaling

Jaeger

Jaeger is a distributed tracing system toolkit that is governed by the CNCF. Jaeger support comes out-of-the-box in WSO2 API Gateway. It is well suited for distributed transaction monitoring, performance and latency optimization, service dependency analysis, and many operational problems when moving to a distributed architecture.

Fluentd

Fluentd is an open-source data collector for unified logging layers. The CNCF governs the Fluentd project. By default, containers do not persist any operational data (such as logs, etc.), and when containers terminate, they will lose data. You can configure the Kubernetes cluster to push all aggregated logs from microservices, API gateways, and API integrators to Fluentd. These aggregate data can be used to perform different analyses to identify any operational problems.

GitOps

GitOps is a way of implementing continuous deployment for cloud native applications. It combines the functionalities of Git and continuous deployment tools and provides a developer-centric experience when operating infrastructure.

In a digital enterprise, publishing an API is not just a simple process. It involves creating APIs and then deploying them in a lower API management environment to go through different testing rounds (developer testing, stress testing, QA testing, etc.). Once these tests are successful, they move to the production environment.

Figure 9 - API CI/CD Automation with GitOps

Each deployment environment has a specific Kubernetes cluster configured with WSO2 Kubernetes API Operator and WSO2 API management components. Depending on the enterprise requirements, API management components, such as WSO2 API Publisher, WSO2 API Traffic Manager, WSO2 API Key Manager, and WSO2 API Developer Portal, can be configured.

WSO2 API Manager is 100% compatible with Open API Specification (formerly known as Swagger) based API design and development. Developers create API artifacts and commit them to a Git version control system. Then a CI/CD build pipeline, such as Jenkins, CircleCI, Bamboo, can configure to pull artifacts and create container images and API/integration deployment descriptors. The *apictl* (API control CLI tool) tool helps to script the automation. Generated deployment descriptors can be committed and pushed to the deployment repository.

Then, the deployment pipeline triggers and calls WSO2 Kubernetes API Operator (by using apictl/kubectl commands) and deploys the necessary API gateways and enterprise integrators in the given environment Kubernetes cluster. This automated process helps to carry out different testing rounds (developer testing, stress testing, QA testing, etc.).

After completing environment testing, you can then promote to upper environments (e.g., stage to prod) by merging into the relevant Git branch. With this mode, If you want to deploy a new application or update an existing one, you only need to update the repository; the automated process handles everything else.

Conclusion

By becoming digital enterprises and digitalizing value chains, companies in any sector can integrate and expose their business capabilities as APIs. These APIs should be secured, managed, observed, and monetized. An API-led integration platform is essential for digital enterprises, whether they start with greenfield or brownfield projects.

Kubernetes, an open-source cloud native orchestration platform, provides a complete set of cloud native abstractions and a toolkit to build a scalable, flexible solution that aligns with business growth.

WSO2's API-led integration platform is an open-source, market-leading enterprise solution that supports full API lifecycle management and integration.

The platform and its native Kubernetes integration capabilities provide an effective digital enterprise architecture to increase productivity by enabling agility, flexibility, and scalability through automation and services.

Appendix

Kubernetes Architecture

Figure 10 - Kubernetes Architecture

Control-plane components

Dedicated components manage control-plane functionalities.

ETCD: is a database that stores all node information and container workload information as a key-value pair in a highly available manner.

Kube-scheduler: is responsible for scheduling container workloads for the worker nodes by considering requesting computing resources of the workloads, available resources in the nodes, the type of workload allowed in the worker nodes, and the other management policies and constraints.

Kube-controller manager: consists of multiple controller managers. Node controller is responsible for onboarding new nodes to the cluster and handling node unavailability. The replication controller ensures that the desired number of container workloads are running all the time. In addition to these two, many other controllers help manage different functionalities.

Kube-API server: is the primary communication center for all cluster components. It is responsible for orchestrating all operations within the cluster components and also exposes the Kubernetes API that is used by the external users who manage the container workloads. Worker node components also communicate with the control-plane via the Kube API server.

Worker node components

Worker nodes should be able to communicate with control-plane components and provide necessary information to manage container workloads. Worker nodes have two major management components.

Kubelet: is an agent running in each node and responsible for scheduling container workloads with the control-plane instructions and reporting back the status of the nodes and the container workloads.

Kube-Proxy: is running in each worker node and ensures all communication among the container workloads run in multiple nodes.

Kubernetes cloud native abstractions

Kubernetes' rich cloud native abstractions give the flexibility to create a wide variety of cloud native platforms. This section will only discuss the abstractions (or building blocks) used in the API-led integration platform to build the cloud native digital enterprise.

POD

Figure 11 - POD

A *POD* is a single instance of our containerized application. Kubernetes will not directly deploy containers, and it encapsulates within a *POD*. A *POD* is the smallest unit that scales out depending on the load. A *POD* can consist of multiple containers. The containers in a *POD* are automatically co-located and co-scheduled on the same node in the cluster. Containers living within the POD can share the same network space, storage space, and process space.

ReplicaSet

ReplicaSet helps to define the number of replica *PODs* running at any given time. It monitors the PODs and creates a new one when a failure occurs in the current set. It also helps to scale out the application by increasing the number of replicas.

Deployment

Using the Kubernetes *Deployment* object, we can define the desired state (e.g., number of replicas of a given version). The Deployment Controller changes the actual state to the desired state at a controlled rate and the deployment strategy. We can use Deployment rollouts to provision newer versions and rollback if the current state of the Deployment is not stable.

Namespaces

Namespace helps to divide physical clusters into multiple virtual clusters. The namespace provides a scope to Kubernetes naming objects. Names of resources need to be unique within a namespace but not across namespaces. Namespaces cannot be nested inside one another, and each Kubernetes resource can only be in one namespace.

Resource quota

Resource quota provides constraints that limit aggregate resource consumption per namespace. Cluster administrators can define resource quota allocated to a namespace, then the quota system tracks usage to ensure it does not exceed hard resource limits defined.

Services

Kubernetes *Service* exposes an application running on a set of *PODs* as a network service. When we deploy an application, each *POD* gets its IP address. However, to maintain the desired state in a *Deployment*, it can create and destroy *PODs* dynamically. Kubernetes *Service* provides a single IP address and a DNS name to a set of *PODs* in deployment and can monitor *POD* behaviors and update their IP addresses dynamically. Also, this given Service endpoint is capable of routing traffic to *PODs* in a load-balancing manner. In addition to this, Kubernetes *Services* is essential to have a service discovery mechanism without modifying your application code.

Depending on how you want to expose your application, you can configure Kubernetes Services to behave in the main three modes. The ClusterIP service type helps expose your applications within the cluster.

Figure 12 - ClusterIP Service

If you want to expose applications outside of the cluster, you can use the NodePort type or Loadbalancer type. NodePort will expose applications by opening a port in every node in the cluster and map and forward traffic to the application port.

Figure 13 - NodePort Service

The Loadbalancer type will create a load balancing endpoint that is configured with the cluster's relevant cloud provider and exposed through the cloud provider's load balancer.

ConfigMaps

Every reusable application has some kind of configuration. Decoupling configuration with the application code helps to use the same application code in different environments. *ConfigMaps* allow you to decouple configuration artifacts from container image content to keep containerized applications portable. Secrets

Like the configurations, we need to pass sensitive information, such as passwords, OAuth tokens, and ssh keys, to the application to perform the required tasks. Rather than placing sensitive information in a container image, we can use a Kubernetes Secret object. By default, Secrets obfuscate with Base64 encoding, but you can make them more secure (encrypt) by using deference techniques.

Horizontal POD Autoscaling

Figure 14 - Horizontal Pod Autoscaler

The capability of scaling each microservice gives the real benefit of the MSA. The Kubernetes Horizontal POD Autoscaler helps to automatically scale microservices by scaling the number of Pods in a deployment, replica set, or stateful set based on observed CPU utilization (or, with custom metrics support, on some other application-provided metrics).

Kubectl

The kubectl command-line tool lets you interact and control Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs. Kubectl can be used either in imperative mode or in a declarative way.

Imperative

In imperative mode, we have to run a series of commands and follow a given step to complete a task. In Kubernetes, you can use the kubectl command to do things in imperative mode. Here are some examples:

- Create a POD : `kubectl run --image=nginx nginx`
- Create a Deployment : `kubectl create deployment --image=nginx nginx`
- Create a Service : `kubectl expose deployment nginx --port 80`
- Edit existing object: `kubectl edit deployment nginx`
- Scaling a Deployment: `kubectl scale deployment nginx --replicas=3`

All of the above are imperative approaches to managing objects in Kubernetes.

Declarative

In the declarative model, we can list our requirements (in a required format), and the system will be capable of completing the required task. In this mode, we can create a set of files that define the expected state of applications and services on a Kubernetes cluster. With a single kubectl apply command and Kubernetes should set the desired state by reading the given configuration files.

Example: `kubectl apply -f nginx.yaml`

Labels and Selectors

Labels are key/value pairs that are attached to objects, such as pods. Labels can be used to organize and to select subsets of objects in a loosely coupled fashion. Application deployment and management often consist of multi-dimensional entities, such as multiple release tracks, multiple tiers, environments, etc. Mapping these into labels will help in day-to-day management and operational activities.

Example labels:

- "release" : "stable", "release" : "canary"
- "environment" : "dev", "environment" : "qa", "environment" : "production"
- "tier" : "frontend", "tier" : "backend", "tier" : "cache"
- "partition" : "customerA", "partition" : "customerB"
- "track" : "daily", "track" : "weekly"

The label selector is the core grouping primitive in Kubernetes. You can use label sectors to perform operation activities like the rolling update, rollback, drain nodes to put in maintenance, scaling, etc.

Health check probes

Heath probes are critical to achieving the high availability required to run our application in production. Kubernetes uses three health check probes to determine the necessary action to achieve auto-healing and high availability.

Liveness probe

Defining liveness probes help to determine any deadlock scenarios, where an application is running but unable to make progress, and restart the container to overcome the situation.

Readiness probe

Readiness probes help determine when a container is ready to start accepting traffic. One use of this signal is to control which PODs use backends for services. When a POD is not ready, it is removed from Service load balancers.

Startup probes

Some applications have longer startup times. In such situations, setting up a startup probe helps to disable liveness and readiness checks until it succeeds, making sure those probes do not interfere with application startup.

Rollout

When you create a new deployment, Kubernetes will attach a revision related to it. When you roll out a new version of your deployment, it will update the revision history. You can see all the rollout versions by using the following commands.

```
kubectl rollout status deployment.v1.apps/nginx-deployment
```

```
kubectl rollout history deployment.v1.apps/nginx-deployment
```

Kubernetes support two deployment strategies. The first strategy is to recreate, destroy all currently running instances, and create the same number of new instances, not a zero-downtime deployment strategy.

Kubernetes default is the rolling-update deployment strategy. This strategy will not take down all currently running instances at once; instead, it will take down a current version and bring up a newer version one by one. The best way to do this is by updating the declarative application file (yaml file) and executing the kubectl apply command. It will trigger a new rollout and a new revision is created.

Rollbacks

Even though we perform thorough testing, sometimes we need to roll back to a stable state due to a late-found error. You can use the following command to rollback a deployed version.

```
kubectl rollout undo deployment.v1.apps/nginx-deployment
```

Alternatively, you can rollback to a specific revision by specifying it with --to-revision:

```
kubectl rollout undo deployment.v1.apps/nginx-deployment --to-revision=2
```

Abstractions

Icon	Name	Description
Component	Microservices and serverless components	Core business logic, aggregation and service composition, transformation.
Component	Gateways	API gateways, ingress gateways, mesh gateways, micro integrators, exposed APIs, events and streams, policy enforcement points
Component	Legacy and data services	Databases, existing systems, registries and repositories, user stores, business processes
SaaS EPR	External endpoint	Access using APIs, events, and streams, cloud systems, and SaaS
Front end Client	API consumers	Mobile apps, reactive apps, API consumers
Desktop Client	API consumers	Mobile apps, reactive apps, API consumers
Mobile Client	API consumers	Mobile apps, reactive apps, API consumers
Bot Client	API consumers	Bots, API consumers
IOT Client	API consumers	IoT devices, apps, API consumers
WSO2 API Gateway	WSO2 API Gateway	WSO2 API Gateway, WSO2 Micro API Gateway
WSO2 API Developer Portal	WSO2 API Developer Portal	Developer portal for API discovery, an API marketplace
WSO2 API Publisher	WSO2 API Publisher	API Publisher Portal, API design, develop, API package, API publish
WSO2 API Key Manager	WSO2 API Key Manager	Manage API keys and all security tokens
WSO2 API Traffic Manager	WSO2 API Traffic Manager	Rate Limiting, Traffic control policy defining portal
WSO2 API Analytics Server	WSO2 API Analytics Server	Real-time analytics server, generate API business intelligence, anomaly detection engine, and altering
WSO2 Micro Integrator	WSO2 Micro Integrator	Enterprise integrator, mediation, transformation engine

Icon	Name	Description
WSO2 Stream Integrator	WSO2 Stream Integrator	Enterprise stream integration
WSO2 Enterprise Workflow	WSO2 Enterprise Workflow	Workflow engine
Kubernetes Deployment	Kubernetes Deployment	An object that manages replica sets, rollouts, health checks
Kubernetes POD	Kubernetes POD	A single instance of our containerized application
Kubernetes Service	Kubernetes Service	Exposes an application running on a set of Pods as a network service
Kubernetes Custom Resources	Kubernetes Custom Resources	A custom resource is an extension of the Kubernetes API
Kubernetes ConfigMap	Kubernetes ConfigMap	Contains the configuration data
Kubernetes Secret	Kubernetes Secret	Contain sensitive data like keys, credentials, certs
Kubernetes Horizontal POD Autoscaler	Kubernetes Horizontal POD Autoscaler	Scale workload based on observed CPU, memory utilization
Kubernetes Namespace	Kubernetes Namespace	Divide physical clusters into multiple virtual clusters
ETCD	ETCD	A database that stores all node information and container workload information as key-value pairs
Kubernetes Cloud Control Manager	Kubernetes Cloud Control Manager	Kubernetes Cloud Control Manager
Kubernetes Control Manager	Kubernetes Control Manager	Kubernetes Control Manager
Kubernetes Kube Proxy	Kubernetes Kube Proxy	Running in each worker node and ensures all communication among the container workloads
Kubernetes Kubelet	Kubernetes Kubelet	An agent running in each node
Kubernetes scheduler	Kubernetes scheduler	Scheduling container workloads for the worker nodes

Icon	Name	Description
Kubernetes API Server	Kubernetes API Server	Primary communication center for all cluster components.

References

- https://kubernetes.io/
- https://wso2.com/api-management/
- https://wso2.com/integration/micro-integrator/
- https://github.com/wso2/k8s-api-operator
- https://www.gitops.tech/