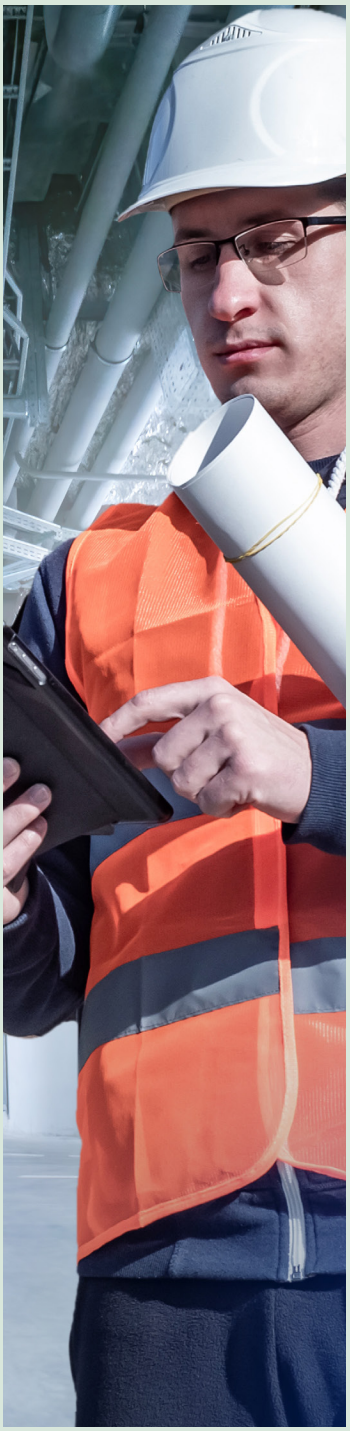
WHITE PAPER



ARCHITECTURE GUIDANCE ON CONTAINERS

Abstract

Containers are rapidly becoming the platform of choice across the industry for hosting modern applications and services. According to Cloud Native Computing Foundation, Containers exemplifies the approach to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds.

Container programs provide a robust Container Hosting Platform which is inherently more resilient, secure, and operationally viable compared to the previous generation of hosting platforms.

Infosys®
Navigate your next

Table of Contents

Introduction

- Containers are software packages which consist of necessary elements which are required to run an application in any given environment. They are a form of operating system virtualization which can run anything from a small microservices or software process to a large application.
- Kubernetes is an open-source container orchestration system. It helps in scaling, automating software deployment and management of environments.
- Containerization helps provide three key technical advantages with the potential to benefit to the business. These are - increased predictability and dependability, increased speed from development to deployment, increased operational velocity.

Drive into the world of containers

- Industries are driving goals looking for IT cost optimization through innovation with an intention of achieving digital transformation across the enterprise.
- The terms Container and Kubernetes are the new buzzwords in the IT world. Companies that adopt the technology see themselves as innovators in the field and hope to position themselves as highly competitive amongst their peers. It helps built container capabilities and realize the value of containers and Kubernetes platform through orchestration of early adopter applications.

Factors driving containers adoption in Financial Industry/Securities

Business drivers

The business landscape is becoming more competitive as the securities markets are going through major advancements and modernization. This, coupled with additional challenges such as cost pressures and cyber security threats, is mandating a paradigm shift on the financial services and regulatory businesses.

Application Modernization

Containers focuses on a very secured, identical, stable, and speedy delivery, these features help close gaps between business and technical architecture to help support financial markets achieve and realize modernization goals. It progresses at application modernization holistically across the Core Clearing, Risk and Solutions businesses. If we look at the core business functionalities such as data ingestion, data transformation, matching, clearing, data reporting etc. the core application component of all technical stack can use containerization approach and benefit from its capabilities such as ubiquitous deployment, inherent security, and speed of delivery.

Operational risk factors

Every organization have a huge legacy code base which supports their business in a most efficient manner and runs on a highly available platform. However, we have the same challenges of people, processes, and technology as prevalent in our industry. It has built quite a sizable amount of technical debt over time, which adds to the risk of legacy systems. We need to aggressively identify end of life technologies and processes and gaps in skillsets of our people.

- Container's capabilities built thus far is well positioned to overcome challenges and become integral part of the foundational capabilities to drive digital transformation.

How to accelerate container adoption in Enterprises

Container capabilities can be viewed from two different perspectives namely, Platform Provider and Platform Consumer view.

Platform Provider

Provider creates a strategic roadmap to adopt, develop and enhance container platform capabilities and support the growth of containers across IT organization for an enterprise

Typically - providers offer a central Container-as-a-service (CaaS) environment for use by taking care of

a) Cluster administration: Kubernetes Cluster consist of various factors which help manage environment. There are various factors which cater them are Admin role, Manage CPU and memory.

b) Certificate management: Kubernetes provide an API which lets provision of TLS certificate signed by Certificate Authority (CA). These certificates can be used by your workloads to establish trust.

c) Namespace RBAC: In Kubernetes, Cluster Roles and Roles define the actions a user can perform within a cluster. Roles and Role Bindings are placed inside of and grant access to a specific namespace, while cluster roles and cluster role bindings do not belong to a namespace and grant access across the entire cluster. or namespace, respectively.

d) Health checks: Kubernetes consist of two kinds of health checks viz the Readiness Probe and the Liveness Probe.

Both use the below types of probes -

I. TCP

II. HTTP

III. command execution

Platform Consumer

Consumer develops, configures, and enhances application services to support high availability deployments for business functions using container capabilities.

Consumer focus on on-boarding applications onto an existing Container-as-a-service CaaS environment by leveraging container and Kubernetes features. For instance,

a) ReplicaSet: It maintains a stable set of replica Pods running at any given time. It mainly guarantees the availability of a specific number of identical pods.

b) Scale a StatefulSet: It manages scaling of containers, increasing or decreasing replicas

c) Persistent volumes (PV) storage: A Persistent Volume is a piece of storage in the cluster. It can be provisioned by either by an administrator or by dynamically using Storage Classes.

d) Autoscaling:

Scaling talks about automatically increasing or decreasing our application based on workload.

I. Horizontal Pod Autoscaling

Horizontal scaling means that the response to increased load is to deploy more Pods. A Horizontal Pod Autoscaler automatically updates a workload resource (such as a Deployment or StatefulSet), with the aim of automatically scaling the workload to match demand.

II. Cluster Scaling

Cluster scaling involves dynamically adjustment of the number of nodes and pods to scale as per demand of the user. Along with Horizontal Pod, Cluster scaling can be handled automatically. The Autoscaler automatically adds new nodes in the cluster if there is requirement for a new node and it finds pending pods waiting. Similarly, in case it identifies there are unused nodes waiting for a definite amount , it will reduce them. This helps to reduce cost of the user and you pay as per requirement only.

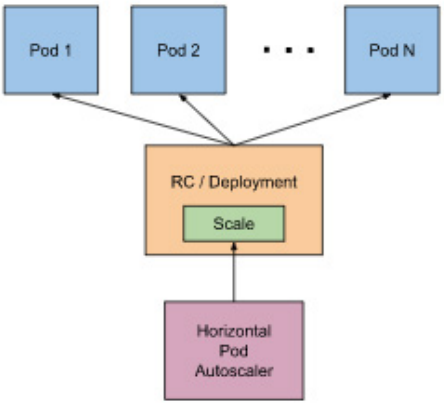
e) Liveness and Readiness Probe

Liveness and Readiness probes are used to control the health of an application running inside a Pod's container. Liveness probes determines when it is in need to restart a container. Readiness probes determines the availability of a container to accept request.

f) CronJobs

A CronJob object is just like an entry in crontab in Unix/Linux. It runs a job periodically on a given schedule.

To meet business requirements of clients in most efficient and timely manner, it is critical to create standard implementation patterns and promote the culture of reusability.



Platform Options

Container is the common system component on which applications can be layered on and deployed on top of underlying private or public cloud infrastructure.

CaaS provides services that manage containers at larger scale including scaling, starting, stopping, and organizing containerized workload.

Memory, CPU and API Resource Management:

a) Limits on Namespace with Default Memory Requests Configuration

We should specify a default memory resource limit for a namespace. This ensures every new Pod in namespace has a memory resource limit configured.

b) Limits on Namespace with Default CPU Requests Configuration

Always determine default CPU resource limits for a namespace. This will ensure that every new Pod in namespace has a CPU resource limit configured.

c) Define Memory Constraints for Namespace

Always determine range of valid memory resource limits - Maximum and Minimum, for namespace. This will ensure that every new Pod in namespace is within the specified range configured.

d) Define CPU Constraints for Namespace

Always determine range of valid CPU resource limits - Maximum and Minimum, for namespace. This will ensure that every new Pod in that namespace is within the specified range configured.

e) Define Memory Quotas for a Namespace

Define overall memory limits for a namespace.

f) Define CPU Quotas for a Namespace

Define overall CPU resource limits for a namespace.

g) Define a Pod Quota for a Namespace

Restrict how many Pods you are allowed create within a namespace.

Providers available in Market

Kubernetes has become the open source industry standard for container orchestration. Apart from Vanilla Kubernetes, there are various Kubernetes products for managing containers, including OpenShift, Tanzu.

OpenShift

OpenShift is a software product developed by Red Hat. Kubernetes incorporates Linux containers which are orchestrated and also managed by them. OpenShift provides a Console which has developer and administrator-oriented views. It also provides CLIP that supports superset of actions that the Kubernetes CLI provides. Its platform to provide CaaS services in most secured and resilient way to inherit. Container platform is fully deployed in Active-Active configurations across regional data centers. Each platform has High Availability (HA) configuration for local availability within a region.

VMware Tanzu

VMware Tanzu is built for enterprises that must deploy and manage applications at scale. It is not a single product, but rather a suite designed to modernize infrastructure on which they run. It is a VMware product. It thus keeps strong connection to the VMware virtualization portfolio.

How to Integrate Container Platforms into Enterprise

By integrating container platform with common services, we can optimize cost with improve value. There are various services available to optimize features like monitoring, security, storage, build pipelines and so forth.

Following are some common services Container platforms can integrate to achieve operational excellence:

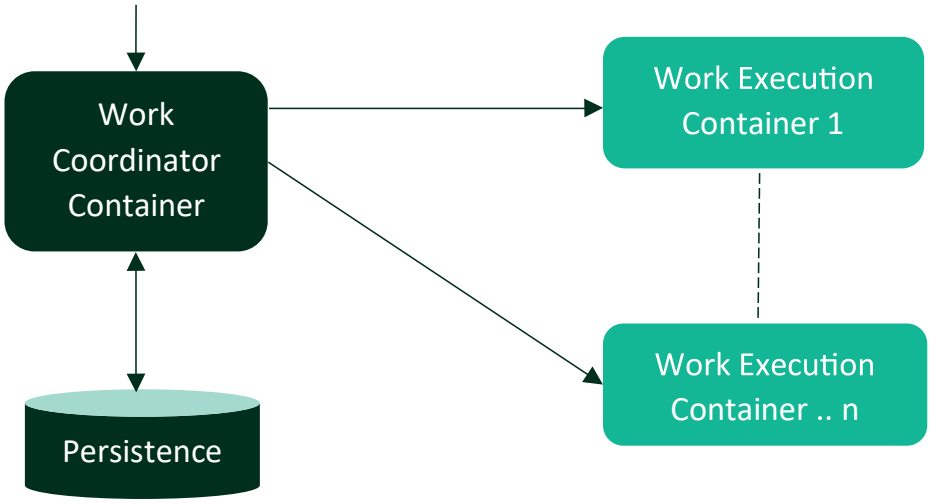
Feature	Common services to be integrated
Secrets Management	Vault
Vulnerability scanning	AquaSec, Trivy, Clair, Kube hunter
Alerting/Monitoring	ServiceNow, Kubelet, cAdvisor, Prometheus, Jaeger, Kubewatch
Performance monitoring	AppDynamics
Operations Manager	Micro Focus
Logging	Splunk, Loki, Zebrium, ELK Stack, Fluentd
Security logging	Qradar
Build and deploy	Jenkins
Persistent Storage	File / Block / S3 compatible Object storage services

Architectural Patterns for Building Applications with Containers

As we modernize our application portfolio and decompose monolithic applications into loosely coupled services, we need to evaluate and develop new design patterns for application functionalities.

We need to create new design patterns focused on communication mechanisms and interactions between the container pods and managing platform.

Majority of applications could have a need for such distinct patterns, some of which are mentioned below:



1. Work Queue Design Pattern

   Work queue pattern propose that you divide a large task into minor tasks to minimize running time and complexity. This patter is ideal for a container that run as a batch service until completion or scheduled to run periodically.

2. Singleton Design Pattern

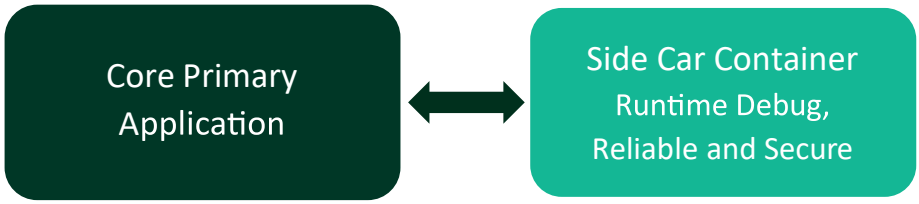
   There could be a singleton service which ensures only one instances of service is active at a time and still need to be highly available. In this case, we use a single-containerized structure which expose a HTTP service. It is thus a good option when containers are expected to solve only one given problem statement.

3. Multiple pods behind load balanced end point

   Typical Web service design pattern where multiple stateless containers run simultaneously behind a load balanced service end point. Platform manages discovery and routing of the service requests.

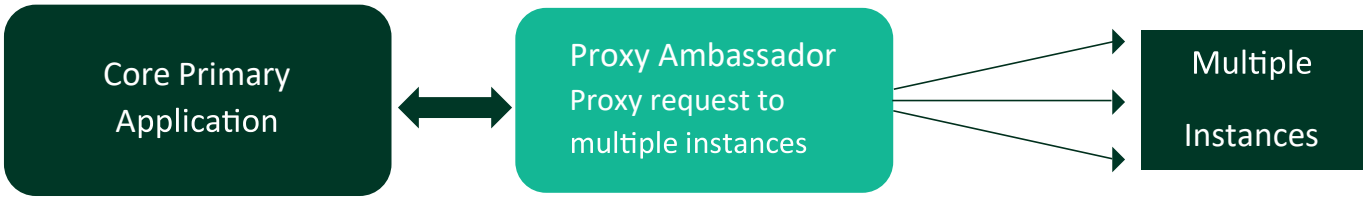
4. Sidecar Design Pattern

Sidecar pattern emphasizes on extending the behavior of a container. With sidecar pattern, we are decoupling out system in different parts. Here, each part has its own task, and each part solves a unique different issue.



5. Ambassador Design Pattern

By selecting ambassador pattern, you are determining proxy for different sections of system. This patter talks about transferring the responsibility to divide the load of network, reattempts, or monitoring checks. A container should have one simple responsibility. For containers, the connection with external system will be an endpoint. It won't know (or care) if what's out there is a set of servers or just one server. This patter is best suitable for microservice based architecture.



6. Adapter Design Pattern

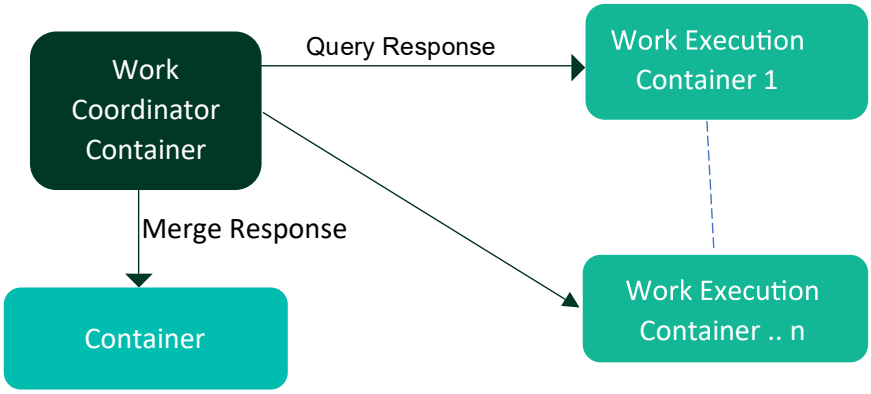
Adapter pattern manages communication between containers. It defines a standard for communicating with a definite set of contracts manages to make request in same way always. This also helps to predict the response, as its always in same format. With this structure, you can replace an existing consumer or client, without any prior notice since the response is always the same, only the implementation is different. We can also reuse this configuration anywhere without worrying about managing other application logs.

7. Leader election Design Pattern

In leader election design, customers that required highly available system is granted with redundancy of containers. Such patterns can be observed in tools such as Elastic search.

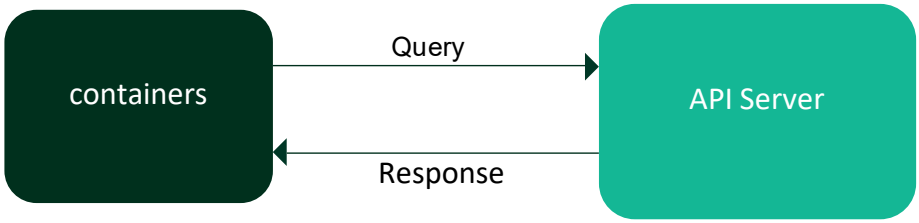
8. Scatter / Gather Design Pattern

This pattern is like work queue pattern as it also talks about splitting bigger task into smaller ones. But has one main difference. Containers are requested to give response back to the consumer immediately. Thus, we need not launch series of tasks and distract from the required response. This pattern overall combines the response to give one response. Best example of this pattern is MapReduce algorithm.



9. Self-Awareness Design Pattern

This pattern manages itself by introspecting and getting metadata about itself and its environment while its executing. Best scenario of this implementation is cases where we need applications metadata and environment variables impacting itself runtime.



Benefits of Containers in Financial Services

Financial Industry and banks were among the first and most enthusiastic supporters of Docker containers.

Goldman Sachs invested $95 million in Docker in 2015.

Bank of America has its enormous 17,500-person development team running thousands of containers.

Why are Containers so preferred in Financial Services?

- Containers take care of the biggest Challenges faced in financial sector i.e.: Security and Compliance with help of its basic features like easy to deploy templates, automation, monitoring, incident reporting and patching.
- It provides an optimal path for services seeking to decrease risk, improve developer efficiency, deliver cost savings through increased agility and time from idea to production.
- An orchestrated container architecture can reduce complexity, offer higher application reliability, and result in more flexible horizontal scalability across the entire back-end infrastructure of financial system.
- Containers and orchestration platforms can help financial institutions get ahead of the competition and deliver disruption to the industry using its strategy.
- Containers promotes microservice architecture which increases data traffic and network, access control and services complications.
- Data Integrity is managed securely in containers. Container systems have incorporated an infrastructure which allow authorized users and admin to sign container images, this helps prevent untrusted or unapproved containers from being deployed.
- To summarize, it helps financial industry to get lower capital costs, greater security, and increased ease of administration.

Best Practices for Operating a Container Platform

Containers have proven to be an ideal platform for microservice-based application architecture. They provide tremendous benefits to the developer community to improve time-to-market agility for application modernization efforts. Financial sector is thus more inclined towards containerization approach.

Below are the guidelines and best practices for using containers for application hosting.

State Management

Immutable and Stateless Kubernetes manages the lifecycle creating and terminating instances based on requirement.

Statelessness is achieved by storing any state of the instance outside the container platform. With a condition of not running within a container, we can configure various type of external storages like Cloud Storage, Persistent Volume, Redis, Cloud SQL, or managed database on premises.

Immutability is achieved in container by not modifying it during its lifetime. By modifying it implies no update or patch or configuration modification. If any update is required in the application code, it is advised to build a new image (preferably with a new version) and deploy it. It is also recommended to externalize the configuration within a container in Secrets and Config Maps.

Configuration Management

Configuration management in Containers are best done using Config Maps. We inject environment variables into the application using this strategy. We can trigger auto deployment of application whenever the configMaps are changed thus makes changes deployed of application faster.

Logging

Use native logging container mechanisms to write your logs to stdout and stderr. They will be automatically shipped, stored, and indexed.

Kubernetes uses Stackdriver logging by default.

You can write your logs in JSON format, which enables to seamlessly add metadat. You can then use the metadata to search through your logs in Stackdriver Logging.

OpenShift is bundled with the open-source tools Elasticsearch, Fluentd, and Kibana (EFK) stack. Fluentd collects the data from stdout and stderr and securely forwards it to Elasticsearch (or Splunk).

Image Management

- Do NOT use privileged containers.

  It is advised not to use privileged containers for the same reason that is for not to run applications as root. In case we need to modify settings on the host, provide the specific capabilities through the security Context to the container. In addition, inside the container avoid executing application as root. As in if a hacker get remote control and finds a vulnerable code which is running as root user, he can then escape the container and have access on host as root.

- Keep one-to-one mapping for between a process/service and container Pod.

  This mapping makes it easy to monitor the container for failure and take corrective action. In addition, it makes the overall system scalable at process or service level.

- Image Registry

  A container registry is a repository—or collection of repositories—used to store and access container images. It can be a public or private registry. Docker hub is most popular public image registry available. It is possible that identical image have different versions but they are always unique by their tags. We can trigger auto deployment of application whenever image is updated on image registry.

- Vulnerability Scanning

  Vulnerability Scanning is an automated process of proactively identifying network, application, and security vulnerabilities. Organization use third-part security provider to perform the scanning. Various tools available for vulnerability scanning are

a. Trivy

   Trivy is a simple and comprehensive scanner for vulnerabilities in container images, file systems, and Git repositories, and configuration issues. It detects vulnerabilities of OS packages and language-specific packages. In addition, it scans Infrastructure as Code (IaC) files such as Terraform, DockerFile and Kubernetes, to detect potential configuration issues that expose your deployments to the risk of attack.

b. Clair

   Clair is an open-source project for the static analysis of vulnerabilities in application containers. Clair scans docker images by doing static analysis, which means it analyzes images without a need to run their docker container.

Monitoring

- Monitor your containers.

  As with logging, monitoring is also an important part of application management.

- Expose and monitor health

  K8s consist of two types of health checks: liveness and readiness probes.

  Liveness probe – If the application on the containers is running successfully and all its required dependencies are met, application should expose a HTTP endpoint. This endpoint should return "200 OK" response

  Readiness probe – If the containers is healthy, initialization is successful, valid request are met, application should expose a different HTTP endpoint. This endpoint should also return "200 OK" response.

  Kubernetes will then start sending traffic to containers.

Secret Management

- Sensitive Data should never be stored in a Docker build file or in Docker images.

  Sensitive data, such as username, password, and token string need to be handled carefully and should always be passed on runtime as an environment variable when the Docker image is run. All such data must be stored in a Secrets store. Secret store act as a secured gateway to store data. We can store data on Kubernetes secret as well, however this data is visible to Admin, thus external storage is advised wherever we want to store secret data which should not be visible to anyone. Data best store using secret store are passwords, confidential data, credentials, database connection strings or access keys to running pods.

Security

Kubernetes Namespaces Are Not Fully Secured. The usage of namespaces in Kubernetes significantly simplifies the management of a Kubernetes cluster. However, managing multiple microservices on the same cluster comes with a security cost when not planned correctly. By applying security and segregation with Namespace and Cluster Role, the required security can be achieved. An RBAC Role or Cluster Role contains rules that represent a set of permissions. Permissions are additive in nature. When a role is created, its always sets the permissions within the namespace.

Microservice Implementation

Microservices architectures are inherently distributed. Building Microservices always bring in the most challenging problems, such as resilient service invocation, distributed transactions, on-demand scaling, and exactly once processing of messages.

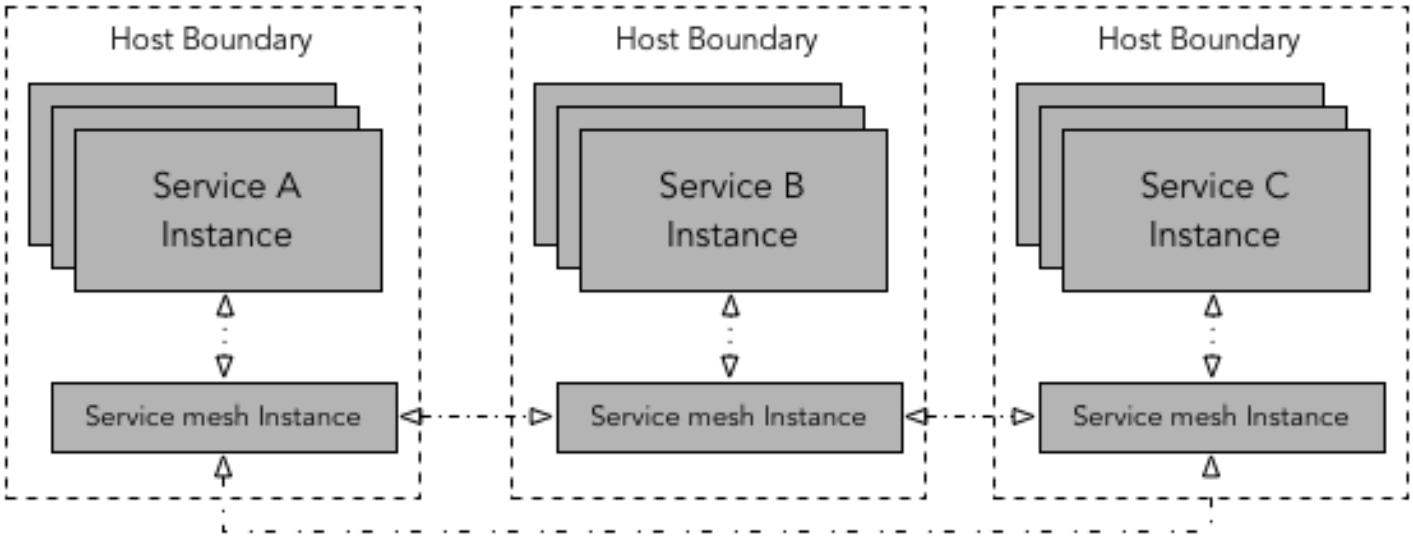
So, developers remain searching for a portable runtime to build distributed, scalable, and event driven Microservices.

- Use Service Mesh Architecture alongside Sidecar pattern

  A Service mesh is specific infrastructure layer that manages service-to-service communication over the network. This structure enables different parts of the application to communicate. It helps you to observe, secure and connect microservices. It helps implement cross-cutting concerns like Externalize configuration, Distributed tracing, Logging, Metrics, Health checks.

  A service mesh is often implemented using Sidecar container architecture pattern. As explained above, sidecar pattern implies expanding the nature of container. With sidecar pattern, we can decouple our system in different parts. Here, each part has its own responsibilities, and each part solves a different problem.

In this type of architecture, requests are usually routed between microservices using proxies in their infrastructure layer. Thus, individual proxies that define a service mesh are signifying "sidecars," as they are running alongside each service, and not within them.



Dapr

Dapr is using Kubernetes as the primary hosting environment to run production applications, though Dapr end users aren't tied to using Kubernetes. An open-source project to make it easier to build microservices.

Istio

Istio is an open source service mesh that layers transparently onto existing distributed applications. It addresses the challenges developers and operators face with a distributed or microservices architecture. Istio is the path to load balancing, service-to-service authentication, and monitoring – with few or no service code changes.

LinkerD

It is an open-source service mesh designed to be deployed into a variety of container schedulers and frameworks such as Kubernetes. It results in running services relaxed and protected by providing runtime debug facility, reliability, vulnerability, observability, and security, with no requirement of any changes in your code.

Advantages and Limitations of Container Platforms

Containers are efficient, lightweight, and definite way for applications to exist between environments and run independently. Apart from shared Operating system, everything required to run the application is packaged inside the container object: code, run time, system tools, libraries, and dependencies.

It is also best applied when we need to run multiple instances of a single application.

Advantages:

- Security management using OAuth.
- Managing various deployments across public, private and hybrid cloud environments made easy.
- Cluster management.
- Console is user-friendly.
- Automated deployments possible for different requirements.
- Managing Images, Routes, Services, pods made easy.
- Patching the environment can be controlled easily.

Limitation:

- Configuration management requires more time.
- Costly when many microservice architecture is involved.
- Log management is challenging in case of cluster environment
- Cluster management is complex, which makes learning curve steep
- OS available by provider can only be used.
- Debugging is complex.

Case Study #1: Selection of Container Platform

Background

Selecting best suitable Container platform for containerization of application.

Problem Statement

There are many approaches available once we decide to containerize a project. Selecting the best approach is very important when you migrate.

While proposing containerization to our client, Infosys did a comparative study of various approach against multiple features.

Resolution

Below is the metric created which helped us choose and design the best approach for the client.

This metrics is created using features which were specific to a case study. The metrics talks about three container platform: Plain Vanilla, OpenShift 4.X, VMware Tanzu.

Each platform is evaluated against key features namely Installation. Security, Storage, Resiliency, Logging, Monitoring, Ease to use and manage, etc. They are then categorized as Very High, High, Medium, Low depending on its complexity.

Features	Option 1 Docker + Kubernetes (Traditional way)	Option 2 RedHat OpenShift Container Platform 4.X	Option 3 VMware Tanzu Enterprise Container Platform
Installation	Complex Procedure	Easy to Install	Very Easy to Install
Strength of Cluster	Medium	Strong	Strong
Security	Low	Very High	High
Container Orchestration	Medium	High	Very High
Storage	Low	High	Medium
Container lifecycle management	Medium	High	High
Container Runtime	Medium	High	Medium
Logging and Monitoring	Medium	Medium	High
Resiliency	Medium	High	High
Build and Deploy	No	High	Medium
Container Runtime	Low	High	Medium
Ease of Use	Low	Medium	High
Reusability of existing VMs	Low	Low	High
Ease of Integration with testing framework	Low	Medium	High
Ease of Management	Low	Medium	High
Ease of training and skill gap	Low	Medium	High
Type	Container-as-a-Service	Platform-as-a-Service	Platform-as-a-Service
Health Checks	Medium	High	High

Findings:

Option 1: Plain Vanilla

This platform is most complex in implementation and lacks most on Security, Storage and Easy

Option 2: OpenShift 4.X

This platform has high performance and scores highest on Security, Storage and Easy. OpenShift has Very high Security layer available which enhances the security of the entire cluster environment. OpenShift enables efficient container orchestration, allowing rapid container provisioning, deploying, scaling, and management. The tool enhances the DevOps process by streamlining and automating the container management process.

In case the existing applications are on VMWare, this solution will enforce to deprecate them and rely on OpenShift provided environments only.

Option 3: VMware Tanzu

This platform has high performance and scores moderately high on Security, Storage and Easy. Tanzu is easiest to install. It is supported by various tools like Helm chart for its build and deployment. It creates application templates with baked-in security and compliance guardrails. Build containers with secure components and helps maintain them automatically. Connect and protect your apps in production.

Outcome

Since client had 130 + existing applications which were all using VMWare currently – Tanzu approach is best suitable. With Tanzu approach, we could use existing VMWare environment and enhance to as a Tanzu cluster environment with minimum cost. Tanzu has easiest installation and has good logging, monitoring and security. It enables you to build software more securely and continuously, reducing risk in production.

Case Study#2: Modernization of monolithic application

Background:

Current application is a standard monolithic application. Clients want to modernize it to a Containerization approach and deploy on OpenShift platform.

The Standard architecture pattern consists of following areas:

- A Java 8 Enterprise application (UI and back end).
- A database system.
- A Node.js application (User interface).
- A web server (a load balancer).

Problem statement:

The monolithic application is a legacy application deployed on on-Premises server. With modernization approach, client wants to migrate the entire application to Openshift. We need to identify various stages and areas that will need updating.

Resolution

The deployment team followed below task to containerize the application.

- Created required namespace in OpenShift (version 3.11) and K8s (1.11) on the administrative workstation node.
- Used existing on-Prem DB. Opened firewall for same.
- Build a Docker image for the application component both server and UI.
- Maintain docker image in Nexus repository.
- Build a network which allows direct communication within containers.
- Execute containers for each Docker image that is built.
- Configure Route for appropriately routing traffic using ping integration to the containers.
- Created HashiCorp Vault for storing secure data. Configured application to connect vault security.
- Create Venafi certificate to enable ping setup using JWT token.

Benefit:

Containers require less system resources than traditional or hardware virtual machine environments because they don't include operating system images. Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms.

Post migration, deployment of images is easy and managed by environment itself. Also, deployment of application on various environment like test, pre-Prod and Prod are managed efficiently and accurately. Image repository maintains the application image and can be used easily to deploy anytime anywhere.

Conclusion

Containerizations benefits makes it imperative for any organization looking for edge over market.

Even for large well-equipped cloud enterprise architecture, adding container can maximize its results.

On other side for small business, it gives minimum resource usage and maximum productivity.

With best architecture and services, a complete and secure solution is possible using Containerization approach.

About the Author

Jyoti Sharma
Technology Architect

References

- https://www.cnbc.com/2015/04/14/goldman-sachs-invests-95-million-in-docker.html
- https://www.networkworld.com/article/3013474/cloud-computing/how-goldman-sachs-and-bank-of-america-use-the-cloud-and-containers.html
- https://techbeacon.com/enterprise-it/7-container-design-patterns-you-need-know
- https://kubernetes.io/docs

For more information, contact askus@infosys.com

Infosys®
Navigate your next