Cloud Native Resilient Architectures & Security

By Abhishek Gautam

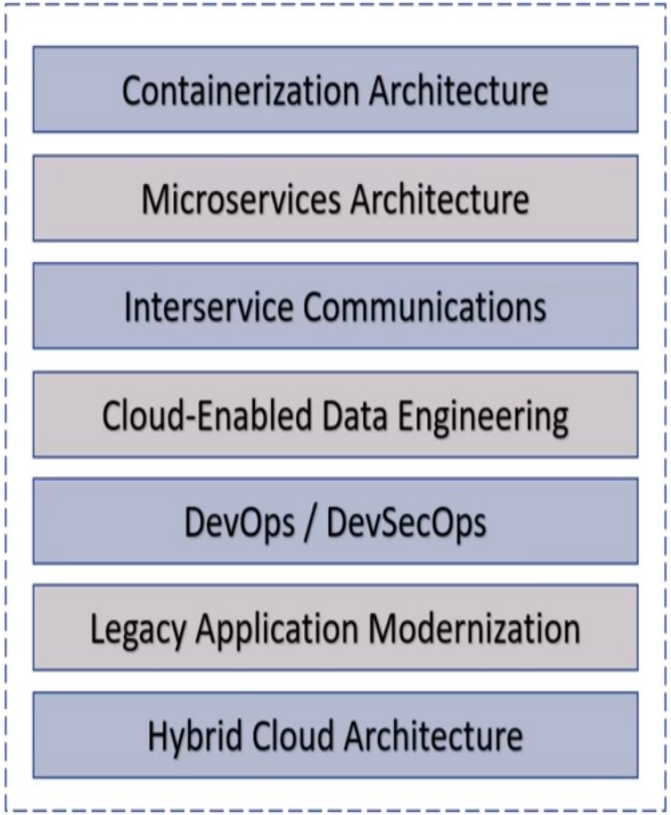
miniOrange

Multiverse Cloud Architect

Over 17 years of experience

Technology Driven

Java Enthusiast


Containerization Architecture
Microservices Architecture
Interservice Communications
Cloud-Enabled Data Engineering
DevOps / DevSecOps
Legacy Application Modernization
Hybrid Cloud Architecture


2023 Industry Trends
Overall Architecture
Architectural Best Practices
Industry Recognized Architectural Guides

Introduction to Cloud-native Applications and Systems

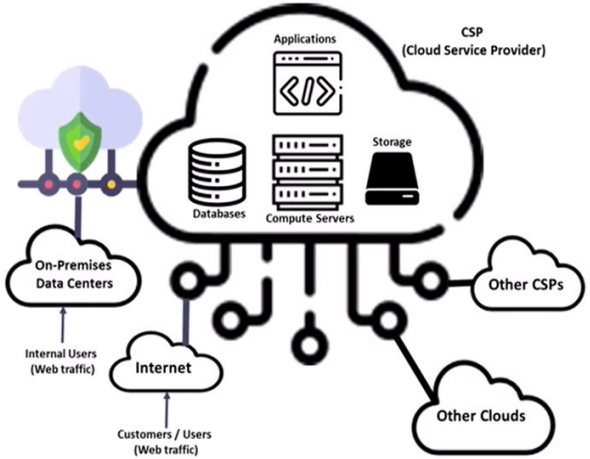
Cloud computing – 2023 Industry and Market Trends

Organizations are finding more innovative ways to help them increasingly become digital and reimagining their business models.

- The shift from public and private to 'hybrid' – McKinsey
- The increase in the types and numbers of smart devices is accelerating the push toward edge and hybrid architectures
- By 2025, 95% of the organizations would have made the leap to cloud-native – Gartner
- Cloud computing innovation is fueling advanced applications (e.g. remote desktop, cognitive assistance)
- Complex cloud computing architectures is accelerating general automation trends as well as AIOPs.
- The trend toward serverless computing is accelerating
- With business growth and expansion, organizations are looking for advanced and innovative scalability solutions.
- Use of containers / Kubernetes is strengthening by the day as organizations are gaining increased confidence – CNCF
- Amazon, Microsoft, and Google (AMG) continue to define the market currently at $180B and growing by 24% (IDC).
- The cloud computing trends are universal across all industries.

Introduction to Cloud-native Applications and Systems

Cloud Architecture Introduction



IT / Cloud / Digital Solution

- Business Logic / Applications
- Database
- Networking
- Technology / Infrastructure

Driven by Business Requirements (Functional Requirements + Non-Functional Requirements)

- Container Service
- AI / ML Service
- Analytics Services
- Big Data Service
- VPN Services
- SD/WAN Services
- Security Services
- Compute Services
- Storage Services

Use multiple cloud services to stich a cloud solution.

Introduction to Cloud-native Applications and Systems

Introduction to Cloud Native Systems and Applications

Cloud-Native Applications / Technologies

"Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil." **(CNCF)**

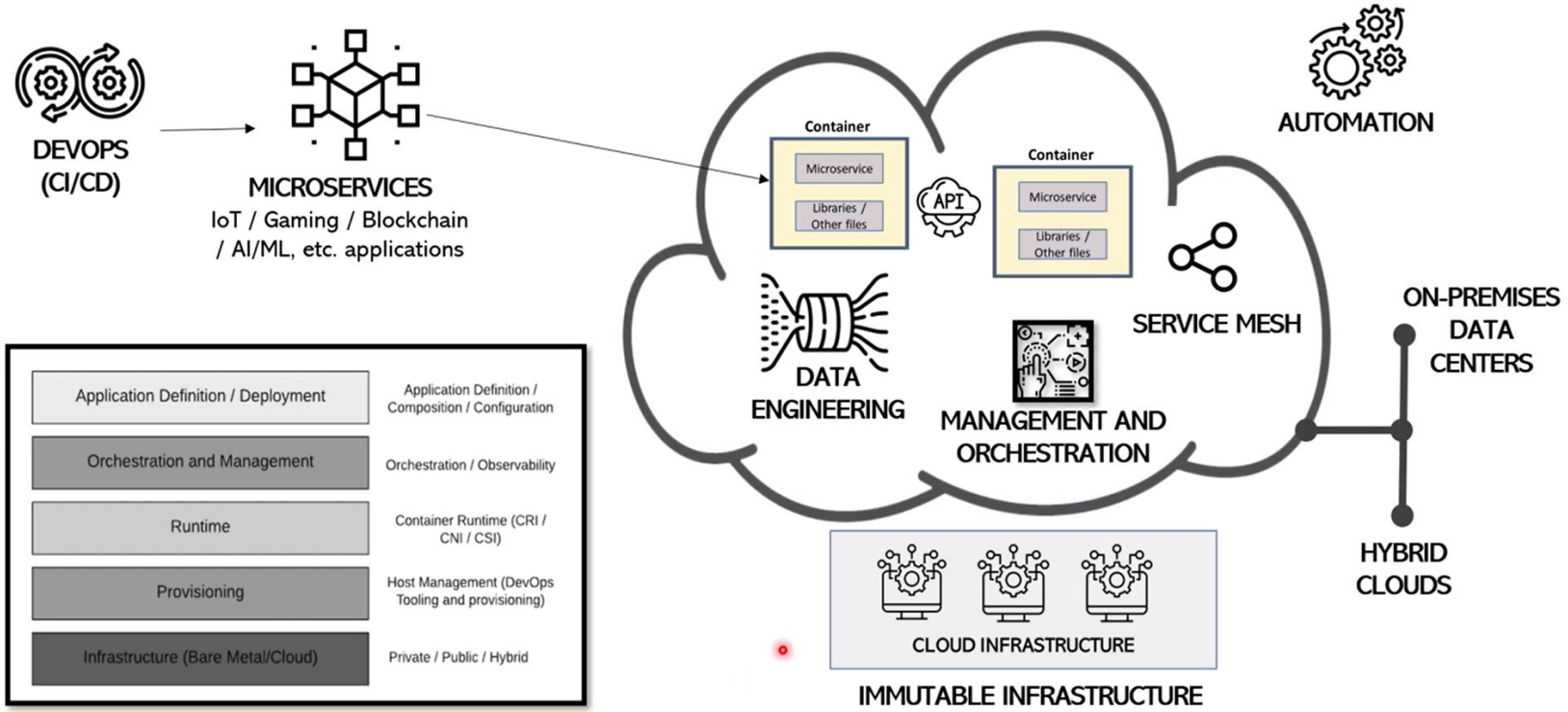
Cloud-Enabled Applications

Legacy applications, which are modified enough to be able to run in a cloud environment.

- Not born on the cloud.
- Not optimized to run on the cloud.
- Don't make full use of the cloud features and strengths.

Introduction to Cloud-native Applications and Systems

Key Building Blocks of Cloud Native Applications and Systems



Cloud native reference architecture by the CNCF

Containerization Architecture and Best Practices

Containerization – 2023 Industry and Market Trends

Containerization has provided organizations with the means to develop once and deploy on multiple platforms.

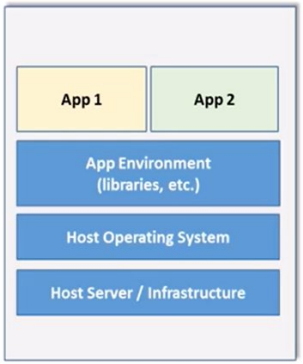
- Containers and microservices are the foundation for building cloud-native applications.
- Large scale migration is occurring toward containers (including legacy and cloud-native) – Docker
- Almost half of all container implementations are in the hyperscalers data centers – Omdia Research
- Most state of the art applications such as AI, ML, etc. are more effective when implemented using containers.
- More full-fledged Kubernetes implementations are increasing (compared to smaller scale implementations)
- Containers along with microservices enable efficient software delivery pipelines
- The use of serverless in the context of containers is accelerating. – DataDog
- Adoption of containers is also accelerating the push toward hybrid and multi-cloud implementations
- The use of container registries hosted at hyperscalers (e.g. Azure, AWS, and Google) is increasing.
- Numerous cloud products and solutions are built on containers (e.g. MySQL, MangoDB, Kafka, etc.)

Almost 65% of worldwide organizations have adopted the use of containers and Kubernetes (as of early 2023)

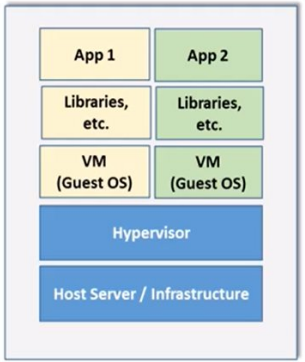
Containerization Architecture and Best Practices

Comparing Compute Instances for Running Applications

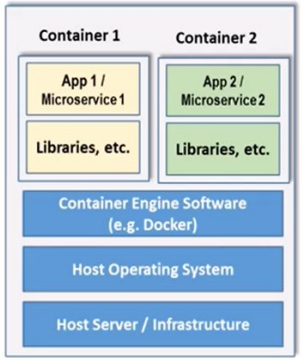
Most common configurations for running applications



Bare Metal Configuration



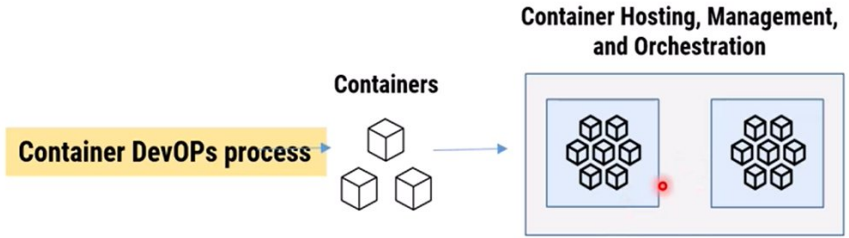
VM Instances



Containers and Microservices

About Containers

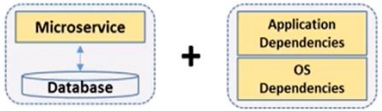
- Small software packages
- Contain all software, dependencies, libraries, etc.
- Enable the building of cloud-native applications
- Isolated from each other
- Application and its dependencies are packaged together
- An application environment with lots of containers can be managed through a container orchestration and management tool.
- Support both Windows and Linux workloads



Containerization Architecture and Best Practices

Key Elements in the Building and Running of Containers

Basic terminology related to containers



Dockerfile
(Contains all commands needed to create a Docker Image)

Container Image
(File with instructions on creating a container in a runtime environment.

Run Command)

Container
(Runtime realization of a container image)

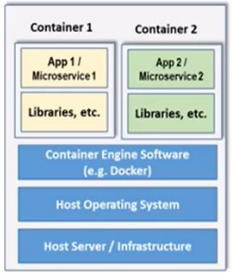
Container Manifest File
(contains metadata information about a set of files to be deployed in production.)

Container Repository
(Collection of artifacts related to containers and the use of containers)

Container Engine
(Container Runtime Environments. Converts a container image into a container.)

Container Orchestrators
(Manage and govern containers. Kubernetes is an example)

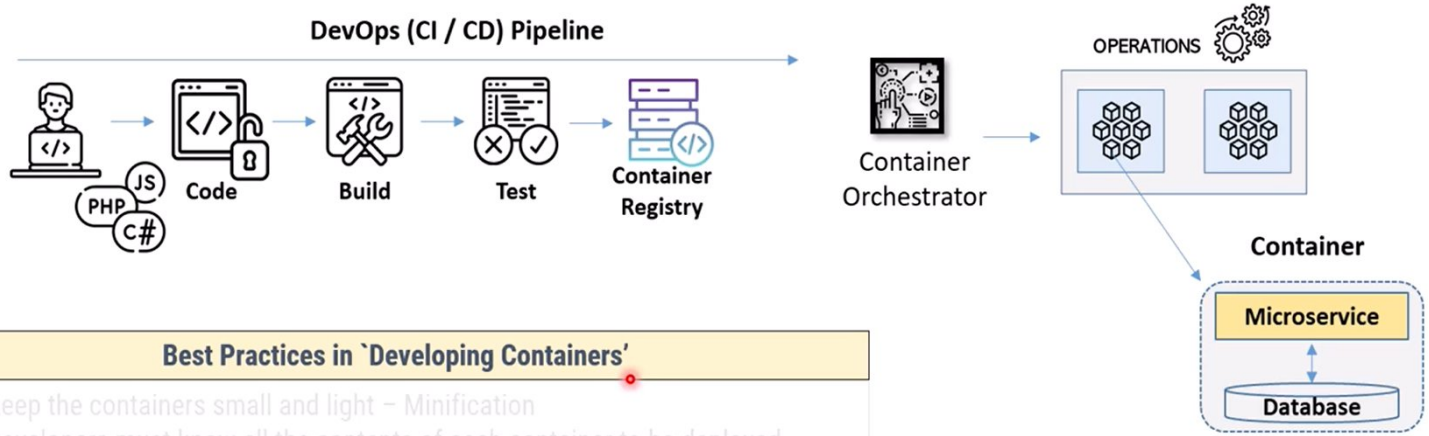
Container Host
(Runs containerized processes)



Containers and Microservices

Containerization Architecture and Best Practices

Best Practices Related to Developing Containers

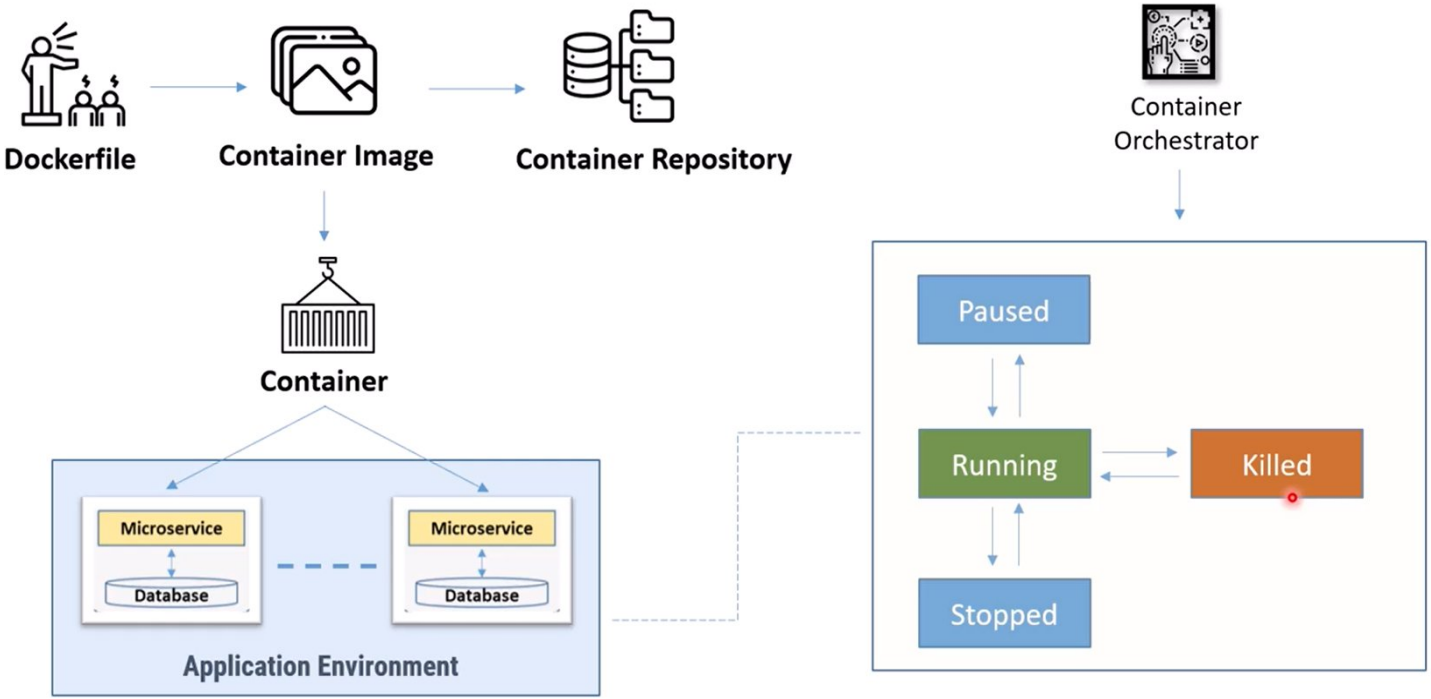


Best Practices in `Developing Containers'

- Keep the containers small and light – Minification
- Developers must know all the contents of each container to be deployed
- Build Standards for Development
- No hardcoding of configuration or other data.
- One application / process in each container.
- Introduce logging to track full application behavior.
- Be careful when using public images
- Automate the build of your images and propagate updates.
- Scan containers for any vulnerabilities. Use Software Composition Analysis (SCA) and Static Application Security Test (SAST) tools.
- Be careful when using public images.

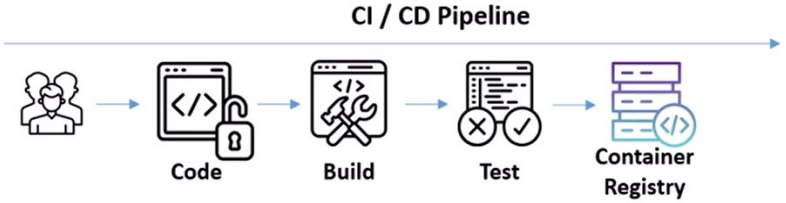
Containerization Architecture and Best Practices

An Overview of the Containerization Lifecycle

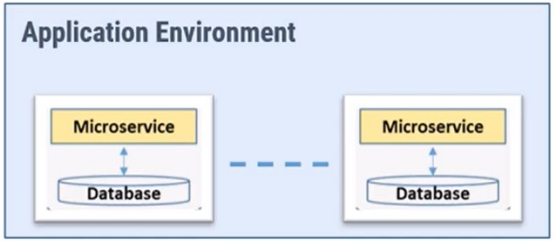


Containerization Architecture and Best Practices

Best Practices Related to Operating Containers (General)



Container Orchestrator



Applications

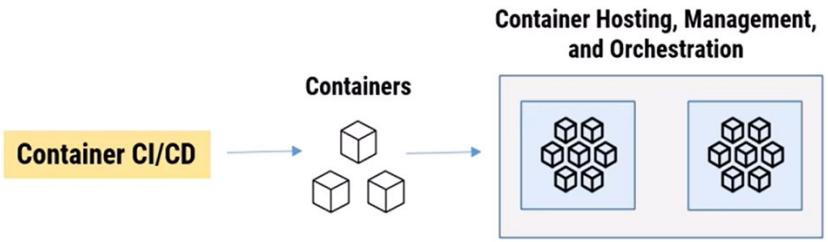
Users

Best Practices in Operating Containers

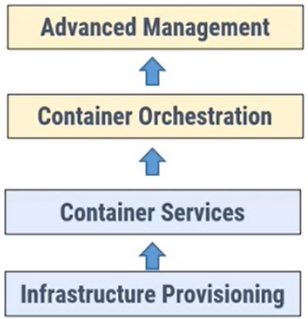
- Ensure to keep containers small
- Track health of all the containers
- Ensure logs are accessible to external applications
- Define metrics for the application to help operations staff to monitor the health of the application
- Applications in the container shouldn't be run as the 'root' user
- Define compute resource limits for each container (CPU, RAM, etc.)
- Use persistent data stores

Containerization Architecture and Best Practices

Factors Driving the Choice of Container Management and Orchestration Systems



Container Management



- Container deployment
- Communications between containers
- Cluster management (configuration and monitoring)
- Load balancing
- Security (including policy management)
- Automatic Rollbacks and Node Recovery
- Data sharing with other containers
- Logging and monitoring
- Multiple-Cloud orchestration
- Etc.

Options for Orchestration Tools

- ✓ No Orchestration
- ✓ Container-as-a-Service (CaaS)
  - Amazon ECS
  - Amazon Fargate (serverless – use with Amazon ECS)
  - Google Cloud Run
- ✓ Managed Kubernetes service
  - Amazon Elastic Kubernetes Service (EKS)
  - Google Kubernetes Engine (GKE)
  - Azure Kubernetes Service (AKS)
- ✓ Self Managed Orchestration
- ✓ Other User friendly Orchestration Tools
  - Nomad
  - Cycle.io

Factors Impacting Selection of Orchestration Tools

- Extent of orchestration (Simple / Complex) – Features
- Control and management of the environment
- Administrative load
- Cost
- Ease of use
- Etc.

Containerization Architecture and Best Practices

Overview of Kubernetes Orchestration Architecture

Pods | Nodes | Clusters

Container (C)

Microservice

Libraries / Other files

Pod

Container 1

Container 2

Share storage and network resources

Node 1 (Worker Node)

POD 1: C C

POD 2: C C

- Bare metal server
- VMs
- Etc.

- Acts as the front-end to the cluster
- Schedule various activities through the worker nodes
- Provides the overall governance of the cluster
- Etc.

Kubernetes Cluster

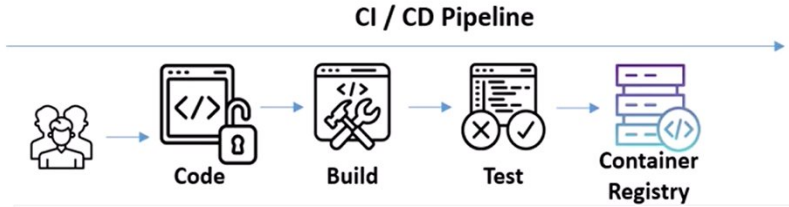
Master Node

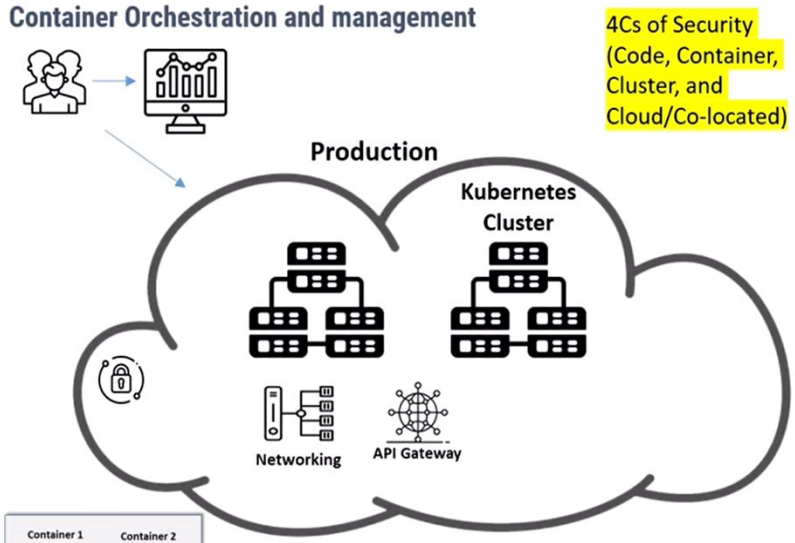
Worker Node | Worker Node

Control Plane

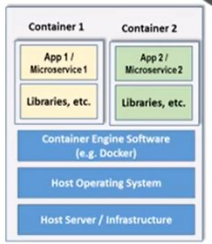
Containerization Architecture and Best Practices

Best Practices for Securing Containers





4Cs of Security (Code, Container, Cluster, and Cloud/Co-located)

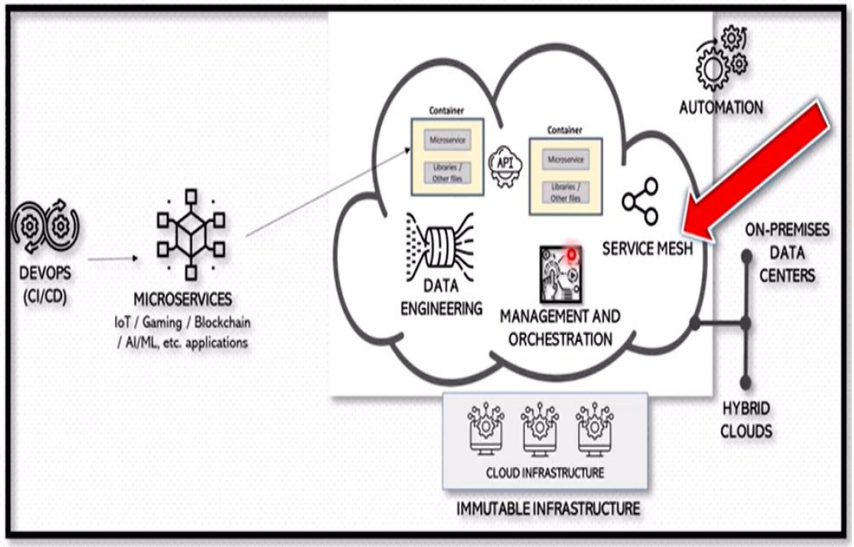


Containers and Microservices

Inherent Risks in Containers

- Containers are light
- Lots of containers
- Distributed environment
- Sharing and reusability

1. Interservice Communications – 2023 Industry and Market Trends
2. Introduction to Cloud Interservice Communications
3. API and API Gateways
4. gRPC and Interservice Communications
5. Interservice Communications using Message Queues
6. Service Mesh Enabled Interservice Communications



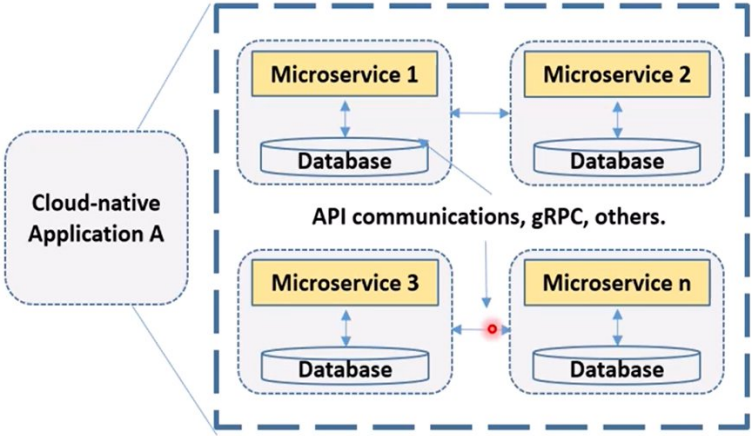
Microservices Architecture and Best Practices

Best Practices for Microservices Development

- World shifting to cloud-native applications based on microservices
- CNCF encouraging the adoption of microservices
- Most organizations have microservices based projects (Gartner survey).

Best Practices Related to Microservices Development

- Containerize microservices
- Limit each microservice to have one responsibility
- Separate database for each microservice
- Design microservices to be independently scalable
- Reduce chatty communications within microservices
- Use Asynchronous communications for backend microservices
- Prevent direct client-to-microservice communications
- Use multiple gateways to route microservices requests based on business application domain
- Report health statuses
- Logging and diagnostics
- Maintain Application cohesiveness through Orchestrators
- Secure microservices
- Structure development teams around microservices

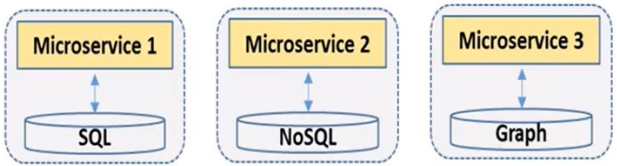


Microservices Architecture and Best Practices

Database-Related Patterns when working with Microservices

Database Types

- Structured and relational databases (e.g. Oracle, MySQL, and DB2)
- NoSQL databases
  - Graph Databases (e.g. AllegroGraph, InfoGrid)
  - Key Value Databases (Cassandra, Redis, and DynamoDB.)
  - Others



Database Issues in Cloud-native Applications

- Database ownership (Keep microservices loosely coupled)
- An application made of multiple microservices may own various types of databases (Polyglot persistence).
- Implementation methods for distributed transactions
  - Saga pattern
  - API Composition Pattern
  - CQRS – Command Query Responsibility Segregation
  - "Shared-database-per-service pattern" (Anti-pattern)
- Use of Multi-Model databases

Summary

1. Select the right of databases for your microservices (and supporting use cases)
2. Choose the right data patterns
3. Plan approaches to work with multiple data models.

Microservices Architecture and Best Practices

Twelve-Factor Methodology / Approach

Set of best practices related to the design and build of services based applications. Recommended by AWS and others.

1. Codebase	• Codebase should not be repeated across apps • Different versions of the codebase can be deployed in multiple environments (dev/prod).
2. Dependencies	• Each app must explicitly declare all its dependencies and must be packaged part of the application deployment package.
3. Configurations	• An app's environment configuration is not to be stored in the code but should be in separate files.
4. Backing Services	• Each backing service should be treated uniformly and swapping one resource for another should be transparent to the application.
5. Build / Release / Run	• The environment should maintain a strict separation between the build, release, and run stages.
6. Processes	• Each process should be stateless. Therefore, sticky sessions and stateful services are a violation of the twelve-factor principle.
7. Port Binding	• Each process should expose itself to the outside world using port numbers and not domain names.
8. Concurrency	• Organize the application's processes on a cloud network in a manner that can allow only the required parts of the application to be scaled.
9. Disposability	• Each process can be started/shutdown rapidly and gracefully (minimize startup and shutdown times and leave a clean environment.)
10. Dev/Prod Parity	• Minimize the gaps that exist between dev and production environments. Gaps are classified as tools, personnel, and time related.
11. Logging	• Each twelve-factor app should not concern itself to writing and processing log events. This ensures that the logging process works independently.
12. Admin Processes	• Separate one-off administration processes and encourage packaging and shipping of these administrative functions with the app.

Cloud Interservice Communications Technologies

Interservice Communications – 2023 Industry and Market Trends

Interservice Communications refers to technologies used in the communications of microservices.

- The most common used communications protocols in the area of cloud-native include APIs, service mesh, gRPC, and asynchronous communications.
- APIs and API gateways are one of the most used communications frameworks for cloud-native applications.
- The Cloud API Market has been growing at an annual rate of 20%.
- The use of service mesh is constantly increasing.
- APIs, service mesh, Asynchronous communications, etc. all are playing unique roles to bridge the communications gap

Cloud Interservice Communications Technologies

Introduction to Cloud Interservice Communications

Synchronous Communications

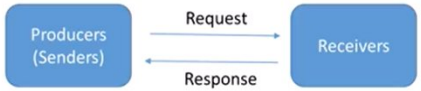
- Sender waits for a response from the receiver
- Most popular communication protocol is HTTP/HTTPS
- APIs are used when external entities try to communicate with microservices
- gRPC is the most popular RPC method used for backend microservices communications

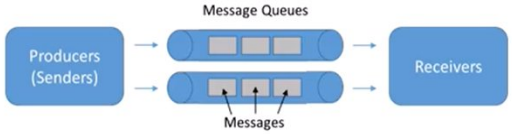
Asynchronous Communications

- Sender doesn't wait for a response from the receiver.
- Can be used without knowing the communicating entities.
- Non real-time communications
- AMQP (Advanced Message Queuing Protocol) is the most common protocol
- Message queues and brokers are used. (e.g. Kafka and RabbitMQ queue)

Selection Factors

- Communications latency.
- Real-time / Non- communications.
- External access.
- Monitoring and Observability.
- Security (authentication).
- Security (clear text / binary communications)
- Communicating entities.
- Traffic volume.
- Number of microservices.
- Communications message formats.





APIs **gRPC** **AMQP**

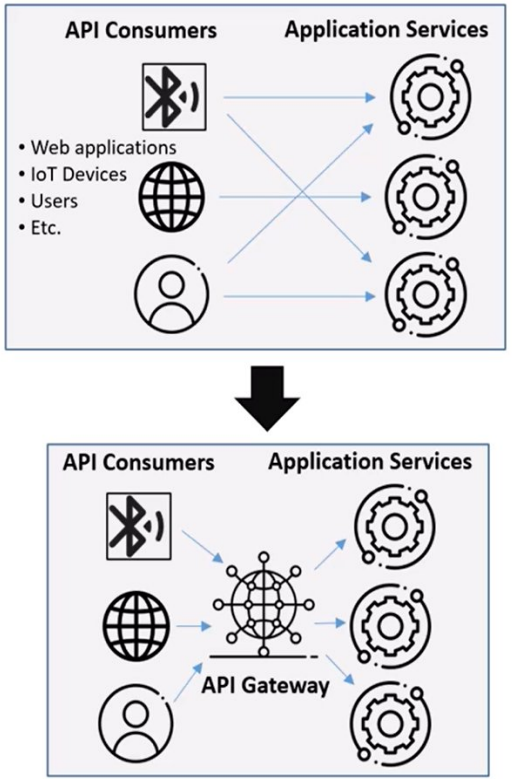
Service Mesh

Cloud Interservice Communications Technologies

API and API Gateways

API Basics

- One of the most common communications methods.
- Use APIs to request data directly from each of the microservices.
- APIs are mostly used between external clients and servers.
- Use an API gateway between the clients and microservices to control data requests.
- An API gateway authenticates clients before allowing data requests.
- Most commonly used API technology is REST (Representational State Transfer).



API related products and services

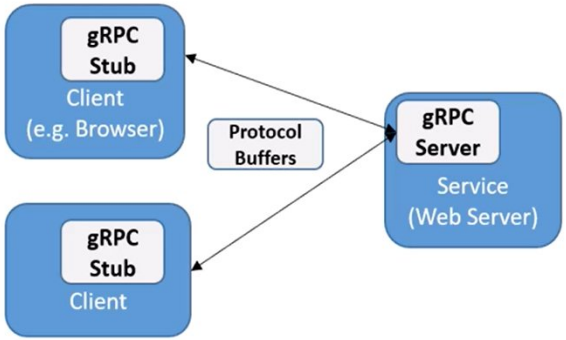
- Google Apigee
- Mulesoft
- Amplify (Axway)
- Software AG
- Kong
- AWS / Azure API services.

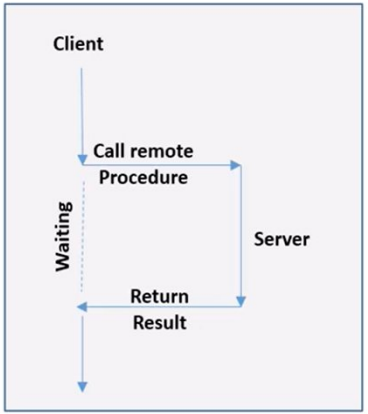
Cloud Interservice Communications Technologies

gRPC and Interservice Communications

Basics of gRPC

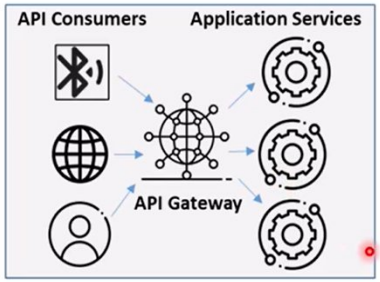
- Relatively new form of communications. Uses HTTP/2 protocol.
- An active CNCF project.
- Being used by Google, Netflix, and many other organizations for their large cloud-native applications and systems.
- Uses binary format. Much faster.
- Used for internal / backend microservices communications.
- Provides security, performance, and scalability.





When to use gRPC

- Low latency use cases
- Extensive bidirectional real-time streaming.
- Connect services developed in multiple languages.
- Used for backend communications.
- Streaming and IoT applications
- Tools support multiple languages.
- Support for health checking.



Cloud Interservice Communications Technologies

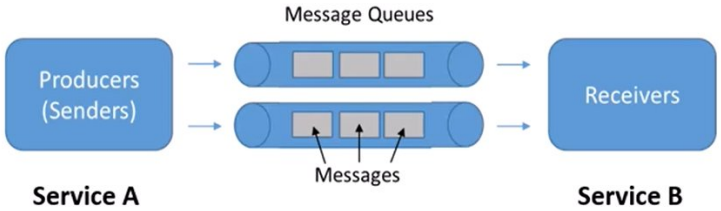
Interservice Communications using Message Queues

Basics of Asynchronous Communications

- Message queues are a form of asynchronous communications.
- AMQP is one of the most widely used messaging protocols used for this type of communications.
- Implemented through the use of message brokers such as Kafka, RabbitMQ, and others.
- Implemented using two methods. (1) One-to-one (queue) implementation. (2) One-to-many (topic) method.
- Facilitates decoupled communications.

When to use Asynchronous Communications

- Real-time response is not needed.
- Multiple services wait for the same event.
- Sending messages to a lot of receivers.
- Streaming data from multiple senders.
- High Traffic Situations
- Numerous retries are needed
- Communications between numerous entities (e.g. microservices).



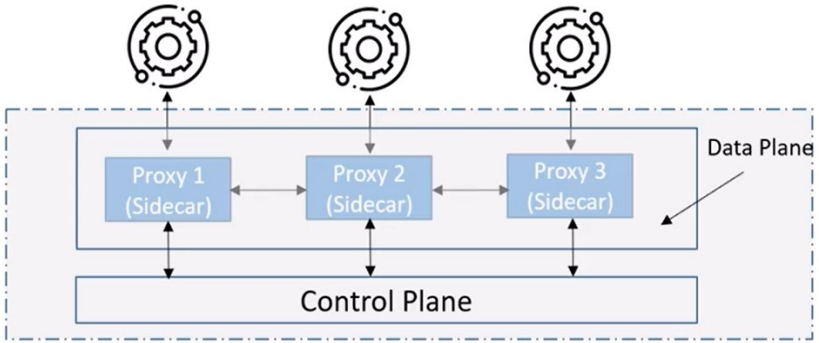
Messaging related products and services

- Amazon SNS
- Amazon SQS
- Amazon MQ
- Kafka
- RabbitMQ

Cloud Interservice Communications Technologies

Service Mesh Enabled Interservice Communications

Basics of Service Mesh



Messaging related products and services

- Linkerd
- Envoy
- AWS App Mesh
- Istio
- Etc.

Modernizing Legacy Applications for the Cloud

Legacy Migration – 2023 Industry and Market Trends

The Legacy application migration market is discussed in the context of application modernization.

- The worldwide market is expected to grow from $20B in 2022 to $50B by 2028.
- Most of the application modernization efforts are concentrated on transitioning to cloud-based applications.
- The major driver for app modernization is the inability of legacy applications to handle new business demands.
- The top players in the application modernization market include IBM, Atos, Cognizant, and Accenture.
- The top app modernization projects are focused on legacy platforms based on COBOL, ADA, PL/1, RPG, and Assembler languages and technologies.

Source: Valuates Report – See 'Additional Learning Resources'

DevOps and DevSecOps Practices

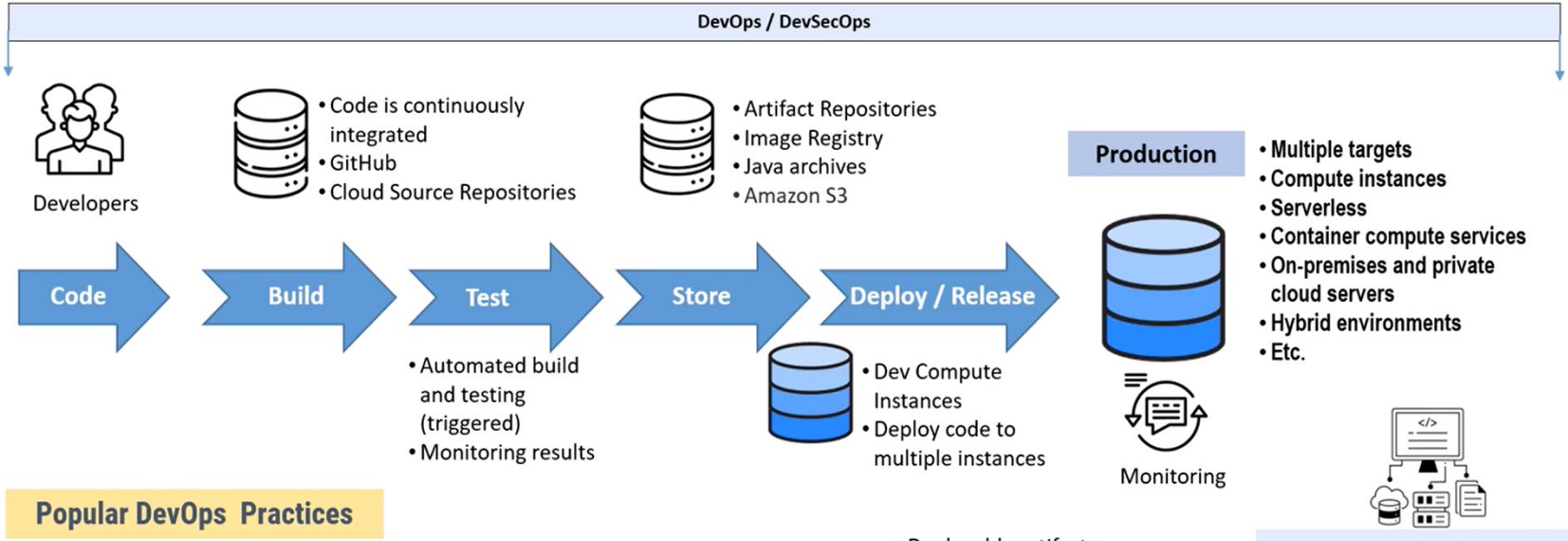
DevOps / DevSecOps – 2023 Industry and Market Trends

DevOps – a set of tools and practices that integrate software delivery, deployment, and operations, is meant to increase the efficiency of the overall SDLC and operations processes supporting applications.

- Instituting of DevSecOps practices
- Integration with other IT processes
- Instituting GitOps practices to automate Infrastructure deployments
- Extending DevOps capabilities to include support of Low Code Applications
- Integration with Kubernetes
- Continued focus on improving 'organizational culture'
- CI/CD pipelines are being automated to reap full benefits of DevOps
- A continuous increase in Dev and Operations Collaboration
- The use of AI in DevOps
- Further increase in the use of Infrastructure as Code (IaC) practices

DevOps and DevSecOps Practices

CI/CD Pipelines in Cloud-native Applications



Popular DevOps Practices

1. Continuous Integration (CI)
2. Continuous Delivery (CD) (Automated except Deployment)
3. Continuous Deployment (Fully Automated including Deployment)

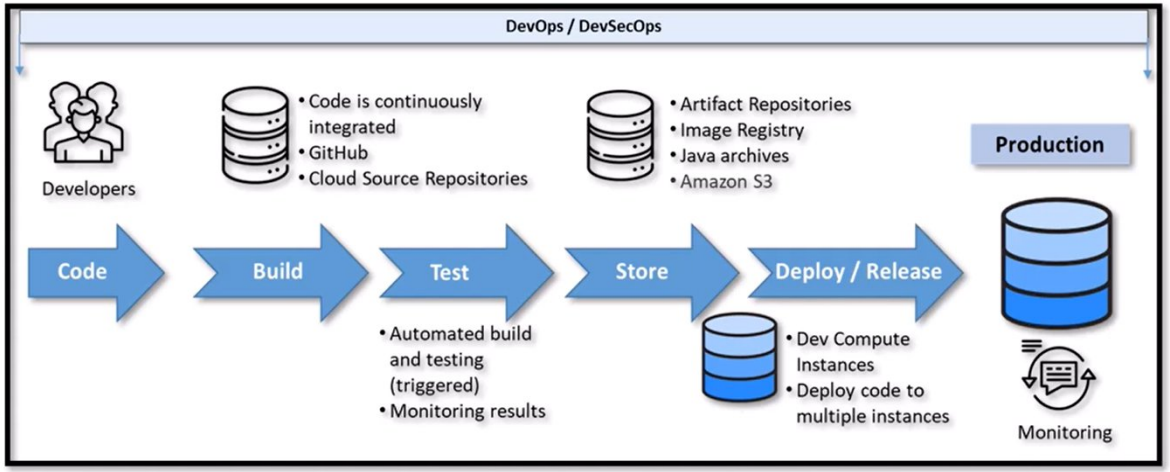
• Deployable artifacts (infrastructure and apps)

Staging

4. Infrastructure As Code (IaC)

DevOps and DevSecOps Practices

CI/CD Tools and Services



Typical DevOps Functions / Tools

- Maintain source code repositories (e.g. Git, TFS)
- Building code (e.g. Jenkins, Artifactory)
- Configuration management (e.g. Chef, Puppet)
- Provisioning of infrastructure (e.g. AWS)
- Testing and automation (e.g. Kobiton, Bamboo)
- Version control (e.g. TFS, Plutora)
- Pipeline orchestration (e.g. Digital.ai's Agility
- Etc.

- AWS CodeCommit
- Azure Repos

- AWS CodeBuild
- Google Cloud Build

- Google Container Registry
- Azure Container Registry

- AWS CodeDeploy
- HashiCorp Terraform,

Fully managed services (AWS CodePipeline, Azure Pipelines)

Best Practices in Hybrid Cloud Implementation

Hybrid Cloud – 2023 Industry and Market Trends

Hybrid cloud computing combines using the on-premises environment along with the use of public cloud providers and their services.

- 82% of organizations have adopted hybrid cloud
- 92% of organizations have adopted multi-clouds
- 70% are using hybrid clouds for application development
- 50% are using hybrid clouds for cloud bursting
- Cloud-native is accelerating hybrid cloud adoption
- Hybrid is necessitating the use of AIOPs
- Growth of specialized cloud is driving the hybrid trend
- Use of Integrated cloud solutions is increasing
- Popularity of Hybrid / Multi-Cluster Kubernetes solutions
- Use of edge clouds as part of hybrid solutions is increasing

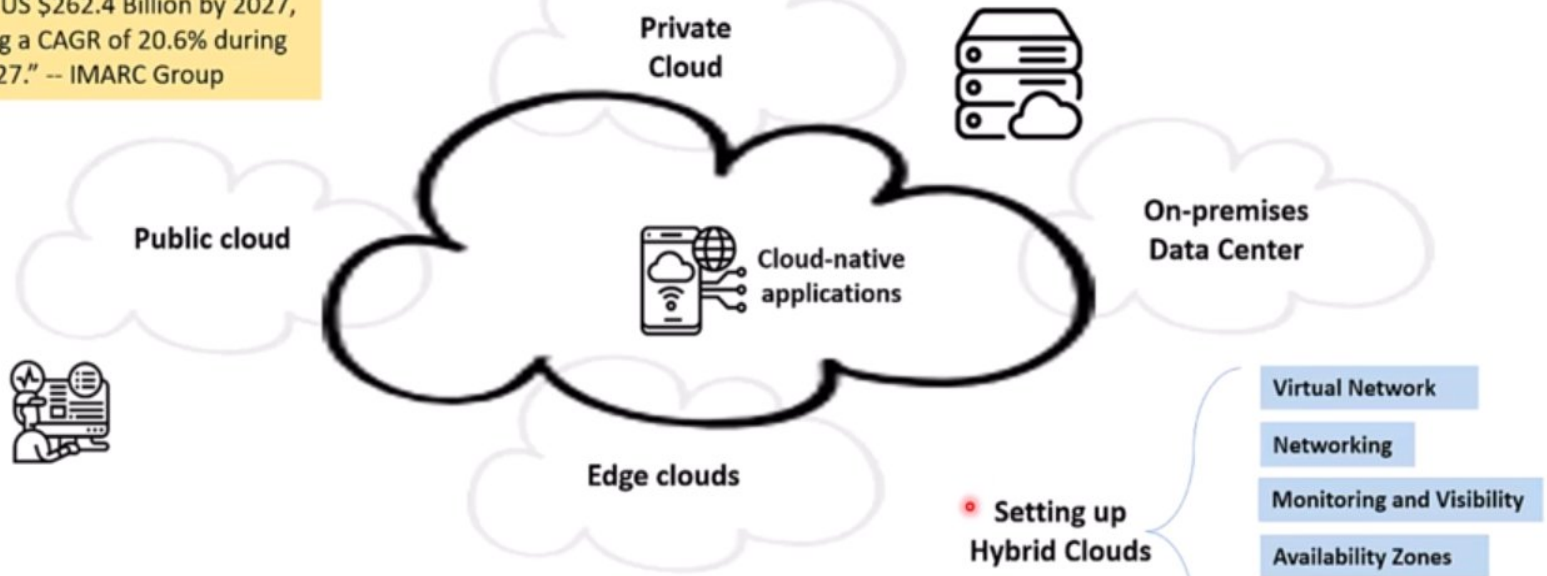
Source: Cisco Hybrid Clouds Report / McKinsey Report

Best Practices in Hybrid Cloud Implementation

Hybrid Cloud High-level Architecture

A Hybrid cloud architecture (as the name implies) uses a mix of both on-premises and cloud provider infrastructure.

"Hybrid Cloud market is expected to reach US $262.4 Billion by 2027, exhibiting a CAGR of 20.6% during 2021-2027." -- IMARC Group



- Setting up Hybrid Clouds
  - Virtual Network
  - Networking
  - Monitoring and Visibility
  - Availability Zones
  - Security
  - SLAs

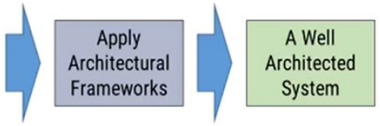
Industry Cloud Architectural Guidance and Best Practices

Benefits of Applying Cloud Architecture Frameworks and Guidelines

Review architecture guidelines and best practices provided by the major cloud service providers.

Cloud Architecture Projects and Examples

- Migrating legacy applications to the cloud.
- Building cloud-native applications from scratch.
- Building data pipelines.
- Building ML workloads.
- Etc.



Architectural Frameworks and Guidelines

AWS Well-Architected Framework and Best Practice Guidance

Microsoft Azure Well-Architected Framework and Best Practice Guidance

CNCF Recommended Journey to Cloud-native

Benefits of Architectural Frameworks

- Help architects in making various architectural decisions.
- Identify best architectural practices.
- Measure the quality of a system architecture against industry guidelines.
- Provides a set of foundational questions that can be used to measure the quality of a cloud architecture.
- Ensure that systems measure up in quality and best practices against the modern cloud-based systems.
- Assess the quality of an overall system and application architecture
- Identify any high-risk issues and mitigate them.

Scan for Feedback

Thank You