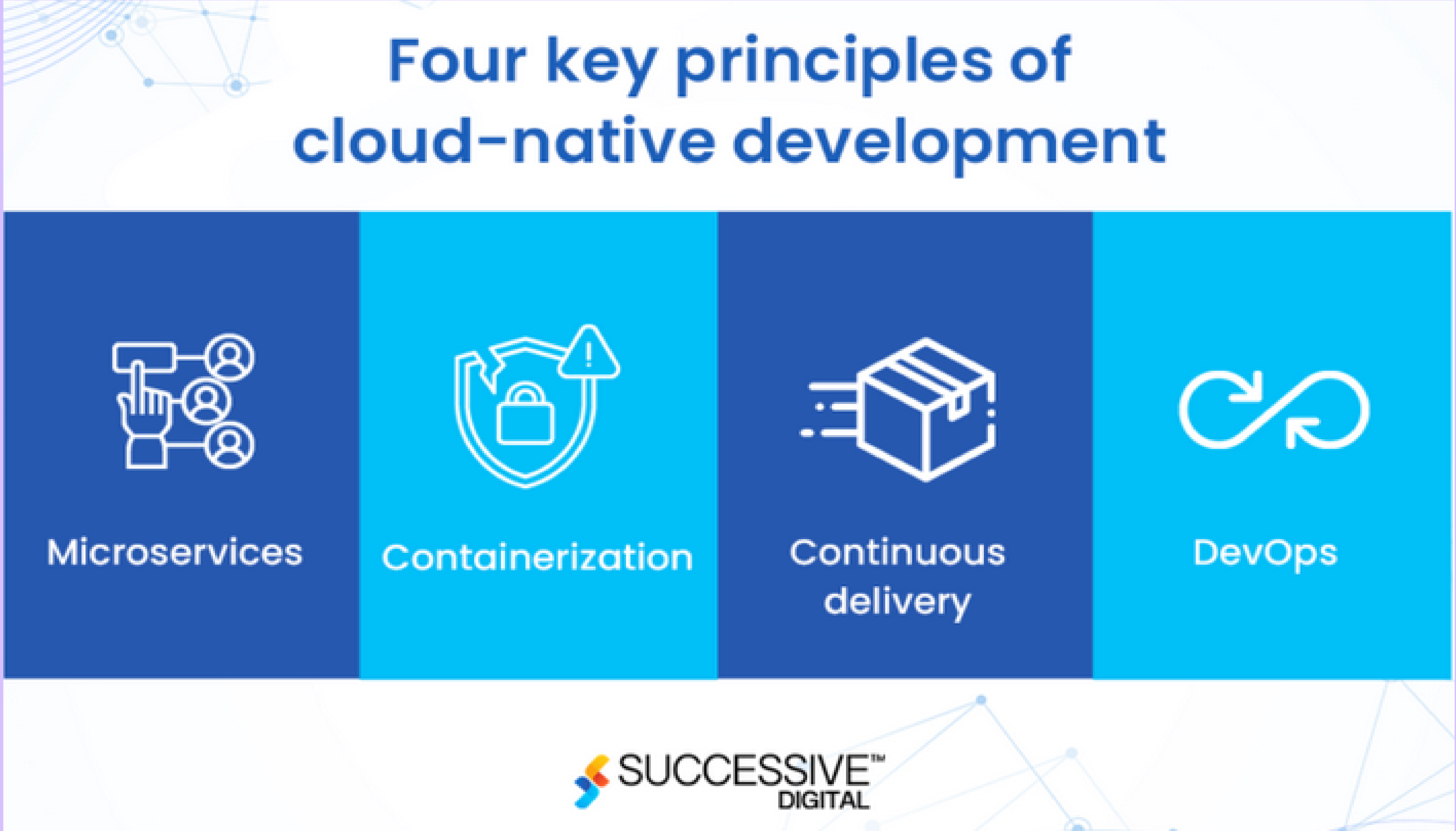
22AIE305: CLOUD COMPUTING



Four key principles of cloud-native development

- Microservices
- Containerization
- Continuous delivery
- DevOps

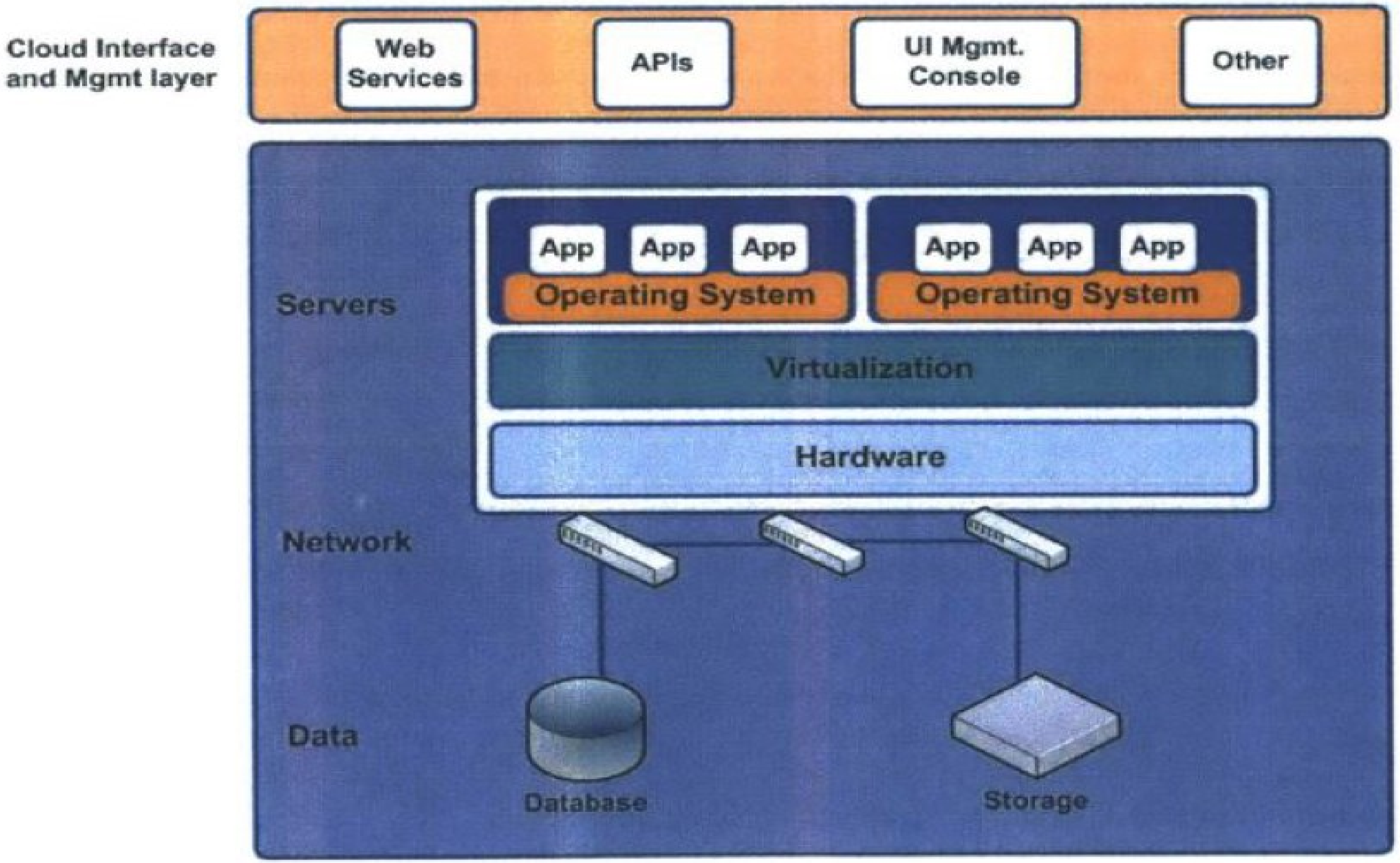
SUCCESSIVE™ DIGITAL

KEY PRINCIPLES

Include microservices, containers and orchestration, Infrastructure as Code (IaC), and Continuous Integration/Continuous Delivery (CI/CD).

the microservices need to be designed in a fault-tolerant way so they can handle failures gracefully.

Fault tolerance is the characteristic that allows a system to function as expected even when certain components or modules fail


Cloud Interface and Mgmt layer
Web Services
APIs
UI Mgmt. Console
Other
Servers
App
App
App
Operating System
App
App
App
Operating System
Virtualization
Hardware
Network
Data
Database
Storage

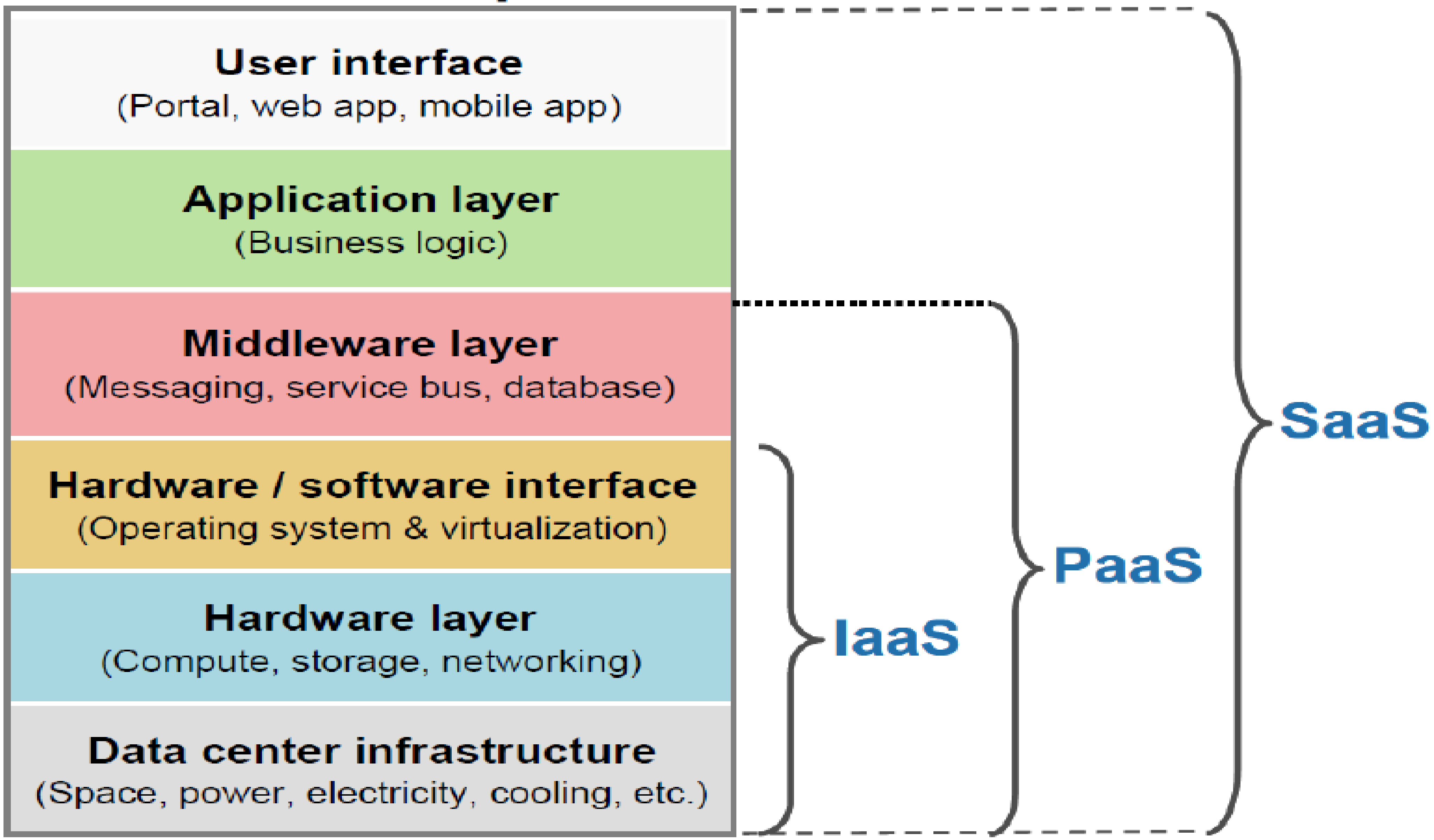
CAPEX VS OPEXP MODELS

CapEx models require lots of investments to start a business

These include hardware, utilities (phone, water, electricity), office space, furniture, networking, etc.

In OpEx model, clients use the Cloud and Operate them remotely using any connected device.

Public Cloud Spectrum: Stack View



User interface
(Portal, web app, mobile app)

Application layer
(Business logic)

Middleware layer
(Messaging, service bus, database)

Hardware / software interface
(Operating system & virtualization)

Hardware layer
(Compute, storage, networking)

Data center infrastructure
(Space, power, electricity, cooling, etc.)

CLOUD NATIVE APPS

Cloud-native application development is a software development trend that involves building and running applications in distributed environments (aka “the cloud”).

It provides organizations with a streamlined approach to building and updating apps while ensuring quality and minimizing risk.

A streaming service like Netflix runs on the cloud, but users can interact with it through their smart TVs or other devices.

BENEFITS OF CNA

Cloud-native applications are designed to work across the cloud, both embedded in its architecture and facilitating cloud access.

The benefits of cloud-native software include resiliency and efficiency associated with cloud operations, as well as lightweight and flexible design suited for the task.

Cloud-native applications take advantage of platforms and processes that were born in the cloud. They're highly scalable, easy to change, and connect to cloud services to extend capabilities without a lot of coding.

SOFTWARE CONTAINERS

Software containers are portable, integrated operating environments encompassing an application and all the software components it needs to run.

Containers have become a wildly popular alternative to complex virtual machines because they are:

Small, typically measured in megabytes or less

Quick to deploy

Reusable and portable
Once you write an application in a container, you can move it to any platform that supports containers (which is most of them), and the application will run without a hitch. Your application isn't bound to a single cloud platform — it can run on any device with enough support resources, from a laptop to a supercomputer.

CONTAINER IS THE KING

Each container has an associated IP address

This IP may change upon restart of the container

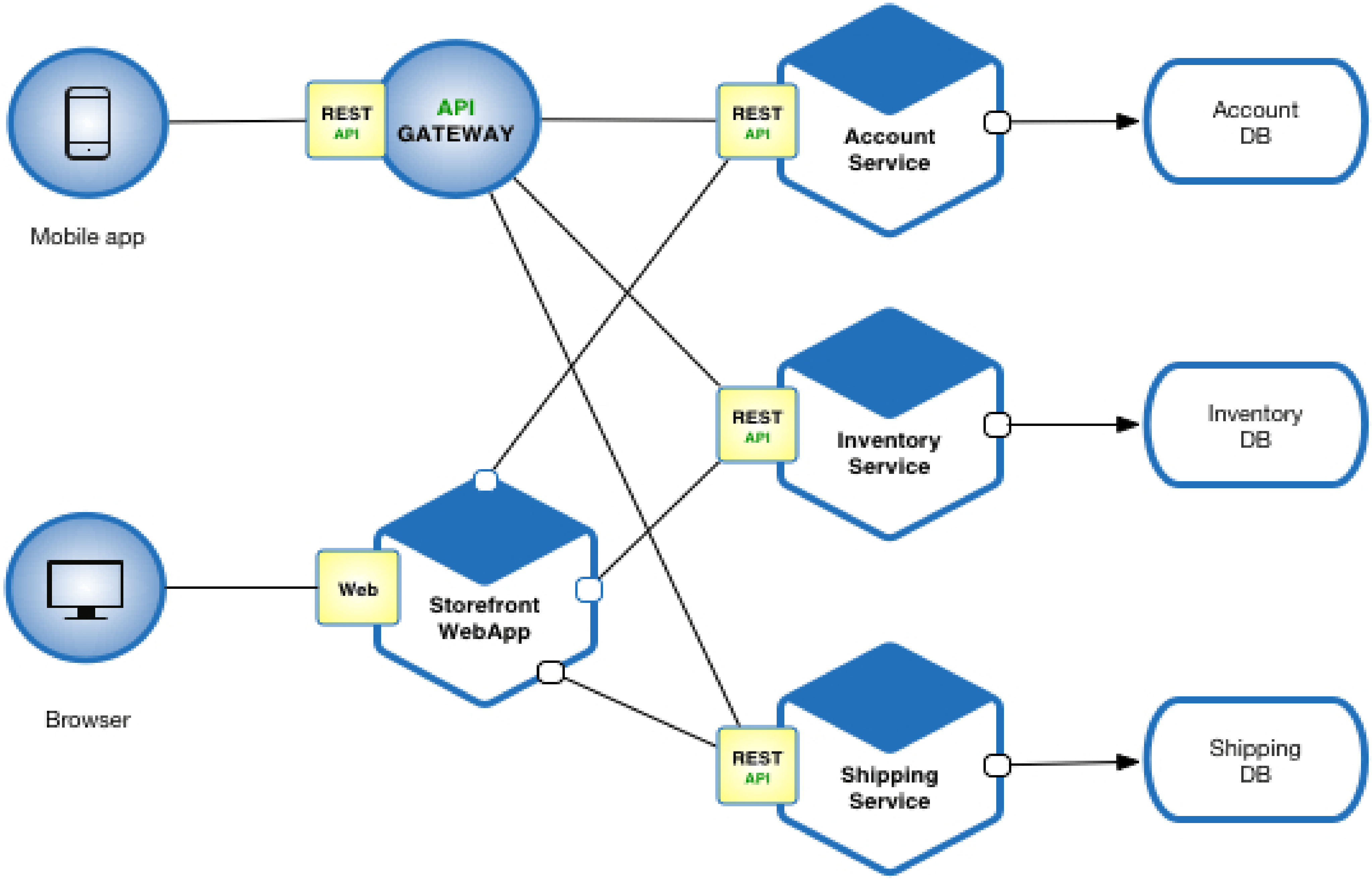
However, Kubernetes has DNS ability so that it can manage multiple containers and redirect the load among multiple containers (called container load balancing).

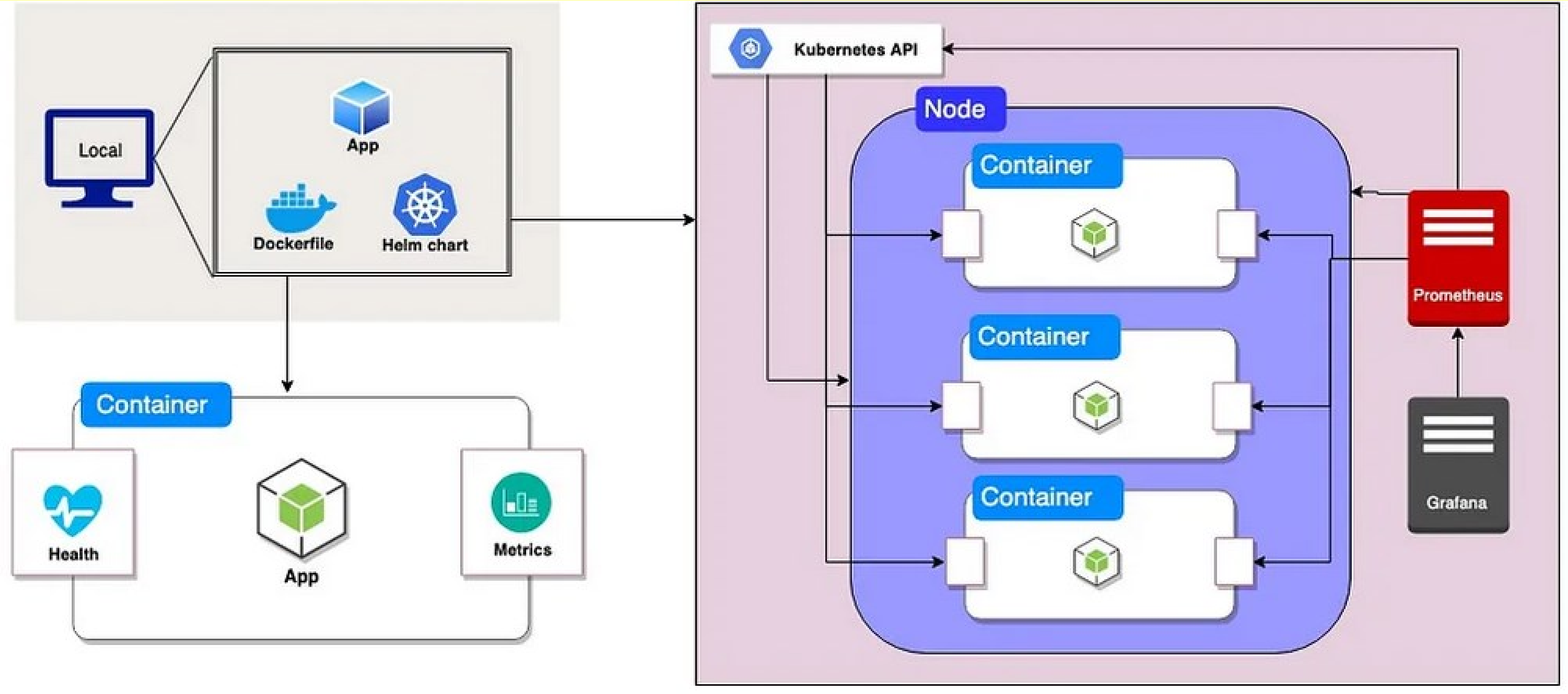
MICROSERVICES

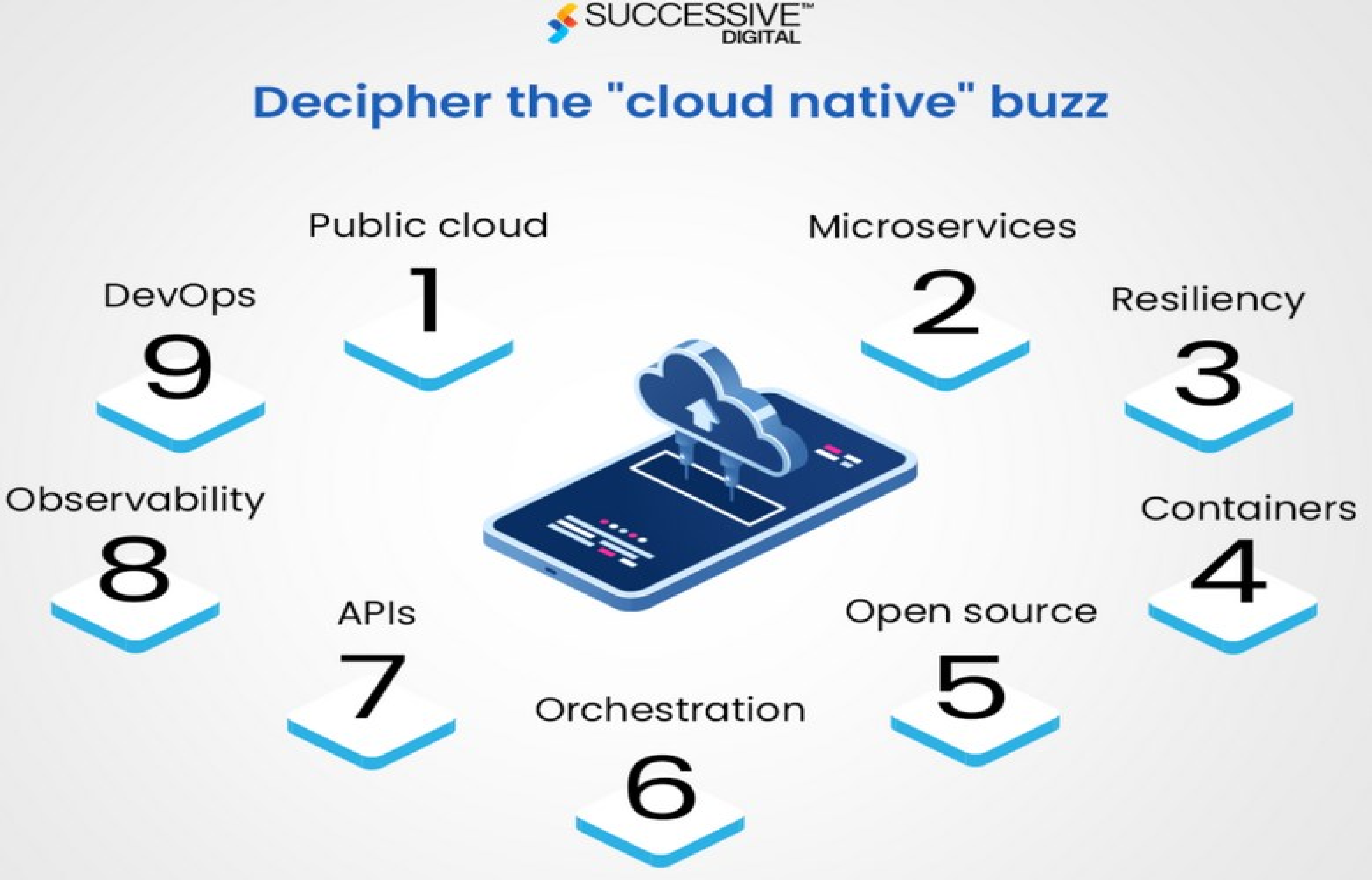
Microservices are loosely coupled software services that can be strung together to create an application.

This saves developers from having to reinvent the wheel and makes applications flexible and extendable.

Applications composed of microservices are pieced together like Lego blocks with minimal custom coding, facilitating faster and more reliable development. You can swap or add in new services without extensive integration testing.


Mobile app
REST API
API GATEWAY
REST API
Account Service
Account DB
REST API
Inventory Service
Inventory DB
Browser
Web
Storefront WebApp
REST API
Shipping Service
Shipping DB


Local
App
Dockerfile
Helm chart
Container
Health
App
Metrics
Kubernetes API
Node
Container
Container
Container
Prometheus
Grafana



SUCCESSIVE™ DIGITAL

Decipher the "cloud native" buzz

1 Public cloud

2 Microservices

3 Resiliency

4 Containers

5 Open source

6 Orchestration

7 APIs

8 Observability

9 DevOps

SUCCESSIVE™
DIGITAL

The Twelve-Factor App

The Twelve-Factor App is a methodology for building distributed applications that run in the cloud and are delivered as a service. These best practices help developers build applications that are portable, scalable, agile, reliable and consistent.

1 Codebase. Use one codebase, even when building cross-platform apps. Address the needs of specific devices with version control.

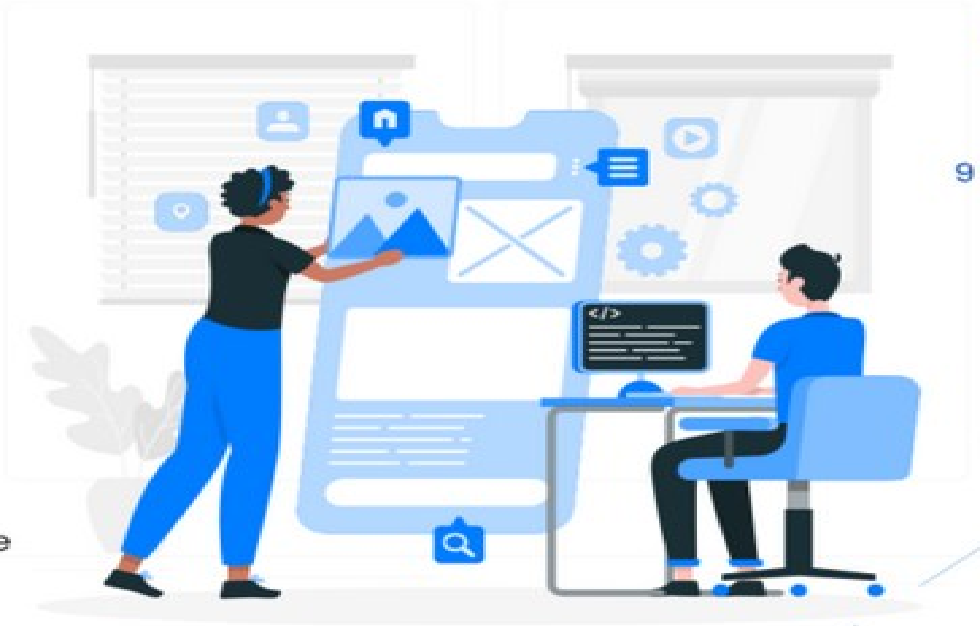
2 Dependencies. Explicitly declare and isolate all dependencies.

3 Configuration. Design the app to read its config from the environment.

4 Backing services. Treat back-end services as attached resources to be accessed with a URL or other locator stored in config.

5 Build, release, run. Strictly separate build and run stages.

6 Processes. Execute the app as one or more stateless processes. Data that must be persistent should be stored in a stateful backing service.



7 Port binding. Use port binding to export services.

8 Concurrency. Scale out apps horizontally, not vertically.

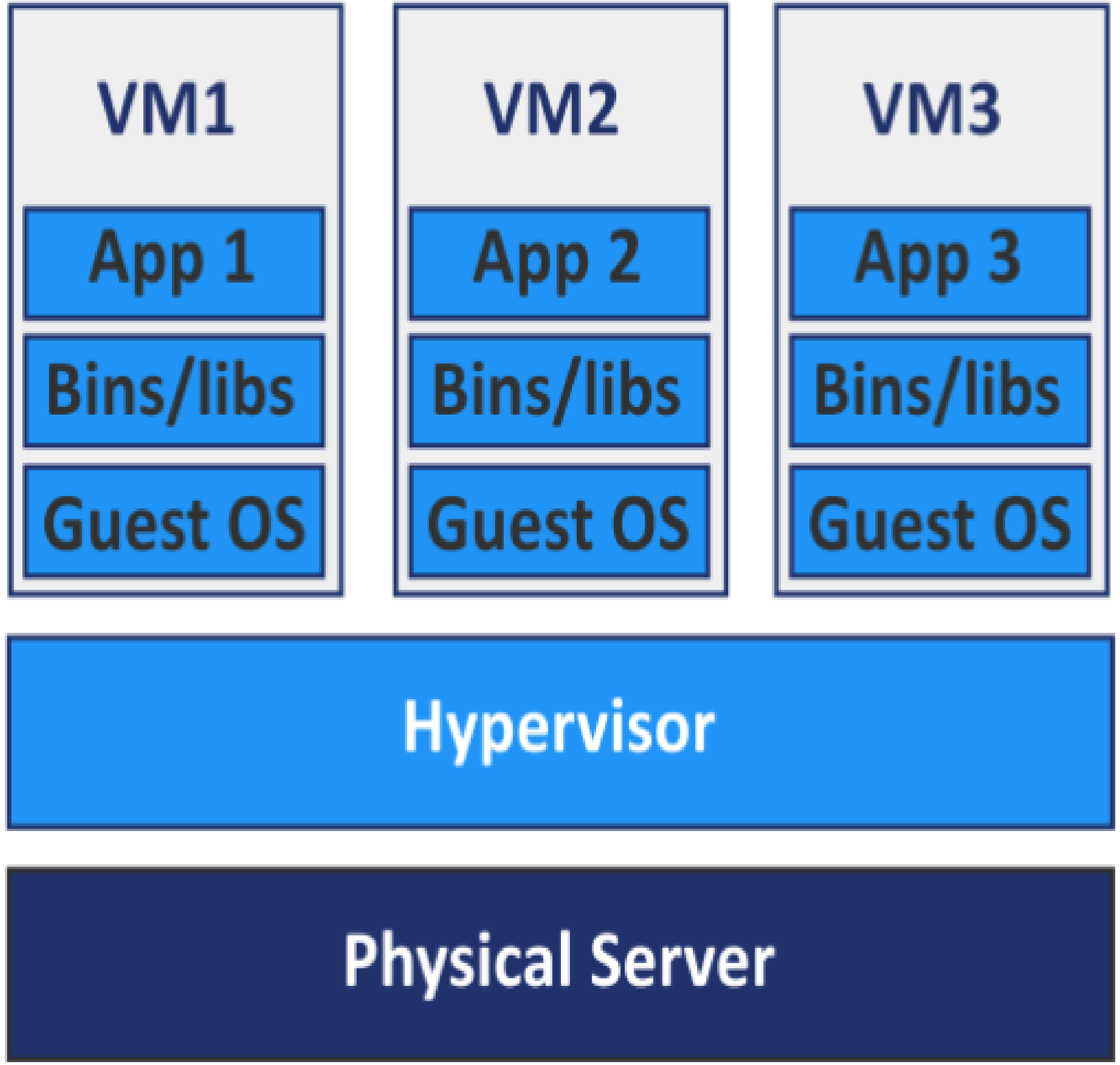
9 Disposability. Use fast startups and graceful shutdowns to maximize robustness.

10 Parity. Facilitate continuous deployment by ensuring that development, staging and production environments are as similar as possible.

11 Logs. Treat logs as event streams.

12 Admin processes. Run admin tasks as one-off processes from a machine in the production environment that is running the latest production code.

Virtual Machines



VM1

App 1

Bins/libs

Guest OS

VM2

App 2

Bins/libs

Guest OS

VM3

App 3

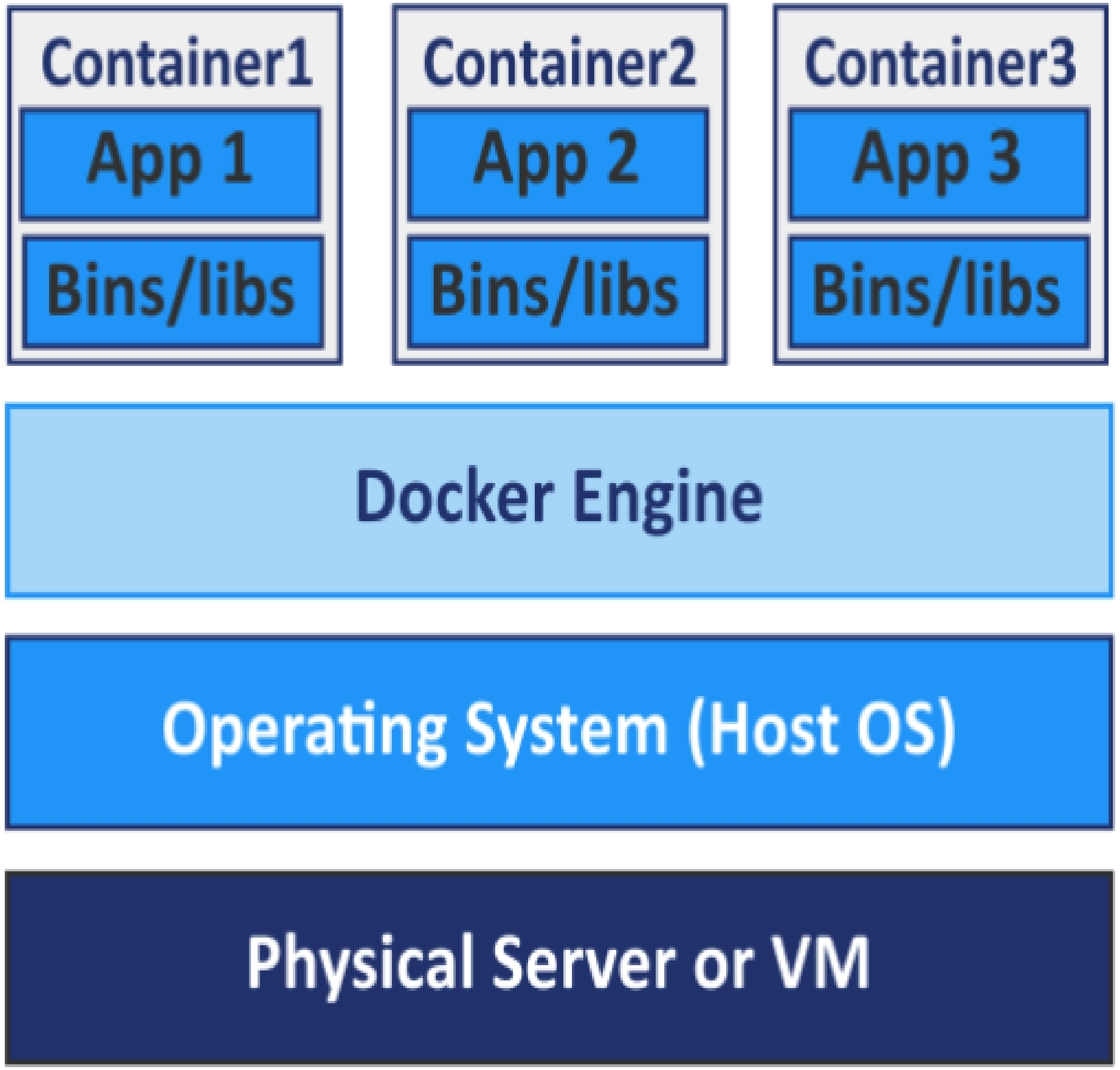
Bins/libs

Guest OS

Hypervisor

Physical Server

Containers



Container1

App 1

Bins/libs

Container2

App 2

Bins/libs

Container3

App 3

Bins/libs

Docker Engine

Operating System (Host OS)

Physical Server or VM

APPLICATION PROGRAM INTERFACES (APIS)

APIs are software connectors that expose functionality that other software can use.

A good example of an API-enabled application is Google Maps.

With Google Maps, a developer of a real estate application can integrate mapping functionality from Google into its program by requesting geographic information using APIs.

Without needing to build map functionality from scratch or install an application on their server, imagine how much time the developer can save.

APPLICATION PROGRAM INTERFACES (APIS)

An Application Programming Interface (API) is a mechanism employed by two or more software programs to share information.

In cloud-native systems, APIs play a crucial role in integrating loosely connected microservices.

Instead of outlining the step-by-step process to attain a particular outcome, an API provides information about the data a microservice requires and the results it can provide.