



The Problem – Traditional Apps Don't Scale

In the past, we built web apps like running a restaurant in one building. If too many customers arrived, everyone had to wait. If the kitchen caught fire the entire restaurant closed.

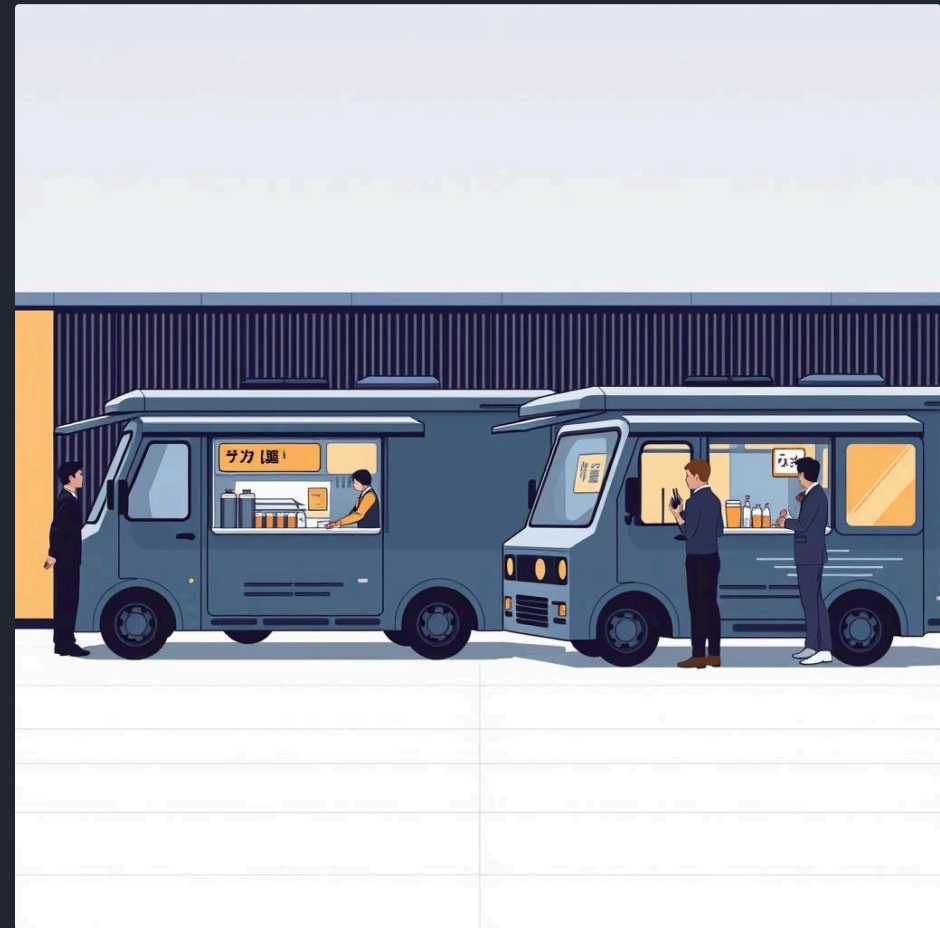
Enter Cloud-Native Thinking

The Old Way



One monolithic server. All components tightly coupled. Scale = bigger hardware.

The Cloud-Native Way



Multiple containerized services. Each independent. Scale = add more containers.

The Shift: Cloud-native apps are built like a fleet of food trucks instead of a single restaurant. Each truck (container) carries everything it needs—code, runtime, dependencies. If one breaks, the fleet manager (Kubernetes) deploys another instantly. Need more capacity? Add more trucks.

Core Technologies Behind Cloud-Native



Containerization with Docker

Packages an app and all dependencies into a self-contained unit that runs identically anywhere. Docker creates lightweight, portable containers that include everything needed to run an application - code, runtime, system tools, libraries, and settings.



Orchestration with Kubernetes

Automates deployment, scaling, and recovery of containers across infrastructure. Kubernetes manages container lifecycles, handles load balancing, service discovery, and ensures your applications stay healthy and available.



Microservices Architecture

Breaks large systems into smaller, independent components that communicate via APIs. Each service can be developed, deployed, and scaled independently, making the overall system more resilient and maintainable.

What Makes an App Cloud-Native?

Cloud-native systems embody five foundational principles that enable modern scalability and resilience:

Containerized

Each component runs in its own isolated, portable environment with defined runtime and dependencies.

Immutable

Instead of patching running servers, we replace entire container images—eliminating configuration drift.



Orchestrated

Automated deployment, scaling, and self-healing via Kubernetes ensures optimal resource utilization.

Observable

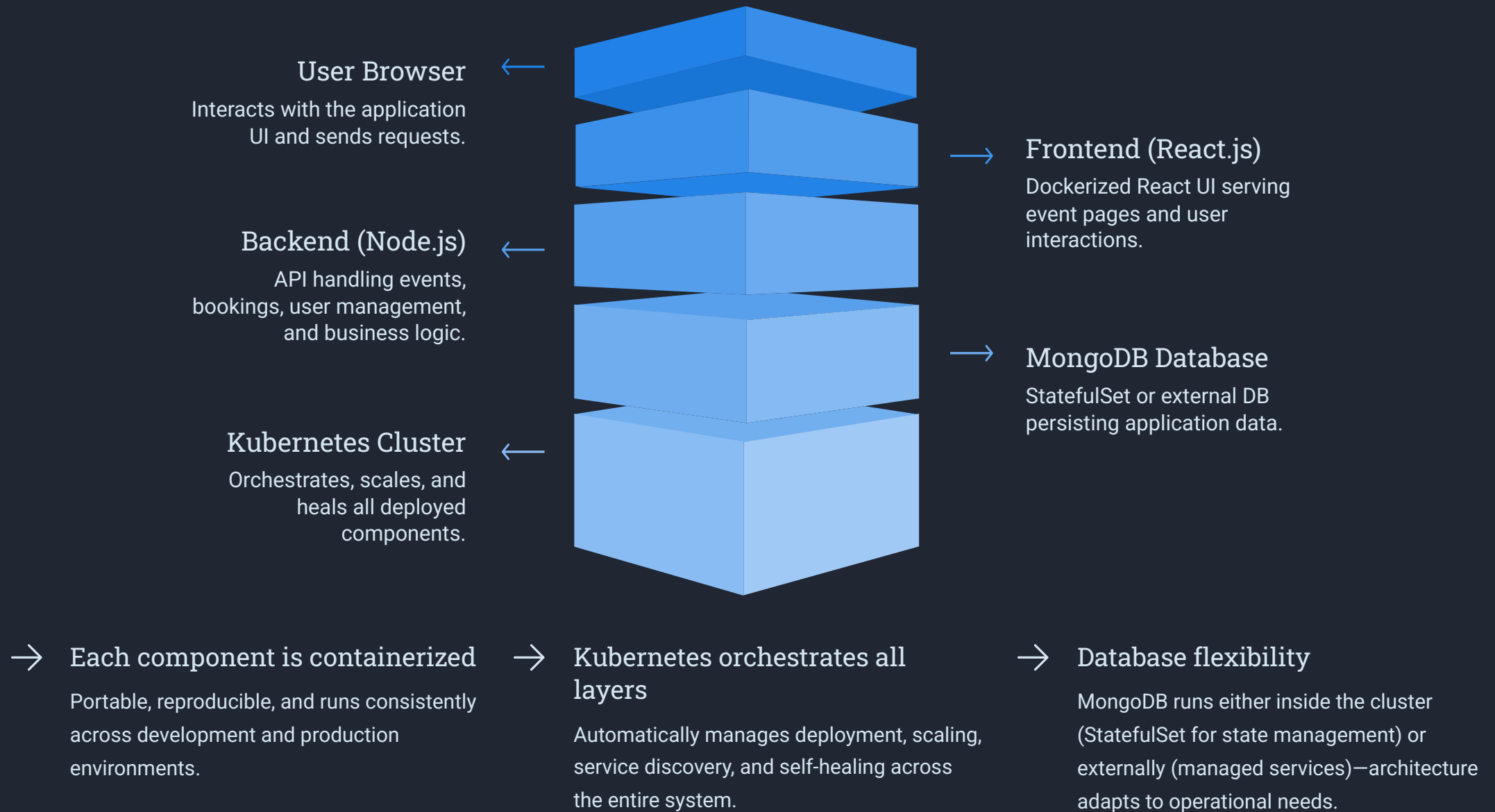
Comprehensive metrics, logs, and health checks drive intelligent automation and reliability decisions.

Declarative

Infrastructure described entirely in code (YAML manifests) for reproducibility and version control.

Analogy: A cloud-native system is like a living organism—each cell (container) operates independently, yet all work together through coordinated signals (Kubernetes control loops) to maintain health and adapt to demand.

Our Event Planner System – Modular and Scalable



Result: A system that scales independently at each layer, recovers automatically from failures, and handles traffic spikes without manual intervention for seamless event planning.

From Code to Cloud: The Deployment Lifecycle

1

Code

Source code in version control (Git)

2

Containerize

Package with Dockerfile including runtime and dependencies

3

Registry

Push image to container registry for central storage and versioning

4

Deploy

Kubernetes pulls image and runs containers based on manifest

5

Scale & Monitor

System automatically scales and self-heals based on metrics and desired state

Key Concept — Declarative Infrastructure: We describe our desired state in YAML manifests—"I want 3 replicas of my backend service." Kubernetes continuously ensures the system matches this declaration, replacing failed instances and scaling as needed.

Thank You!

Questions & Discussion

GitHub Repository: <https://github.com/HanaAbdelbari/Cloud-Native-Web-App.git>

We hope this presentation provided valuable insights into the power of cloud-native development. Embracing these principles allows for more resilient, scalable, and efficient applications.