AKKA

TECHNICAL REFERENCE

Akka Cell-Based Architecture Guide

Table of Contents

Introduction

Cell-based architectures are on the rise in recent years. The desire to isolate workloads for improved elasticity and agility as well as to isolate the "blast radius" or area of impact from failures has been a major driver. This pattern is also ideally suited for cloud-native applications.

The rise of AI-centric applications and agentic AI is creating an even greater need for agility and increased velocity that adopting a cell-based architecture can bring. In addition, cell-based architectures, when combined with event sourcing (providing a level of agency), are a great fit for this approach, as agentic systems and LLMs are inherently event driven, and also bring new challenges in dealing with the isolating impact of AI APIs which can often be unreliable, slow and behave unpredictably.

Akka is a platform for building and running applications designed to deliver guaranteed resilience and achieve up to 99.9999% availability. This level of availability is accomplished by running applications concurrently across multiple regions, with seamless, transparent data replication between regions to ensure consistency and fault tolerance.

Akka Platform deploys cell-based applications into Bring Your Own Cloud (BYOC) regions within major cloud providers, including AWS, GCP, and Azure. These regions operate entirely within the customer's Virtual Private Cloud (VPC/VNet), ensuring customers maintain custodial ownership of the infrastructure and the environment where the Akka applications are executed. Akka BYOC regions are installed, monitored, and updated by the Akka team, enabling organizations to benefit from Akka's operational expertise while maintaining full control over their cloud environments.

What is cell-based architecture?

A cell-based architecture in computing is, in essence, an architecture for systems built of modular self-contained "cells" that can be built into applications, networks, and other systems. In application architectures, cell-based architectures refer to applications or systems built of modular "cells" which are self-contained deployment modules containing all required interdependent components across data, application logic, memory and compute, that can be easily deployed and scaled, typically onto container-based run times like Kubernetes and Docker, and elastically scaled up/down.

Historically, the earliest cell-based examples were in telecommunications, where the cells were building blocks for mobile networks, giving rise to the term cellular communications and cell phones. In computing hardware, cell-based paradigms have been applied widely in building memory, processors, high density storage, and compute.

Cell-based architecture tenets

- **Cells should be fully self-contained and modular, with failure Isolation** — Logic and State should be sharded or partitioned with each logical cell having self-contained complete state, following the Bulkheading pattern (modeled after ships with bulk-heads to isolate hull breaches.) Cells should have fault isolation boundaries that restrict the effect of failure on other cells. Smaller cells can generally isolate failures more effectively.
- **Cross-cell replication** — replication between cells for both resilience and inter-cell and inter-cell and inter-service communication

should be seamless and automated.

- **Cell fail-over / fail-back** — failover for resilience and disaster recovery should also be seamless and automated.
- Truly self-contained independent cells should support rolling updates and deployments without system wide outage or a need for complex deployment techniques such as blue-green deployments. Cell-based architectures may also enable easier A/B testing.
- Run anywhere — multi-cloud — application cells should be self-contained and run anywhere, across Azure, GCP, and AWS and hybrid.
- **Local development and testing, CI/CD DevOps** — cell-based architectures enable greater velocity and agility for development teams. Developers can develop independently locally and deploy continuously. Cell-based architectures also simplify continuous and automated testing, improving reliability, and enabling techniques like canary deployments at a more granular level.
- **Reusability of code across all cells** — granularity in cell-based architectures can facilitate greater reuse while also enabling cells to run independently so that some cells may be updated at different times from others despite sharing code.

Benefits of a cell-based architecture

- **Up to five or even six nines of uptime** — Data can be replicated between cells across availability zones for resilience and across data center or cloud regions for disaster recovery. The limited "blast radius" of errors further improves resilience.
- **Zero downtime deployment**s — Cells are self-contained and independent, so deployments to one cell won't affect others. This enables continuous deployments without downtime.
- **Security advantages** — Cell-based architectures allow granular security controls and isolation of where and how cells run. This supports data sovereignty, privacy, and compliance with regional regulations. They're especially suited for Zero Trust Security.
- **Multi-cloud and hybrid cloud readiness** — Cell-based architectures can be deployed across cloud regions, data centers, and edge environments. This makes them ideal for multi-cloud and hybrid cloud architectures.
- **Ease of development and testing** — Cells support local development and testing, making them easy to build, code, and maintain over time.
- **Code reusability** — Code can be shared across cells without creating complex interdependencies, streamlining maintenance.
- **Efficient hardware utilization** — Modular cells combining memory, compute, and storage can be tuned for performance. Paired with the Actor Model and Event Sourcing, this eliminates blocking calls and optimizes hardware use.
- **Data sovereignty and privacy** — Independent, self-contained cells help meet data sovereignty and privacy requirements more easily.
- **Lower operational complexity** — Cells' modularity simplifies root cause identification and limits the impact of failures, reducing **operational costs and outages.**

Challenges and considerations

- **Implementation considerations** — Cell-based architectures demand careful attention to partitioning strategies and a deeper understanding of distributed systems. Techniques like Event Sourcing, CQRS and Sagas help maintain cell isolation and address failures effectively.
- **Architectural and design needs** — These systems require upfront sophistication to ensure the architecture aligns with the cell-based model.
- **Observability hurdles** — Cross cell interaction in large, complex systems can make observability and tracing issues more challenging.

- **Supporting technologies** — Cell-based architectures thrive when paired with other patterns like Event Sourcing, CQRS, and the Actor Model. Automation technologies such as Kubernetes, CI/CD Tools, infrastructure as code, and service meshes further boost reliability and performance.

Challenges with many cloud technologies and cell-based architectures

Adopting a Cell-based architecture will not yield the scale and efficiency without changing the paradigm away from traditional 3 tier architectures and technologies such as Spring or .Net, Postgres, Kafka, Spark, Flink and others. There is still limited abstraction of resources for developers, inherent interdependencies that limit the ability to isolate cells and limit the blast radius of failures, further complicating testing and DevOps. These technologies also don't scale efficiently and elastically, often requiring over provisioning to ensure handling peak loads or to support blue-green deployments.

Key challenge: blocking calls and complexity. Even with a Cell-based architecture, traditional distributed microservice, database and event bus technologies still run into challenges with blocking calls tying up resources resulting in brittle applications. Locks and synchronous calls ripple across distributed systems that don't adopt Event Sourcing and the Actor Model. The lack of isolated data locality further complicates deployment, operations, and performance and resilience and the added unnecessary network hops further add latency, introducing bottlenecks. Cells built with these technologies may be isolated from other cells during failures, but within these cells, there remains complexity and brittleness.

Akka's platform enables these seamlessly while enabling developers to focus on business logic, as you will see next.

How Akka supports a cell-based architecture

The Akka Platform will significantly de-risk this effort and will drastically accelerate the delivery of a cell-based project. The Akka SDK makes the power of Akka accessible to a much wider community of enterprise developers across, drastically reduces delivery time, and increases maintainability of all services by a well-designed separation of concerns (domain, application, API.)

Akka benefits for cell-based architectures

- Akka manifests the cell architecture in a very cost-effective way both in terms of cloud infrastructure and operational effort. The Actor Model and Akka were driving forces in the global adoption of architecture patterns such as bulkheading, data locality, and event sourcing that provide critical building blocks of cell-based architectures.
- Akka provides high scale, multi-region elasticity, agility, and resilience with unparalleled performance: (6ms: p99 read latency)
- Provides up to 5 Nines of uptime
- Minimizes operational costs and accelerates agility and velocity
- Enables easy cloud portability (Azure, GCP, AWS, virtual private cloud, and more)
- Provides an architecture that can also transcend the cloud and run on-prem or on the edge.

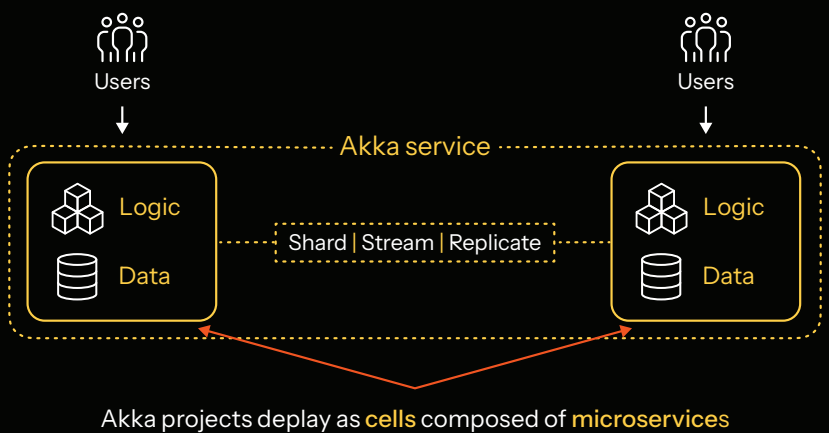
The Akka Platform cell-based architecture

Akka Platform is the only PaaS that follows the cell-based architecture pattern by default, and with multi-region, multi-master replication and federation.

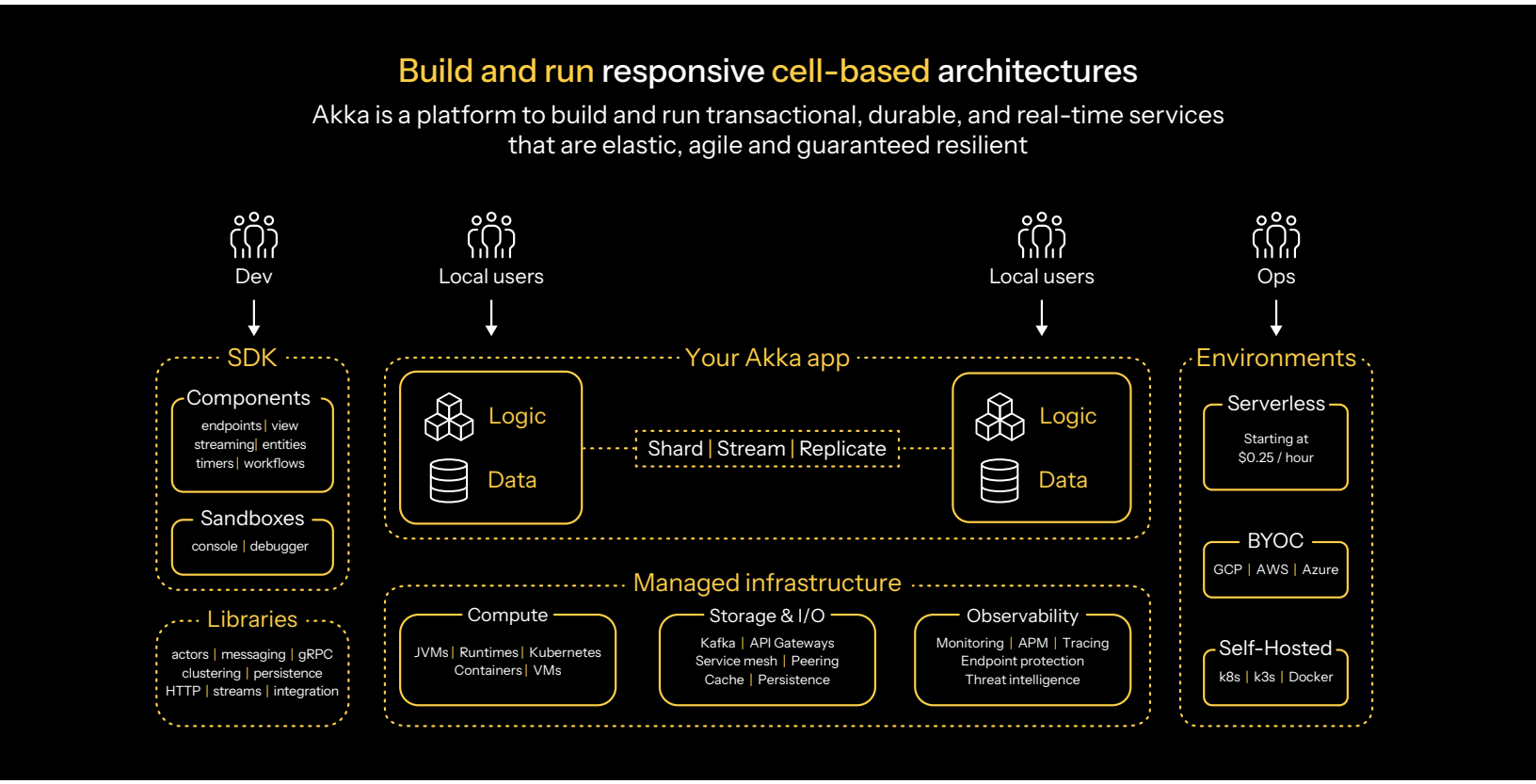
Akka services are each an independent containerized microservices comprising both application logic and data. Each service is elastic, able to scale up or down independently based on load. Akka thrives with up to millions of self-contained autonomous units which can be combined into application cells that run wherever, with no dependency on centralized and heavy weight infra.

Key cell concept: distribute logic & data together

Akka apps act as their own in-memory, durable databases that can change locations and recover from failures

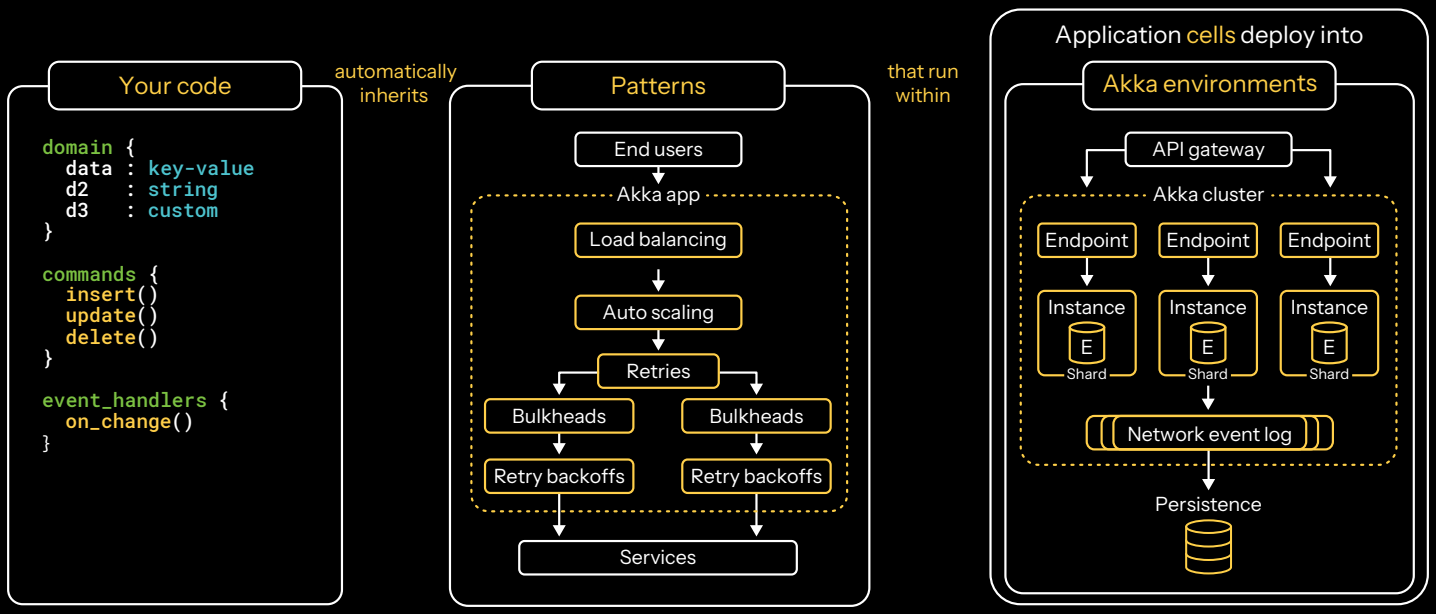


Akka services are managed in application projects to create cells that enable seamless operations with up to 5 Nines (99.999%) uptime. **Data locality** with self-contained **in-memory data stores** ensure minimum latency, maximum throughput, while also enabling a run-anywhere approach. Cells can **deploy anywhere**: multi-cloud, in data centers and the edge.



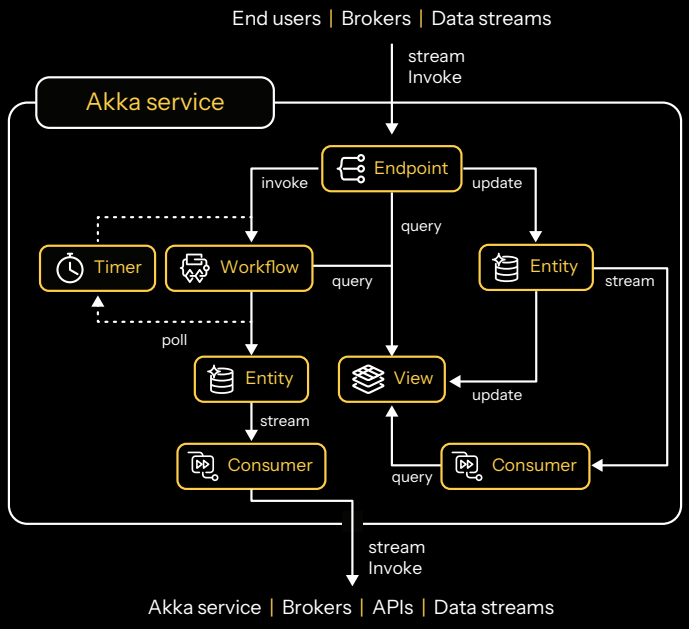
Your application code is unpolluted with configuration or infrastructure code, based on simple domain objects, yet inherits the best practice patterns built upon Akka's 15 years of experience with highly distributed, highly elastic event sourced applications. Customers get the power of the Actor Model and event sourcing without needing to have experience with either.

Distributed computing simplified



Each Akka service is composed of a set of standard Akka components used in our SDK as building blocks for cell-based applications. Each Akka component is, itself built upon the Actor Model, isolating resources and enabling granular levels of isolation and scaling. Individual Akka components can scale independently and elastically. Each service can itself be viewed as a cell with independent deployment and operational control.

Akka Components are packaged into an Akka Service



The Akka SDK requires learning six simple components, with simple unpolluted business logic, enabling rapid ramp up for developers, faster development, and lower maintenance effort.

Components

Entities
Build apps that act as their own in-memory, durable, and replicated database.

Views
Access multiple entities or retrieve entities by attributes other than entity id.

Endpoints
Design HTTP and gRPC APIs.

Streaming
Streaming producers and consumers enable real-time data integration.

Workflows
Execute durable, long-running processes with point-in-time recovery.

Timers
Execute actions with a reliability guarantee

Offline dev

Local console, event debugger, sandboxes, and separation of concerns ensures high dev velocity with minimal local configuration.

Separation of concerns

Decorators and APIs create technical debt, cross-team friction, and coherence leaks. With Akka, build and test your domain code separately. Akka components inject intelligence at build.

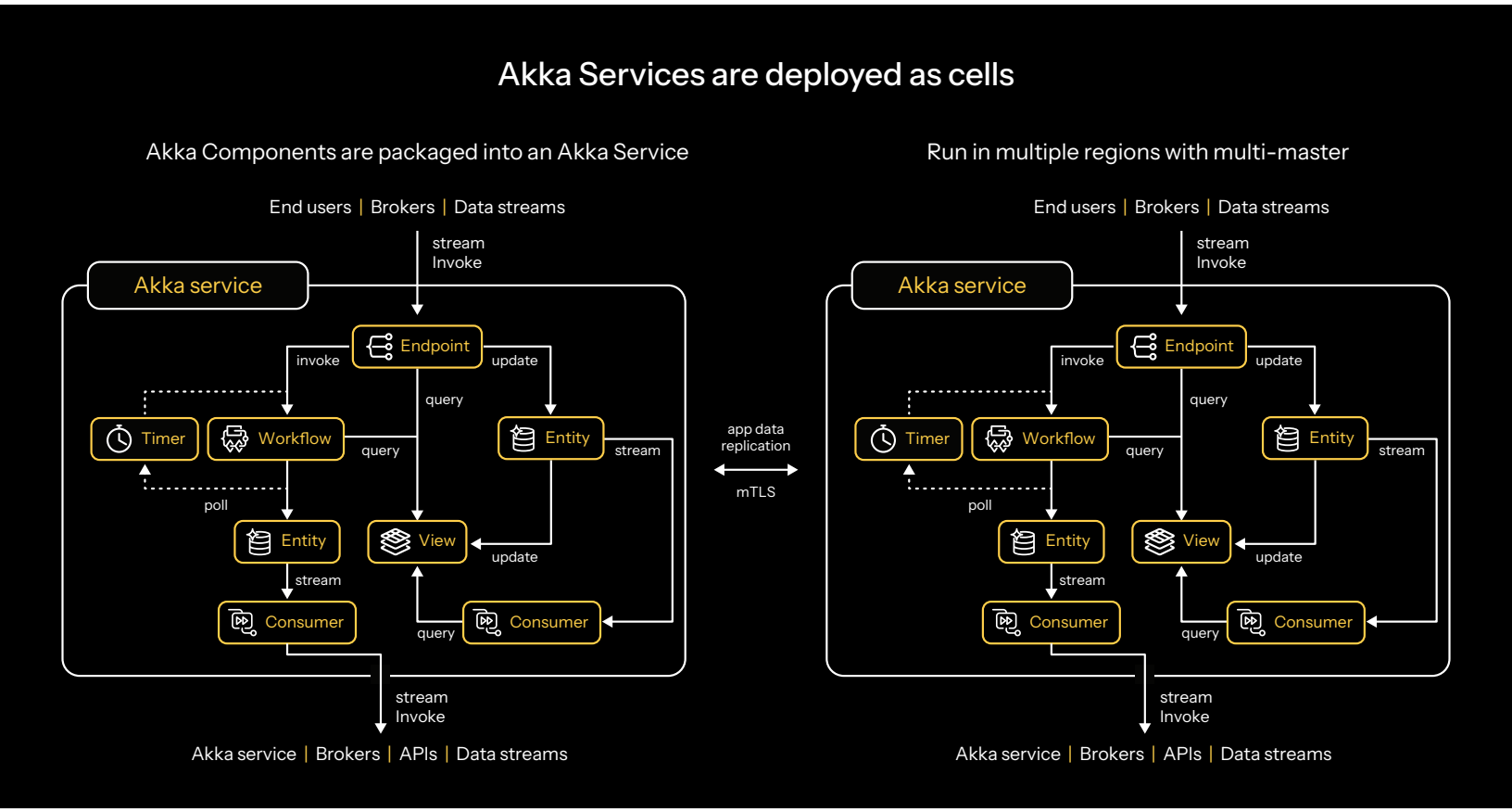
With Akka:
Domain separation

```
domain  {
   data         : custom
   method()     {}
   unit_test()  {}
}
```

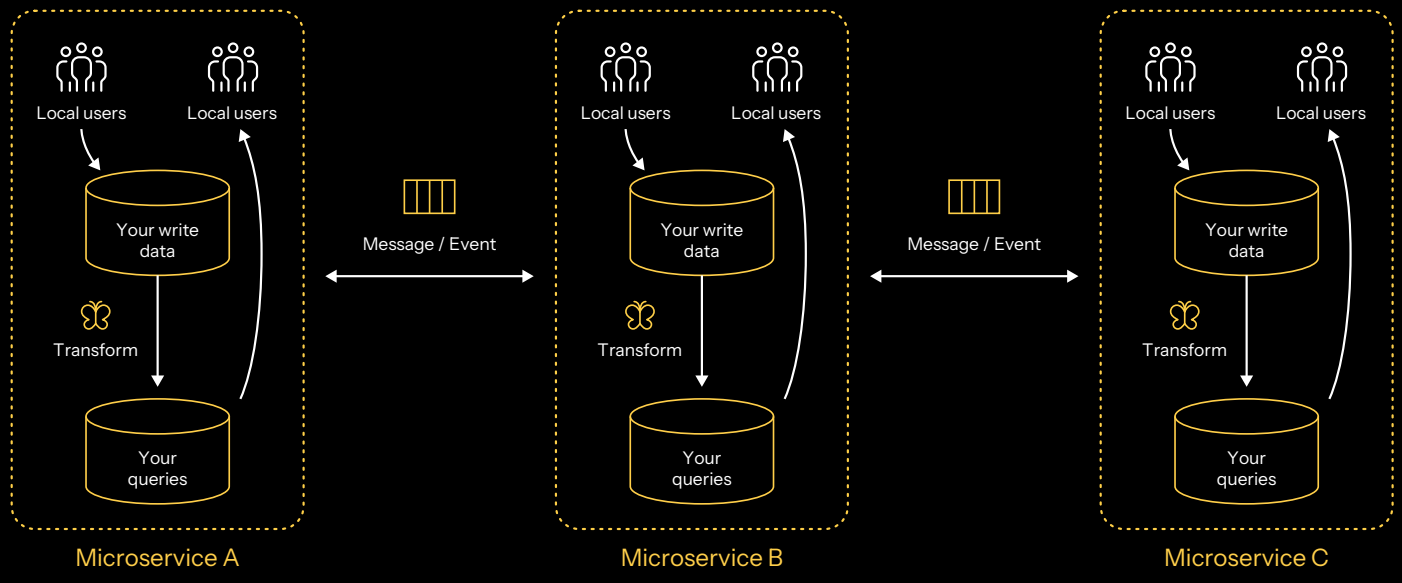
Without Akka:
Polluted domain logic

```
@non_domain_tag
domain     {
   @more_debt_tags
   data :    custom

   @even_more_tags
   method() {
         leaky_messaging_API()
         leaky_data_API()
   }

   @more_dependencies
   unit_test() {}
}
```

Akka service cells can deploy anywhere and continuously replicate data with each other.



All communication follows the Actor Model and Event Sourcing out-of-the box. Developers do not need to be experts in these to get the benefits. Communications is brokerless, real-time, event-driven and highly efficient over gRPC.

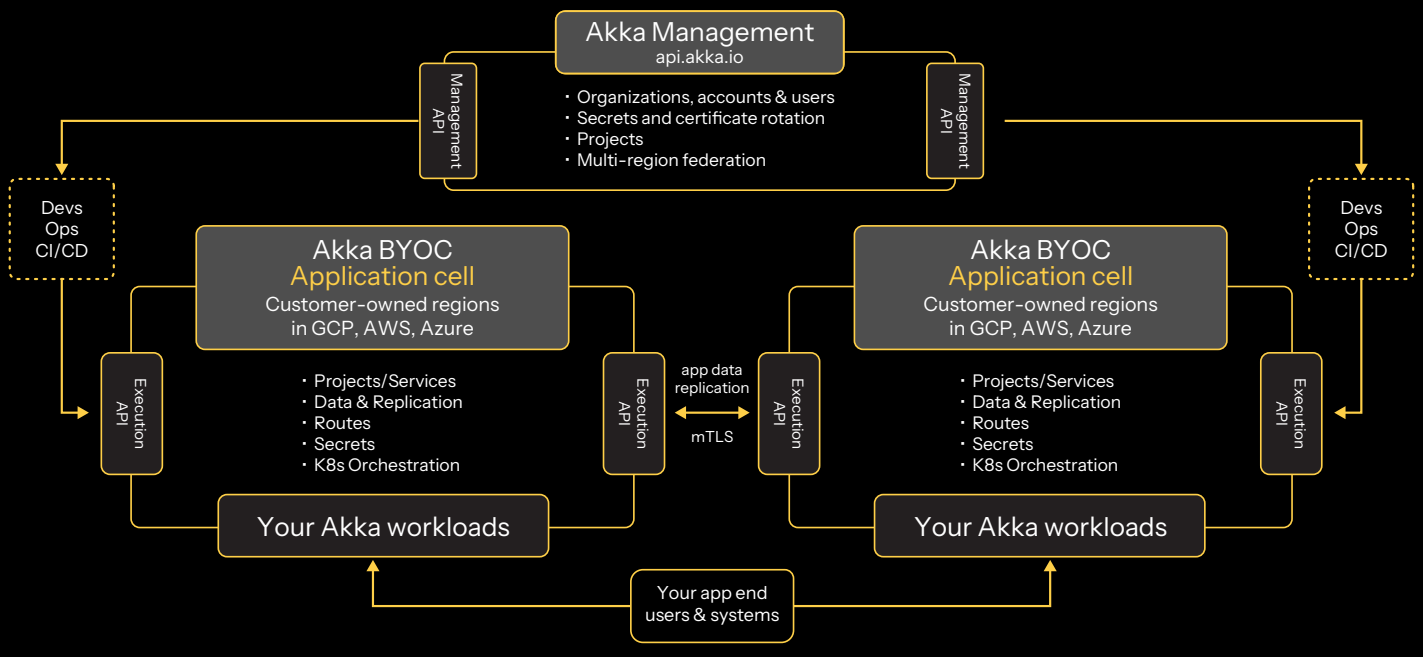


Architectural parallels

Feature	Akka Platform	Cell-based architecture
Basic units	Component (built on Actor) & Service (Services combine closely related components deployed together)	Cell
Communication	Message-passing	API gateways & eventing
State management	Private, encapsulated	Cell-local storage
Failure containment	Supervisor hierarchies	Fault-isolated boundaries
Scalability mechanism	Component replication/sharding	Cell replication/partitioning

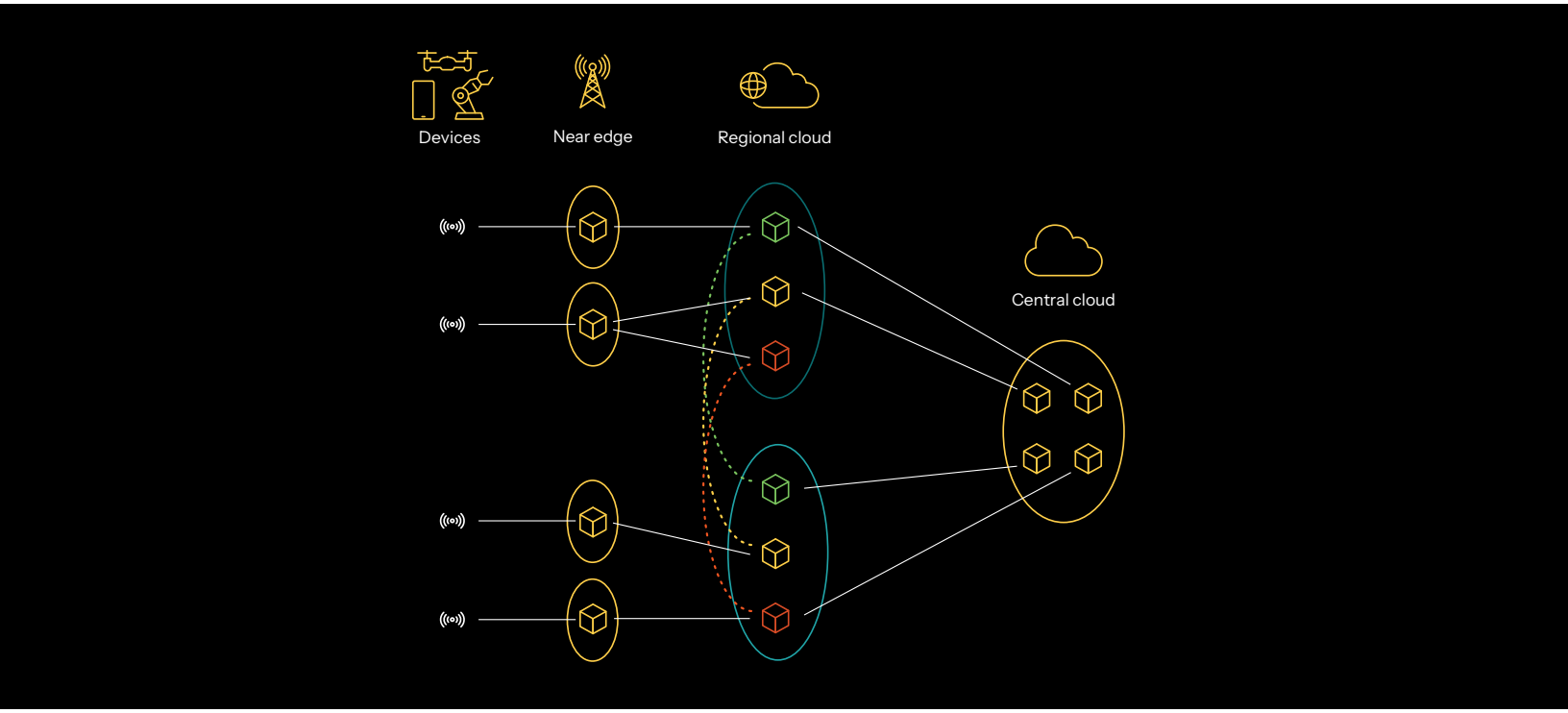
Deployment environment
Akka platform deploys cells into one or more Cloud regions, depending on the requirements of each application in terms of target regions, resilience and elasticity requirements, etc.

Akka Platform multi-region replicated cell deployment



Edge deployment and integration

Akka also seamlessly integrates from edge to Cloud. Deploy Akka services in a cell-based paradigm and replicate data automatically between application cells. Technologies such as Kafka are not well suited to Edge deployments, while others like Flink and Postgres are definitely architected only for Cloud or Data Center deployment.

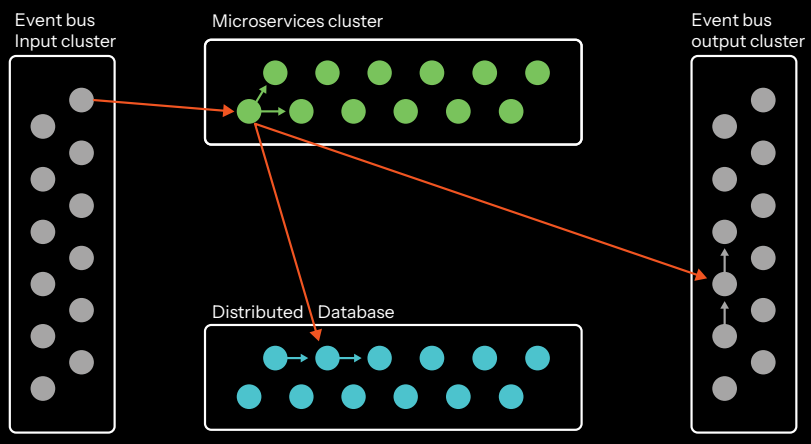


Easily create and deploy digital twins following cell-based architecture.

The problem with blocking calls: lock entanglement

Blocking calls have a ripple effect tying up resources across multiple heterogeneous clusters

A single operation can tie up multiple threads across multiple clusters.



Locks multiply in distributed systems. This amplifies brittleness and complexity across systems tying up resources. The result is less reliable, harder to scale or change, and slows release velocity.

Akka solves this with the Actor Model and the Event Sourcing approach. State changes are captured and replicated across **local in-memory databases**, with persistence following an append-only approach to eliminate the need for locks.

Akka platform inherently delivers cell-based microservices architecture applications. The platform is intrinsically based on Event Sourcing and Streaming. Enabling processing billions of events and enabling deploying cells anywhere, whether Edge, Cloud and On Premise. You get fully tested and battle-hardened capabilities.

With traditional cloud services and technologies, you will have to build your own framework to enable a cell-based architecture, as well as to manage testing, deployments and operations at scale.

Building cells with Akka

With the Akka Platform you build apps with the simple but powerful Akka SDK, not directly with the APIs for Akka Libraries. Applications built using the Akka Platform are easy to develop (5-10x faster than with the libraries), and easier to operate.

The Akka Platform SDK makes the power of Akka accessible to a much wider community of enterprise developers across you, and increases maintainability.

Developers can develop offline. There's a local console, event debugger, sandboxes, and separation of concerns (domain, application, API) ensures high dev velocity with minimal local configuration.

There is a strict separation of concerns. With Akka, you build and test your domain code separately. Akka components inject intelligence at build. Layers are implemented as the three packages: domain, application, API. More on the architectural and deployment models can be found here: doc.akka.io/concepts/index.html

The Akka Platform SDK offers the following components which deploy as Akka Services, microservices which deploy as elastic Cells on the Akka platform, or as Self-managed Nodes, cells that you can manage and scale yourself, deployed in your Kubernetes environment.

Stateful components

- Event Sourced Entities — for managing Event state
- Key Value Entities — for managing Key-Value state
- Workflows — for stateful long running choreography and orchestration.

End points — components to define APIs

- HTTP Endpoints — for RESTful APIs
- gRPC Endpoints — for gRPC APIs

Other components

- Consumers — for consuming and producing event streams
- Views — for Querying State from Entities and Workflows for CQRS pattern.
- Timers — for Timed/scheduled actions and tasks

Summary

We have seen how cell-based architectures, particularly combined with event sourcing and microservices patterns, support greater agility, resilience and elasticity. The Akka Platform excels at enabling these robust cell-based architectures. Akka inherently supports the principles of cell-based design, providing features like self-contained, independent services with built-in data locality and in-memory data stores, ensuring low latency and high throughput. The platform's use of the Actor Model and Event Sourcing eliminates blocking calls, enhances fault isolation, and simplifies replication, crucial for resilience and disaster recovery.

Key strengths of Akka Platform for cell-based architectures:

- Inherent support for cell-based principles: Self-contained services, data locality.
- Robustness: Actor Model and Event Sourcing for fault isolation and resilience.
- High availability: 5 Nines of uptime.
- Developer efficiency: Akka Platform SDK simplifies development.
- Scalability and elasticity: Multi-region and cloud portability.
- Superior performance: Low latency, high throughput.
- Edge deployment capability.
- Ideal fit for applications that must integrate LLMs and other AI models due to their slow performance, event driven nature and the need to maintain stateful interactions to ensure resilient applications that deliver a responsive experience.

The Akka Platform SDK further streamlines development, abstracting complexities and offering pre-built components for state management, APIs, and event stream processing. This dramatically accelerates project delivery and reduces maintenance, allowing developers to focus on business logic rather than infrastructure. Akka also boasts excellent scalability, multi-region elasticity, and cloud portability (Azure, GCP, AWS), while remaining suitable for edge deployments. In contrast to traditional cloud technologies, Akka provides a natural foundation for cell-based architectures, delivering superior resilience, performance, and developer velocity right out of the box.

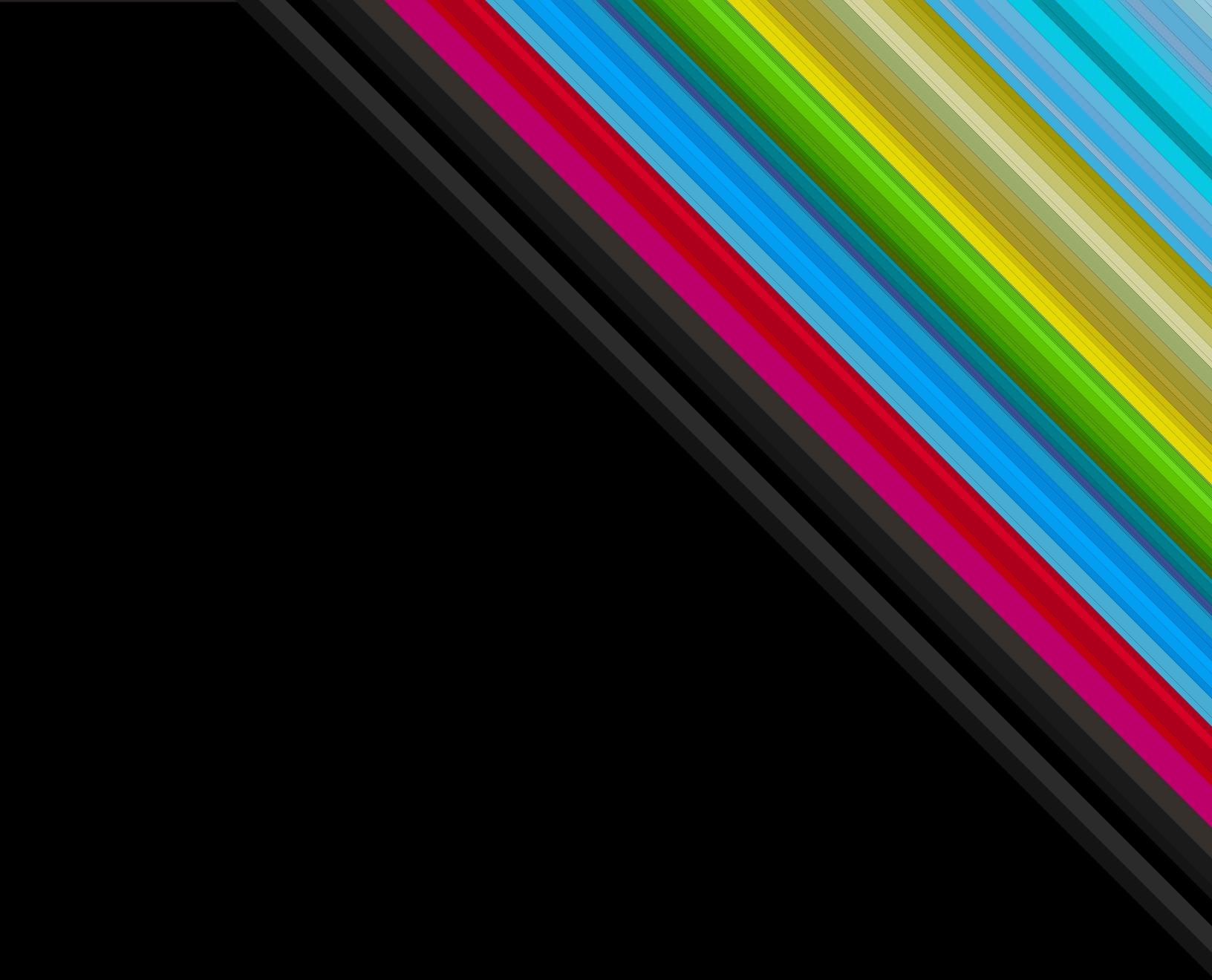
Next steps

Talk it through

Contact us to discuss your unique requirements with our solution architects and engineers. We will — without any pressure — create a tailored, detailed TCO estimate of what it costs to build and operate a multi-region app with any performance profile.

Try it out

With Akka, your team can build and deploy your own multi-region replicated app in minutes. Follow the 5-minute tutorial for new accounts at Akka.io to deploy, run, and test an application that runs across three regions, each in a different cloud provider.



AKKA