AKKA

WHITE PAPER

Architect's Guide to 99.9999% Application Availability

How to reduce the cost and complexity of delivering always-on business-critical applications.

AKKA

Executive summary

Business resilience has never been more important, and application resilience is fundamental to it. A lack of application resilience often puts a halt to business and operations. Downtime causes lost revenue, reputational damage, and regulatory penalties. However, meeting six nines of availability (99.9999%) is challenging and expensive for modern, mission-critical applications.

Traditional approaches demand costly, complex infrastructure changes; most clouds only guarantee up to 99.99% uptime, leaving app owners responsible for higher levels of resilience. Akka apps incorporate additional nines of availability without triggering an architecture rethink or order-of-magnitude cost increase.

Akka's simplified path to six nines

- **Multi-region resilience:** Run your apps with active-active data replication across multiple regions or clouds. If one region fails, others remain functional.
- **Built-in data replication:** Using CRDTs, Akka automatically handles data convergence, avoiding high-latency global transactions.
- **No app database required:** Applications serve as their own in-memory, durable database, reducing the licensing and infrastructure overhead of distributed SQL.
- **Redundant infrastructure:** Each Akka region has identical infrastructure which delivers redundancy. Each region is identical, mirroring upgrades, patches, and maintenance.
- **Proven track record:** In 15+ years, Akka has never experienced a Sev1 incident causing data loss, and it indemnifies against reliability-related losses.

Key business benefits

- **Reduced complexity:** Application-layer resilience replaces heavy reliance on costly infrastructure redundancy.
- **No-downtime updates:** Rolling upgrades and event-driven operations enable continuous delivery without service interruption.
- **Vendor leverage:** Multi-cloud deployment prevents lock-in, improving flexibility and negotiation power.

Next steps

- **Get started:** Visit Akka.io to get started at no cost.
- **Try multi-region:** Deploy Akka across clouds to test seamless failover.
- **Get a TCO estimate:** Collaborate with Akka's team for a tailored cost model.
- **Future-proof your architecture:** Embrace event-driven, eventually consistent design for high availability at scale.

Table of Contents

Business resilience has never been more important, and application resilience is fundamental to it. A lack of application resilience often puts a halt to business and operations. Downtime causes lost revenue, reputational damage, and regulatory penalties. However, meeting six nines of availability (99.9999%) is challenging and expensive for modern, mission-critical applications.

Traditional approaches demand costly, complex infrastructure changes; most clouds only guarantee up to 99.99% uptime, leaving app owners responsible for higher levels of resilience.

Akka apps incorporate additional nines of availability without triggering an architecture rethink or order-of-magnitude cost increase.

The impact of downtime

Application downtime leads to revenue loss, customer dissatisfaction, brand damage, ops disruption, regulatory consequences, increased diagnostic costs, and lower employee morale.

Availability	Annual downtime
99.9999%	31.5 secs
99.999%	5.26 mins
99.99%	52.6 mins
99.9%	8.76 hours

Outages are costly. The Uptime Institute surveyed IT leaders who indicated that 54% of outages cost more than $100,000 and 16% cost more than $1 million. Forrester found that the average organization experiences 830 minutes of annual downtime at a cost of $5.6M. After a July 2024 update caused widespread outages, Crowdstrike estimated a $60M revenue loss from abandoned customers and took a $22 billion market cap hit that was impacted by Delta Air Lines, who launched a $500M lawsuit in response.

The business impact of downtime is more nuanced than the precision of decimal points. Consider how users react to:

- One 4-hour outage vs. 240 dispersed one-minute outages
- Peak-hours outages vs. off-hours outages

The structure and impact of downtime are baked into Service-Level Agreements (SLAs), which quantify uptime and downtime along with enumerating the financial remedies available when a breach is observed.

Uptime is a mandate

Availability is a mandate, appearing as controls or requirements in regulations, compliance initiatives, security, and DevOps standards.

Regulation/standard	Availability requirement
ISO 27001	Requires business continuity and disaster recovery to ensure high availability.
ISO 22301	Focuses on minimizing downtime and ensuring service continuity.
SOC 2 (Availability)	Mandates measurable uptime and incident response for service reliability.
PCI DSS	Ensures uninterrupted service for payment processing systems.
GDPR	Requires availability of personal data and services under Articles 32 & 33.
HIPAA	Demands contingency plans for data availability in healthcare systems.
CMMC	Ensures system availability for defense-related contractors.
NIST SP 800-53	Requires resilience and continuous availability in federal systems.
SOX	Indirectly requires availability for financial reporting in public companies.
FedRAMP	Mandates specific availability SLAs for federal agencies.
FIPS	Supports availability in compliant cryptographic systems.
CSA STAR	Evaluates cloud provider reliability, including uptime.
SRE (Site Reliability)	Uses SLOs and error budgets to maintain measurable availability.
ITIL	Focuses on minimizing downtime via incident management processes.
CIS Controls	Ensures system resilience and availability to mitigate cybersecurity risks.
FISMA	Requires high availability for federal data and applications.
MAS TRM	Mandates availability and resilience for financial services in Singapore.
APRA CPS 234	Requires availability for financial institutions in Australia.
Basel III/IV	Banking regulations require uninterrupted service availability.
Uptime Institute Tiers	Certifies data center performance and availability (e.g., Tier III/IV).
ANSI/TIA-942	Defines availability standards for telecom infrastructure.
DORA Metrics (MTTR)	Measures time to recover from outages, critical for uptime.
DORA Metrics (Change Failure)	Reduces downtime caused by failed deployments.

As an example, the EU's Digital Operational Resilience Act (DORA)—not to be confused with Google's DORA metrics for DevOps velocity—sets strict requirements to ensure the operational resilience of financial entities, including banks, payment institutions, investment firms, insurance companies, crypto-asset providers, and SaaS providers that serve financial institutions.

This includes downtime minimums, availability SLA mandates, rapid recovery mechanism, resilience testing, and infrastructure redundancy. Failure to meet these requirements will cause financial penalties, regulatory fines, legal liability, termination of contracts or exclusion from the EU market, or regulatory oversight intensification.

Business resilience relies upon availability

Business resilience is the ability of an organization to anticipate, prepare for, respond to, and recover from disruptions while maintaining continuous operations and safeguarding key business functions. It encompasses a combination of disaster recovery, risk management, cybersecurity, operational continuity, and adaptability to changing conditions—ensuring that a company can withstand shocks such as cyberattacks, system failures, supply chain disruptions, or market shifts without significant loss of functionality or reputation.

In the context of software systems, business resilience means designing architectures that support **high availability**, **rapid failover**, and **minimal downtime** while proactively mitigating risks through redundancy, automation, and disaster recovery planning. The goal is not just to recover from failures but to ensure they have minimal impact on users and business operations.

Achieving better business continuity and resilience in SaaS systems requires a **direct investment in higher availability SLAs**, as maintaining uptime during failures demands robust disaster recovery (DR) and resilience strategies.

When companies commit to stringent SLAs—such as 99.99% availability or higher—they must ensure that their systems can withstand outages with minimal downtime and data loss. This requires multi-region failover, automated recovery mechanisms, and continuous data replication to maintain real-time integrity – automatic when you create Akka apps. Without these measures, even brief service disruptions can result in financial penalties, customer churn, and reputational damage.

Availability requires agility: safe and continuous change

A high availability SLA is not just a function of system design—it is also a function of a software team's ability to operate, adapt, and respond. If a dev team cannot rapidly diagnose, fix, and improve an operational system, uptime and resilience suffer. Organizations aiming for 99.99%+ availability must invest not only in technology but also in team agility, automation, and operational excellence to ensure that availability is not just a goal but a sustained reality.

The availability of a software development team is directly tied to its ability to rapidly diagnose and resolve incidents, deploy fixes, apply security patches, and adapt to operational changes. Achieving high availability SLAs requires not only responsive teams **but also an application and infrastructure that support near-continuous improvements and rapid rollbacks, like what Akka provides**.

Without fast deployment pipelines, real-time observability, and automated incident response, even a highly skilled team can struggle to maintain uptime. Additionally, reducing dependencies on key individuals and eliminating bureaucratic bottlenecks ensures that operational issues can be addressed swiftly. Ultimately, maintaining high availability is not just about system resilience—it's about enabling fast, safe, and continuous changes to keep the software reliable and performant under all conditions.

The cost of adding nines

Adding additional nines forces an uncomfortable conversation, as the economics to ensure availability have typically been so significant that many organizations avoid prioritizing higher availability until they are under a mandate or facing severe injury to their business if they fail to act.

As SLAs become more demanding, the **complexity and cost of maintaining availability grow exponentially**. Moving from 99.9% to 99.999% uptime means reducing allowable downtime from nearly nine hours per year to just five minutes, which often necessitates active-active architectures rather than simpler failover models.

Historically, systems that want to "add a nine" face an order of magnitude increase in engineering effort and operational complexity. Each nine usually includes fundamental architecture, process, culture, and organizational changes.

Complexity and cost concerns affect organizations of all sizes. Even hyperscalers like AWS and GCP are challenged to achieve six nines where resourcing is significant. Most major cloud providers offer SLAs in the range of 99.9% to 99.99% for their services. This includes AWS EC2, RDS, Elastic Load Balancers; Azure VMs; and GCP Compute Engine and Kubernetes Engine. These services underpin millions of other global services, which usually pass along their cloud SLA to their customers.

These low guarantees are why Gartner advises clients to "focus on application resilience, not infrastructure redundancy." The onus is on application owners to meet these requirements which cannot be delegated to your cloud provider.

Availability	Changes	Dev cost	Ops cost
99%	Orchestration, backup & recovery, observability, DevOps processes	$	$
99.9%	Single-region cluster deployment, standby environments, failover	$$$	$$
99.99%	Multi-region deployment, infrastructure redundancy, continuous health checking, failover automation, on-call operations, active-passive regional clustering	$$$$	$$$$$
99.999%	Redundancy at every layer, active-active regional clustering, decoupled event-driven architecture, synthetic real-time transactions, rolling no-downtime updates, DDoS resistance	$$$$$$$$	$$$$$$$
99.9999%	Chaos engineering, latency-sensitive traffic steering, continuous deployment	$$$$$$$$	$$$$$$$$$

Achieving six nines for apps

An application that does the following, can achieve 99.9999% availability:

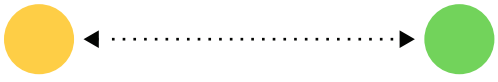
- Serve reads and writes simultaneously in multiple locations
- Reliably replicate all of its data between those locations
- Converge in a timely fashion
- Enable seamless upgrade or failover to other regions
- Continue uninterrupted with new versions or in new regions

To achieve this, an application must have a design where data interdependencies do not restrict system scalability, and that application must run on a resilient-from-within runtime, infrastructure, and data fabric:

Multiple locations
The same application can handle requests in multiple locations simultaneously. If any location cannot service requests, other locations are unimpeded and can continue to service requests.

This creates a form of application redundancy that will enable the application to continue operating when one or more locations fail as long as all locations do not fail simultaneously. Further, if each location operates within a different cloud provider, then the risk of all-locations failure becomes exceedingly unlikely.

Data replication
Each location must be able to service any potential end-user request. With each region handling all potential user requests, each region must operate against the application's entire data set. This data set is managed locally and updated with changes that occur from the same app in other locations. Each location has its own copy of the application's data and a reliable, transparent, continuous system for replicating data between locations.



Local strong consistency with distributed eventual consistency
Each location must be able to operate independently of other locations. This requires all local data transactions to be strongly consistent. Replication is a non-zero cost and time (it takes light 45ms to travel across the United States), which means a transaction that is committed locally is not guaranteed to have completed its replication event before a location failure occurs.

Some systems block the completion of the local transaction until the replication event has been confirmed by all receiving regions, effectively crippling the system's performance and making it unable to scale concurrent users deterministically.

Instead, cross-location data must always be guaranteed to converge deterministically and correctly when being replicated to another location. This must occur while recognizing that unexpected network issues or location failures could cause the replication actions to occur significantly later. The same application running in different locations must assume and understand that the data it is locally working with may not be the latest.

Data convergence
Data within one location must merge with its data counterparts in another location. Any data transaction that occurs in a local region is strongly consistent, and its record is a fact. That change is then replicated to other locations.

By definition, any replicated data that a new location receives occurred in the past. It is history and may not reflect the present state as the data could have further changed in the originating location. The replicated data must be incorporated quickly and consistently into the local version of the data.

This merge is data convergence, which is taking data changes that happened in the past and correctly integrating them into the local understanding of the data so that the originating location and replicated location create the same data interpretation. As a result, future actions will yield the same results.

If the convergence backlog overwhelms a location, it is on a path to failure. Convergence failure from overload would have its traffic rerouted to another location, increasing its load and potentially leading to cascading failures of each region. In practice, this is rare as performance tends to flatten out as a system scales rather than fail.

Interesting side note: convergence with read-or-write replicas based upon Conflict-free Replicated Data Types (CRDTs) is fast and efficient, with data guaranteed to merge as it is a monotonic set. This compares to historical practices, which typically implement a crude Last Write Wins (LWW) merge function that causes unintended data overrides. Convergence in systems without CRDTs either requires end users to resolve merge conflicts (slow) or to algorithmically make a win/loss decision (e.g., LW, W), which makes you take responsibility for the science behind distributed transactions (i.e., slow, error prone, and layers of compensation).

Intelligent clients and routing
The loss of a single location will not affect your uptime, but the client accessing this service must have a mechanism to route its traffic to an available location. This is a largely solved problem in distributed systems, as clients can have multiple locations with automated failover incorporated, a system can deploy global hostnames that map multiple IP addresses to a single domain, or an Anycast IP address can be used which leverages IP and BGP routing to send different requests to different locations based upon latency, location or failure.

Any nines with Akka

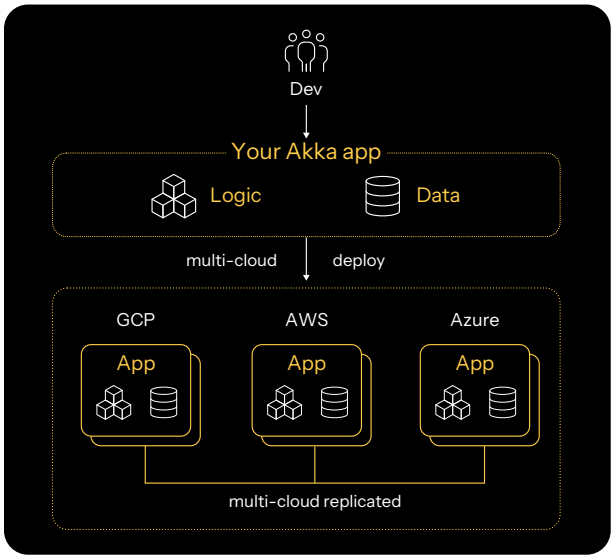
Many Akka customers can achieve (and have achieved) 99.9999% availability, the equivalent of an always-on system. With this level of availability, you can operate 24/7, while ops rest easy with a system that has built-in redundancies and automation for failure recovery.

Akka takes an approach to application design and system operations that can enable your services to exceed six nines availability. We simplify the development required to have an application that is six nines-ready with an SDK that ensures applications are deployable across many regions while also significantly lowering the cost of operating apps across many regions.

With Akka, you build and operate multi-region/multi-cloud replicated applications that are elastic, agile, and resilient:

Multiple regions

An application built with Akka can operate simultaneously across multiple regions. Our serverless offering at A*kka.io provides five global regions*. Customers can also set up private regions with an Akka Bring Your Own Cloud (BYOC) deployment. Akka regions are multi-cloud, allowing them to operate across and within different hyperscalers and data centers. The global regions hosted at Akka.io span GCP, Azure, and AWS.



Redundant infrastructure

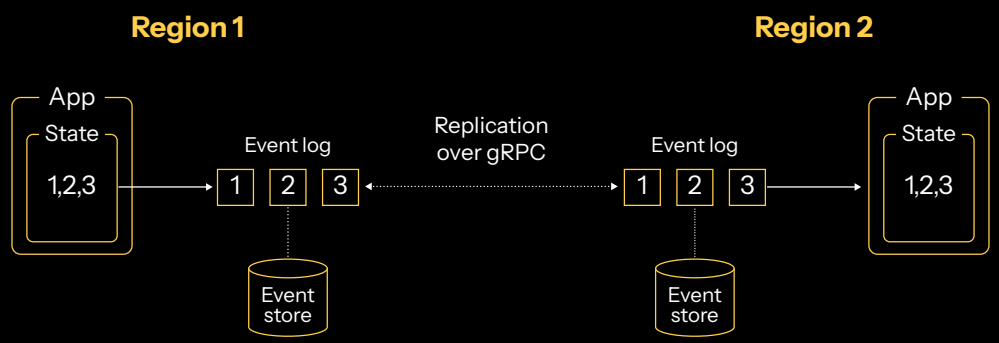
Each region duplicates the Akka-managed infrastructure, creating redundancy through duplication. Akka applications take responsibility for their own outcomes by managing their underlying infrastructure (i.e., resource allocation, failover, resilience) to ensure they achieve a local SLA (i.e., latency, uptime, concurrent requests) that is deemed responsive under any type of external factor. The local infrastructure is itself replicated, duplicated, and clustered where appropriate.

The combination of redundant local infrastructure and regionally replicated infrastructure builds in multiple layers of redundancy. While this level of redundancy can multiply costs, Akka efficiently only consumes the physical compute needed to address the volume of requests it services. This optimization from within, combined with standby regions and replication filters, dramatically lowers total system costs to a level that is not much larger than single region scale-out systems!

App data replication

Akka applications act as their own in-memory, durable database. The application you create becomes its own in-memory system of record for its data. Changes to the application's data are automatically captured as events and persisted within a local storage engine. If you have deployed the same application into multiple regions, then all your application's data is transparently, reliably, and continuously replicated to all the regions that your application is running within.

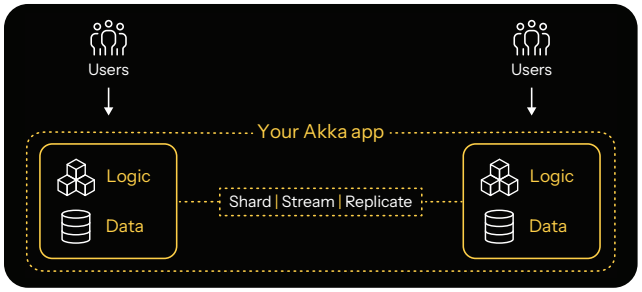
Connectivity between regions and encryption of your application's data are handled by an Akka runtime environment that coordinates cross-region ingress and egress. Data replication has an only-once guarantee coordinated by an embedded brokerless messaging engine that operates over gRPC.



Replication filters

For compliance purposes, such as with GDPR and PII, certain data must be contained within a geographic boundary, preventing it from being replicated to regions outside the regulatory zone. Additionally, as the amount of generated data and regions increases, the amount of replicated data multiplies, leading to cascading failures.

App designers and operators require fine-grained control over the placement of data and how it's copied across topologies to comply with data sovereignty and privacy regulations. Akka provides replication filters to control what data is replicated and where it may be replicated. This applies to coarse-grained data (e.g., all User data to all regions) and to fine-grained data (e.g., a 'Bob' instance of an entity replicates across APAC and NA but not EMEA).



CRDT convergence

Akka leverages embedded CRDTs to merge state changes from the same data instance that is replicated across regions. CRDTs are fast, efficient, and guarantee that merge ordering is consistent. You will always converge. Akka supports single-writer entities where only one instance in one region is modifiable with all other replicated instances as read replicas.

Akka also supports multi-writer entities (Multi-Master Replication) where the same data instance can be written to simultaneously in two different regions. Developers can implement a merge function that programmatically instructs the CRDTs how to merge data that was modified by two parties at the same time, similar to how multiple editors work within Google Docs.

Eventually consistent

Akka applications anticipate and expect data to be eventually consistent. While local operations against your application's data are strongly consistent, that consistency is not provided to other regions as it would create a ceiling on scalability and availability.

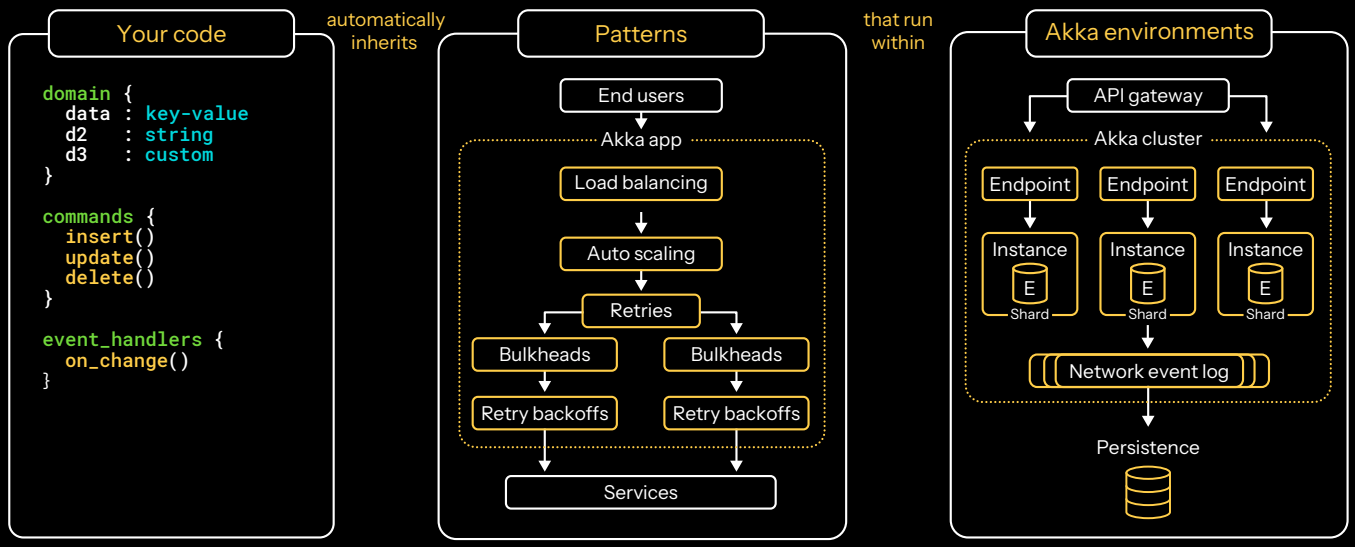
Strongly consistent cross-region transactions would cause local write latency to explode from less than 10ms to well over 300ms (the human eye can detect visual cues >13ms). This limitation is why systems built upon distributed transactions and Raft/Paxos consensus hit upper scale limitations and reduce availability as all parties are strongly coupled.

Akka applications are, by design, able to work with eventually consistent data, unencumbered by whether the changed data it's receiving was modified seconds or even days ago. When an application is eventually consistent by design, you can lose entire regions for extended periods, and the application will be able to seamlessly incorporate all data changes made before, during, and after the region failure.

Load shedding

Akka apps cluster themselves from within to provide elasticity and redundancy to deterministically execute actions no matter what demand is placed upon the application. Akka embeds many techniques to shed load in order to handle failures or avoid overwhelming the system.

This includes backpressure to slow upstream systems with overflow strategies, circuit breakers to reject requests from failing systems, throttling to limit processing rates on custom logic, dynamically balancing load across clustered nodes, supervision to restart or stop overwhelmed service instances, adaptive control for rebalancing data and load as compute alters, and data passivation on infrequently accessed objects to free memory.



Multi-region deployment
Akka applications are developed, built, tested, and packaged offline. When ready, applications can be deployed once, with Akka ensuring that your application's assets are made available within all regions.

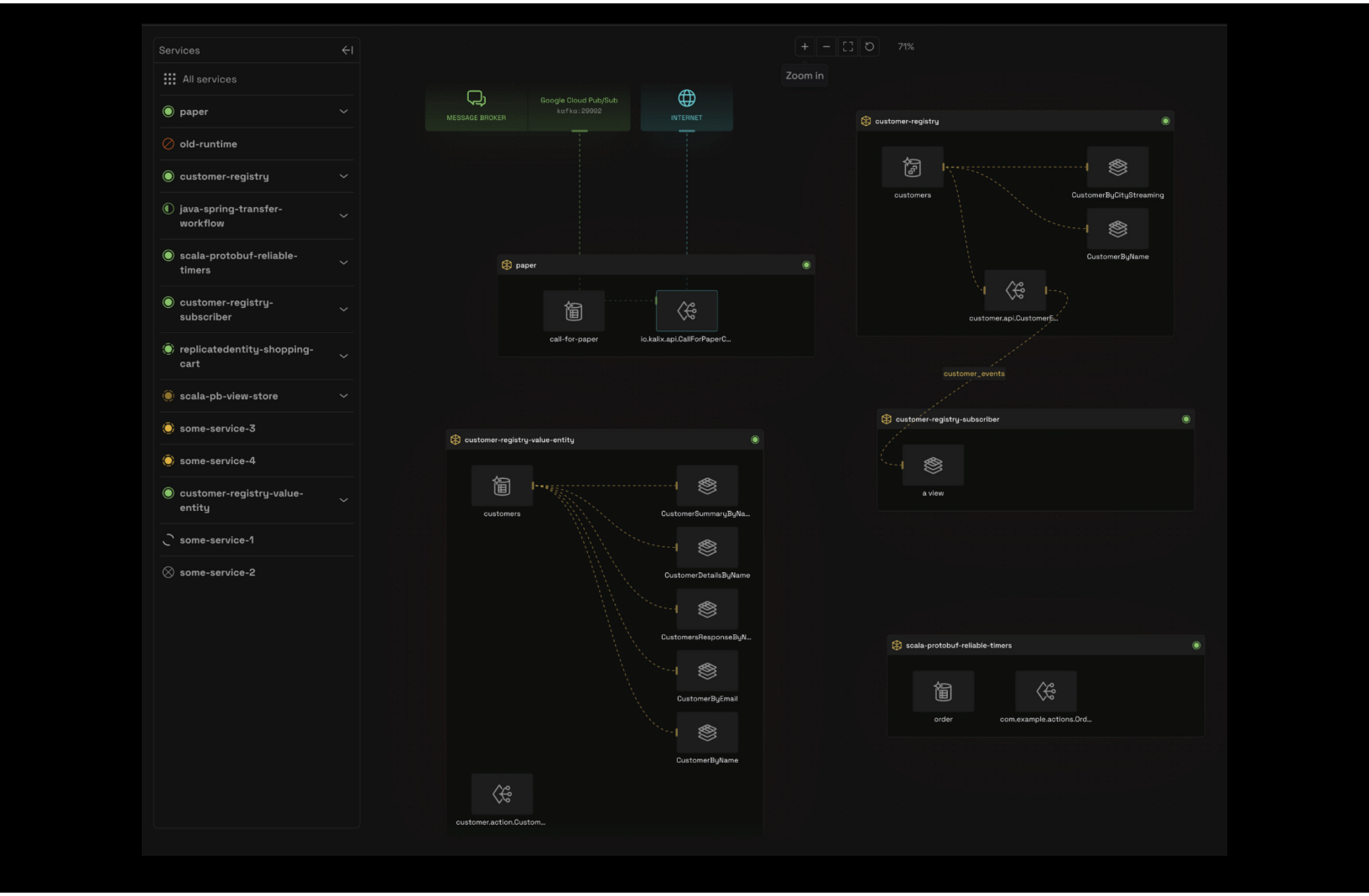
Akka coordinates the service activation in each region, creating region-specific routes for APIs and endpoints. Akka applications are made available within a registry, which could be globally served or embedded within each region.

Secrets, keys, and certificates are centrally managed and pushed into each region with regular rotation for certificates and secrets in control of sensitive data.

Rolling no-downtime regional updates
Many stateful systems block end-user traffic during an upgrade or migration cycle. Akka apps perform no-downtime rolling upgrades within a single region. Application instances are elastic, enabling new versions to form new clusters to serve existing end-user traffic while old instances are slowly discontinued.

With all state changes and communications transported as decoupled events, new application versions or data schema changes are automatically interoperable with older versions. With Akka, you can execute new version updates in all regions simultaneously or within a single region to create regional blue-green upgrades.

Global observability
Akka collects logs, metrics, and traces for running applications within each region and merges them into a central observability view for each application, service, and component. Operations can evaluate per-region SLOs such as availability, latency, errors, and events while also measuring a global availability SLA to quantify system availability. Observability data for each region can be exported to third-party observability solutions.



Global hostnames

The same application running in two regions will expose its endpoints within each region, creating multiple routes for end users to access the application. Each service within each region will have a region-specific hostname and IP address, pushing responsibility out to clients to detect available regions and to select the optimal location for their requirements, whether they are latency, geo-location, or functionally driven.

Akka provides access to global and custom hostnames. Global hostnames (i.e., random-host-id.akka.service) are Akka-specific hostnames generated for a single app as a single entry point to all regions. Global clients can reference this singular hostname and access all regions, deferring the selection of the region to the behavior of how Akka registers within different DNS name servers.

Custom hostnames (i.e., service.mycompany.com) are owned and controlled by our customers who register all of the Akka regions servicing the domain with our customer's DNS name server providing routing and selection logic.

DDoS resistance

You can deploy as many Akka regions as you want, operating within different cloud providers or private data centers under your control. Regions can be directly exposed as routes or as a shadow region where it receives replicated data with endpoints whose routes are not Internet-accessible.

Operating across hyperscalers incorporates the security best practices of multiple cyber defense clouds, which route traffic across multiple data centers, drop malicious traffic at their network edge, and leverage machine learning to adapt to the attack. Shadow regions are hot standbys that can be activated as an active region with routes unknown to attackers.

Additionally, the use of global hostnames and custom hostnames creates security through obscurity, creating a layer of abstraction that poses challenges for attackers to discover and locate the full list of regions that serve an application.

Trust before profit
One risk to any multi-region system is a failure of trust from third-party services, underlying infrastructure, or black-box managed services. This failure could result from systemic errors introduced into an underlying platform or from unintentional and malicious human error.

The xz utility backdoor was terrifying and exposed the risks underpinning the global software supply chain.

Akka's commitment to security and compliance is paramount, leading to attestation or certification of 19 compliance standards and a plan to implement an additional 1,000 controls throughout 2025.

Resilience guarantee
Akka apps have been trusted by F500 and startup disruptors for 15 years. Based upon the Reactive Manifesto and Principles, Akka apps are resilient by design, enabling them to recover an application's state from any hardware or network failure, regardless of the length of the disruption.

Akka is backed by decades of research. Akka's track record of stability is unmatched, with Akka never experiencing a Sev1 outage that led to data loss in 15 years. We embrace this confidence and extend these protections to our customers with legal and financial protections: Akka indemnifies against losses caused by an Akka App becoming unreliable.

Akka lowers the cost of nines

Akka changes the economics of adding nines:

Development
Apps built with Akka are nine-nines ready and deployed independently of your operating goals. Akka has a simple SDK with six easy-to-learn components that enable the full spectrum of enterprise systems.

If an Akka app builds, runs, and passes tests with our offline test kit, it will be consistent when deployed into an operating environment.

Akka apps can operate within a single region or many regions simultaneously. The offline development model, coupled with Akka's rolling no-downtime updates, creates the potential for tremendous velocity improvements as application changes can be made quickly and nearly instantly observable in a multi-region production environment.

Operations
Akka apps manage their own infrastructure to consume only the resources required to achieve their SLA. New regions with redundant infrastructure can be operated for less than $2K / month.

Akka applications can replicate across all or a select set of the regions you provision to optimize the allocated infrastructure spend. New regions can be added in hours - whether in the same cloud or different clouds - and existing Akka applications can replicate into (or migrate) into the new region.

When combined with replication filters and data archiving, you can add many more nines of availability with less-than-duplicative infrastructure increases for each new region. Further, because regions can operate in different cloud providers and Akka apps can migrate between regions, your operations teams gain bargaining power with your cloud provider during your annual spend commit discussions by having applications freed from cloud lock-in.

Availability	Akka's economics	Dev cost	Ops cost
99%	Akka single node, single region, and serverless deployments are elastic, agile, and resilient to millions of TPS.	$	$
99.9%	Akka apps can be simultaneously deployed in one or more regions, with all the application's data replicated between them.	$	$$
99.99%	Akka apps are decoupled, event-driven services. Traffic shedding and failover automation are contained within. All clusters are active-active. Ops integrates with their observability and SIEM tools for monitoring, metrics, traces, and logs.	$	$$$
99.999%	New regions can be added in hours in self-hosted environments or in different cloud providers. Upgrades and new application versions are rolling no-downtime updates. DDoS resistance through regional obscurity, traffic shaping and steering, and global hostnames.	$	$$$
99.9999%	Akka apps expect failure and are guaranteed to recover from any hardware or network disruption. GitOps-ready systems enable multi-region chaos engineering and continuous deployment.	$$	$$$$

Your Akka TCO is a fraction of the alternative

For ~ USD 300,000 / year, you can operate an application that exceeds 99.9999% availability while sustaining more than 1000 Transactions per Second (TPS), a rate of activity more than what is required by 99.9% of APIs. This amount includes the subscription paid to Akka and the infrastructure needed!

With Akka 3, you can *build and run* your applications across multiple regions. Akka provides three types of operating environments: a Serverless environment, where Akka provides the infrastructure; a Bring Your Own Cloud (BYOC) environment, where Akka remotely manages a private region in your cloud VPC; and a Self-Hosted environment, where you provide the infrastructure and management oversight.

Consider a high-traffic service that needs 99.9999% availability across three regions. When considering the total cost of ownership, the single-region infrastructure required is determined by the application's required performance.

Single-region requests / hour	1,000,000
Single-region expected avg. TPS	278 TPS
Three-region expected avg. TPS	833 TPS
Single-region spiked TPS capacity	2,000 TPS
p(99) end-user read latency	8ms
p(99) end-user write latency	14ms
Data cached in-memory to lower latency	96 GB
Stateful data generated / hour	1.5 GB
What % of data replicated to other regions?	100%

This application profile determines the infrastructure required to deliver the targeted performance:

- Higher total aggregate TPS requires more app instances.
- Higher TPS spikes require a bigger persistence store.
- Higher generated data requires more egress and storage.

With this configuration, the total cost to license Akka and purchase the cloud infrastructure required to deliver the app would be:

Akka TCO - 1K Sustained TPS - BYOC

	1 Region cost	3 Region cost
Paid to Akka	$5K	$15K
Paid to your cloud provider: orchestration, registry, observability, proxies, storage, egress	$3.3K	$10K
Monthly TCO	**$8.3K**	**$25K**

This configuration has the capacity to deliver 2500 TPS for your Akka app, much higher than the targeted 1K sustained configuration. And, the price can be lowered as replication filters reduce egress and storage costs.

Most customers choose a BYOC deployment, which gives them data sovereignty and cloud account control. In a BYOC scenario, customers pay for the infrastructure consumed, apart from the licensing for Akka's operating environments, which manage the Akka app and infrastructure. Akka's fees are consumption-based and tied to the consumption of your services. Services with lower traffic will be charged less than those with higher traffic.

What's wild is to compare Akka's TCO against the licensing fees of replicated databases.

Not only must you purchase the redundant infrastructure, the subscription fees for replicating a transactional SQL database will often be 3-5x more than the TCO of an Akka application.

The distributed SQL vendors that you may consider will not manage nor operate the application and certainly won't offer a resilience guarantee for the application itself. Only when you let the application act as its own in-memory, durable database can you observe a higher resilience guarantee, higher system SLA, and lower costs due to its efficient execution model.

Learn more - start your journey of nines

Akka has made more than 100,000 systems elastic, agile, and resilient. Akka is downloaded 13M times monthly, and every day, more than two billion people touch an app that is powered by Akka.

Many of Akka's customers have strived and achieved many nines of availability:

1. CERN: "Akka makes NXCALS data acquisition extremely resilient to failure. Akka's self-healing capabilities fully recreates itself without loss of data during broader infrastructure maintenance.
2. Norwegian Cruise Lines: "In 12 months with Akka, NCL completed over 65 code deployments with no major issues or rollbacks."
3. Icon Solutions: "Akka handles failures and restarts, ensuring that the system can recover immediately, remain stable, and is always available."
4. Judopay: "A payments system must offer 100% certainty, and the self-healing capabilities of Akka give us exactly what we need."

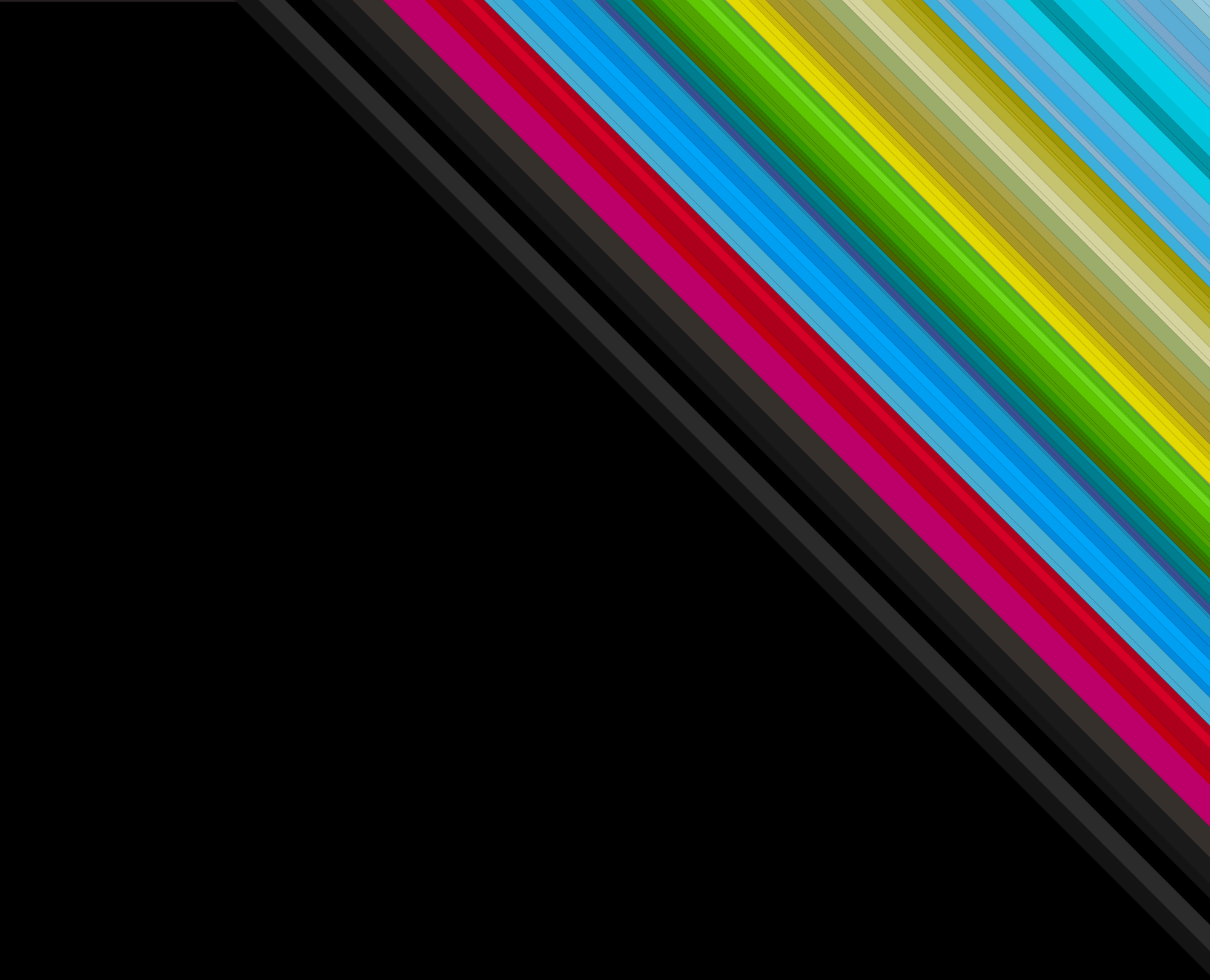
Next steps

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