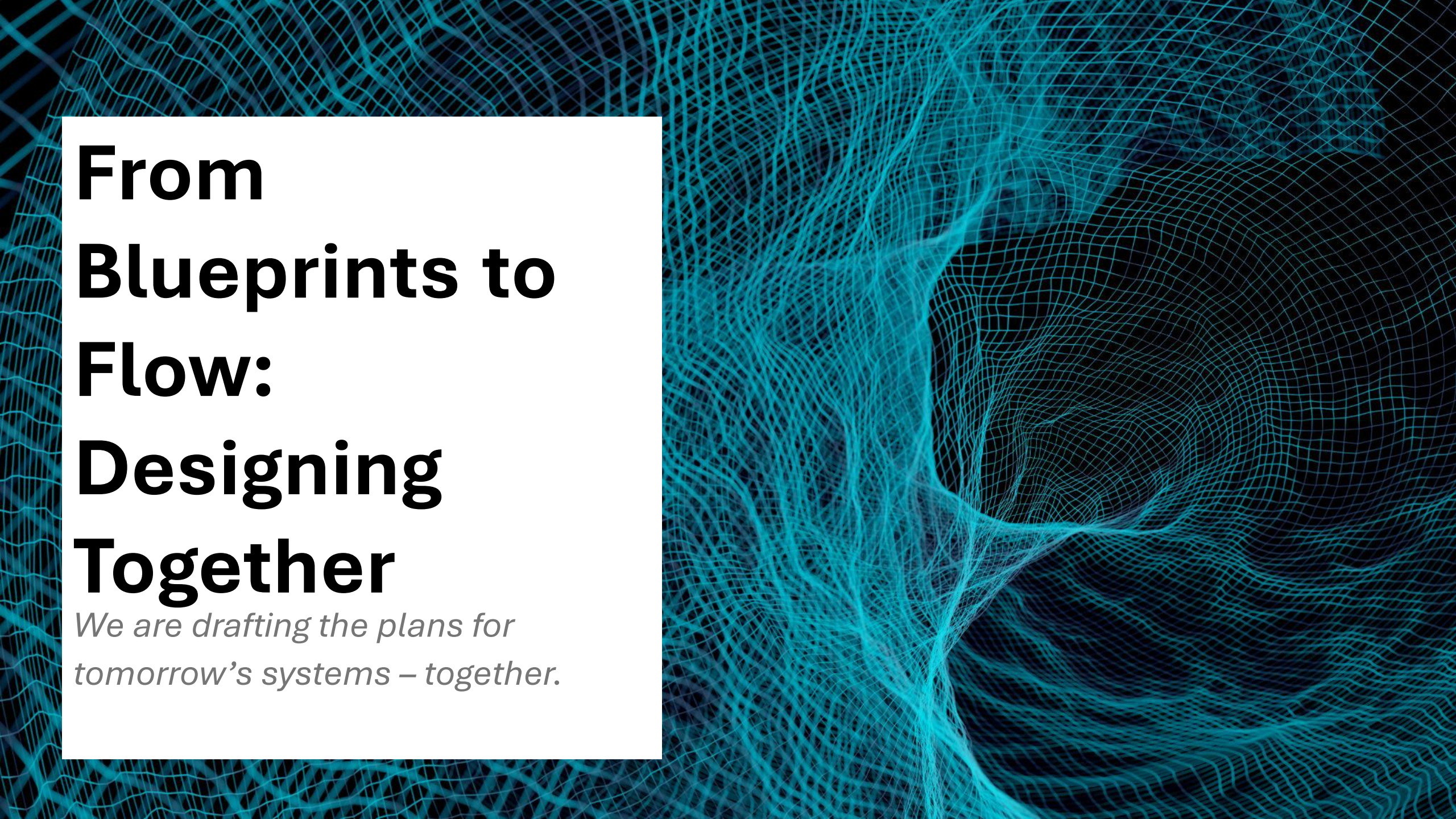
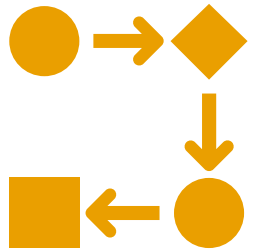


From Blueprints to Flow: Designing Together

We are drafting the plans for tomorrow’s systems – together.

Introduction to the Blueprint and Flow Motif

Blueprint Approach



Flow Motif

Scalable Cloud Solutions

You are the Co-Architects

Shared Ownership

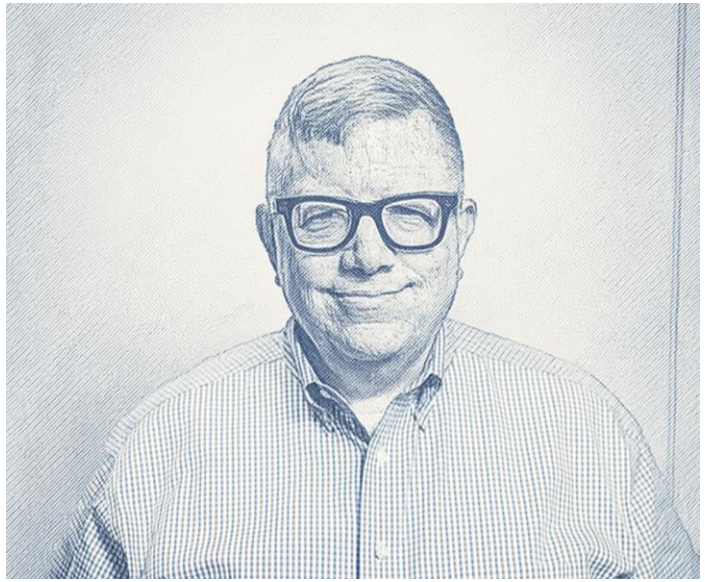
Active Participation

Real-World Relevance

Meet Your Lead Architects

Martine Dowden
Andromeda Galactic Solutions
CTO, UX/UI Design & Developer

Michael Dowden
Andromeda Galactic Solutions
Founder & Technology Leader



Chad Green
Jasper Engines & Transmissions
Event-Driven, Cloud-Native
Systems Architect

Meet Your Lead Architects

Martine Dowden

Wednesday – 14:30: Writing a dynamic API endpoint generator: Consuming arbitrary data against a user-defined layered schema

Michael Dowden

Tuesday – 11:30: Coding Serverless Functions

Wednesday – 09:00: Real-time Serverless APIs

Chad Green

Tuesday – 10:15: Unlock the Digital Gateway: Smarter API Discovery & Governance with Azure

Tuesday – 15:45: Scaling APIs Like the Heart of Berlin: Building Resilient APIs in Azure

Wednesday – 13:30: Ask Me Anything (AMA)

Our Blueprint for the Day

Break

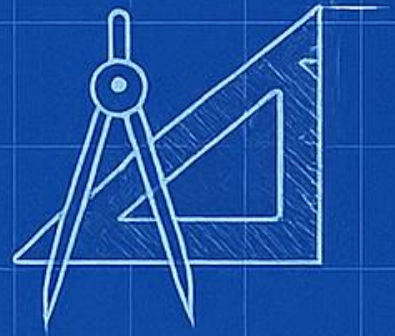
Lunch

Break

Break

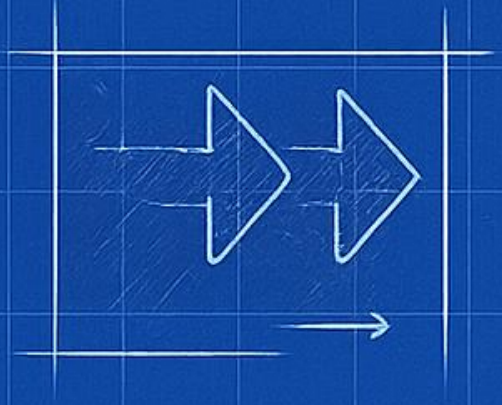
Patterns & Foundations

Phase 1



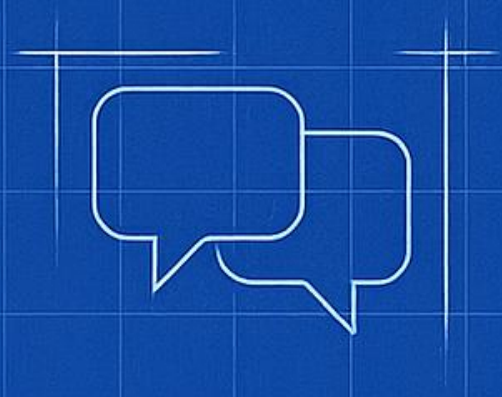
Design Studio I

Phase 2



Design Studio II

Phase 3

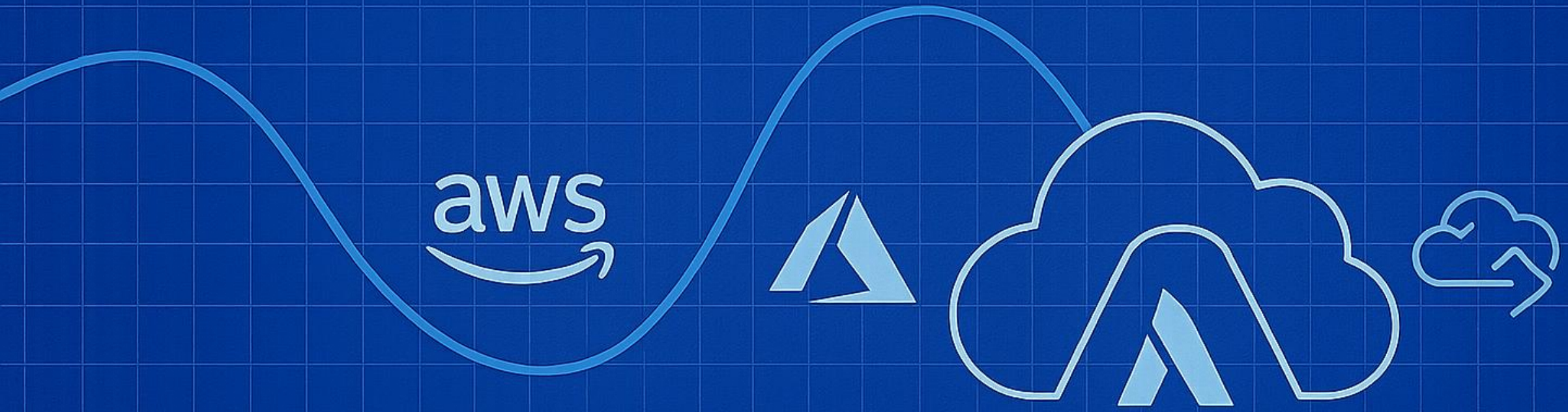


Panel O&A

Phase 4

Who has a serverless project currently running in production?

Which cloud provider(s) are you using?



Which language(s) are you building in?

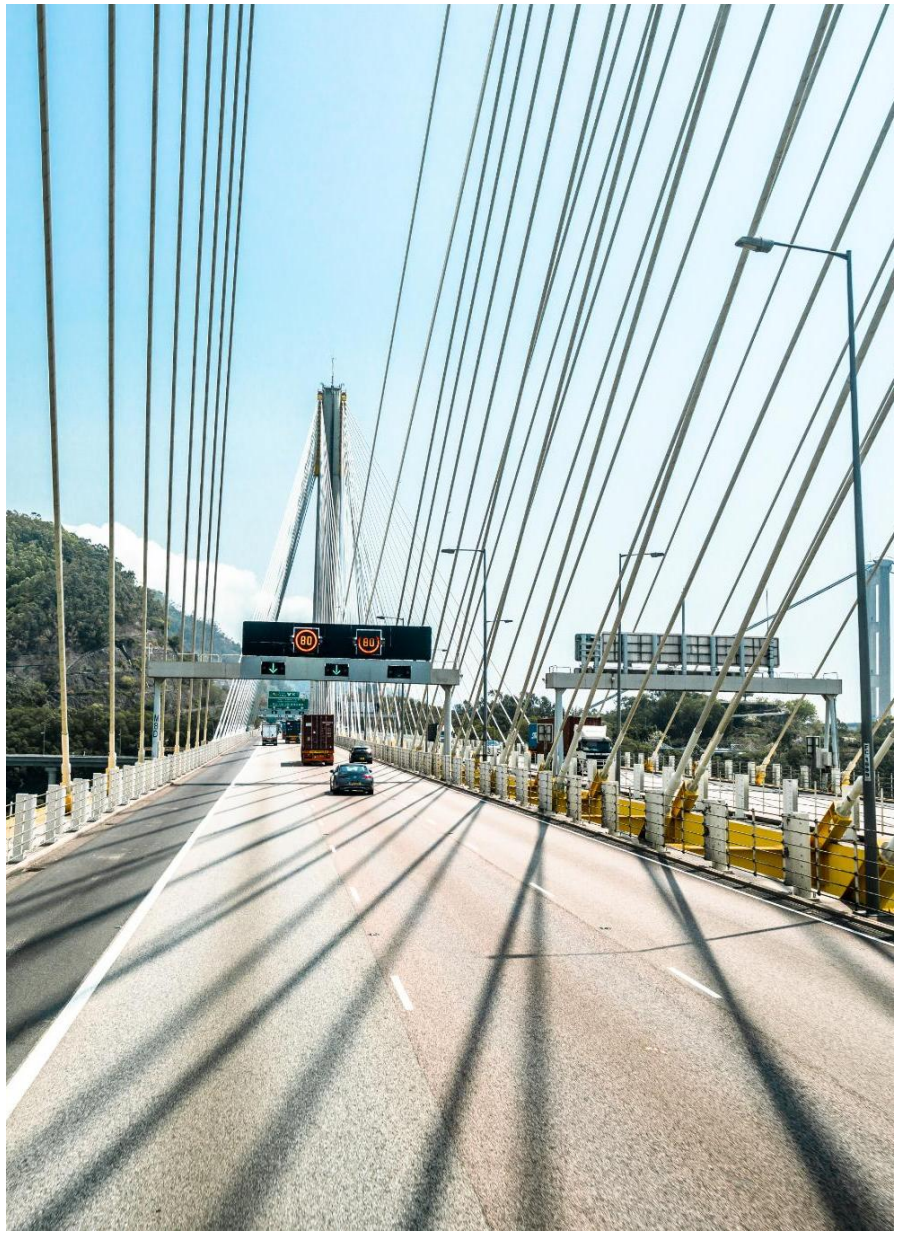
Why Serverless?

Focusing on Business Logic, Not Infrastructure

Developer Productivity

Provider-Managed Infrastructure

Accelerated Delivery



Elastic, Event-Driven Scaling

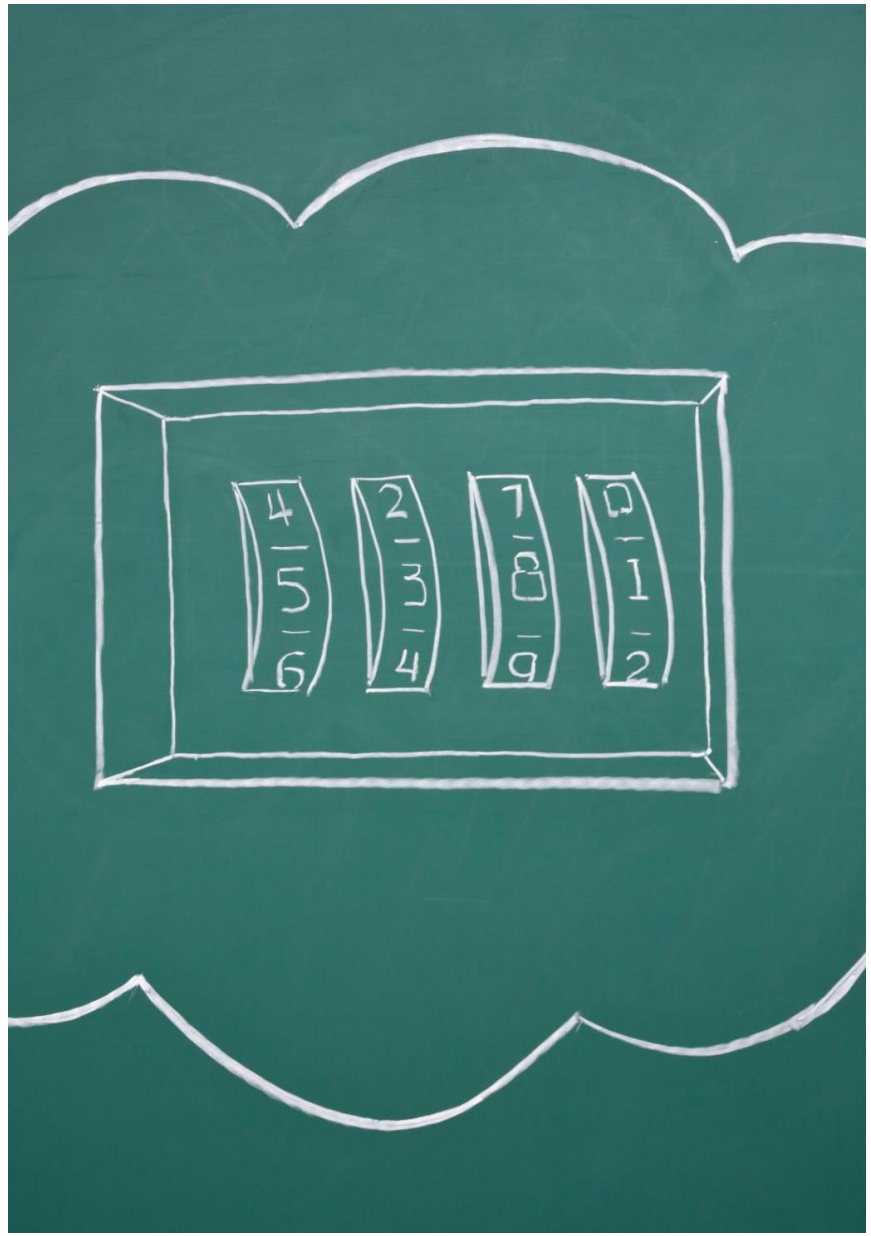
- **Event-Driven Triggers**
- **Instant Scaling Capability**
- **Ideal for Variable Workloads**

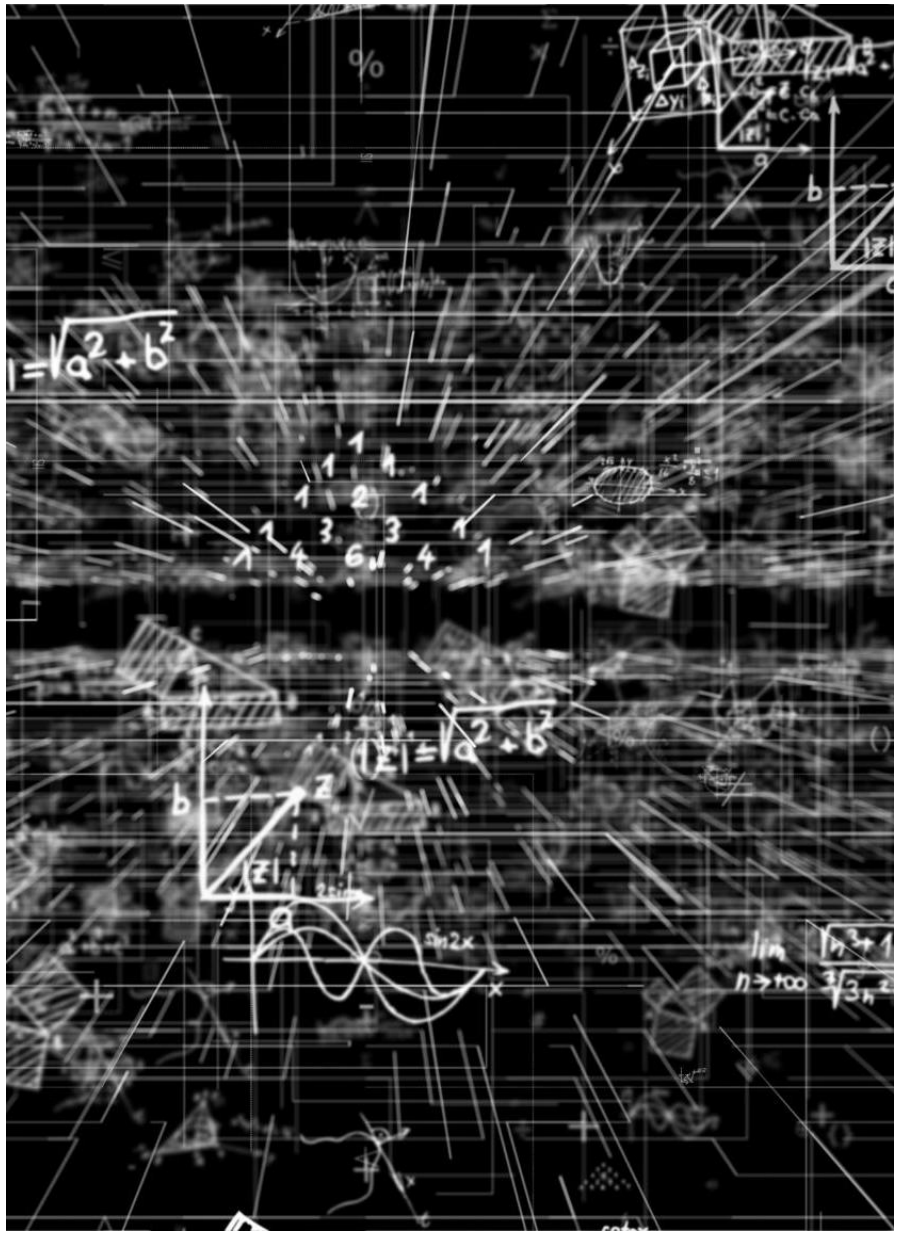
Cost Efficiency

Pay Only for Usage

Elimination of Idle Costs

Budget Management





Faster Innovation Cycles

- **Remove Infrastructure Barriers**
- **Accelerated Iteration & Experimentation**
- **Quicker Product Development**

Built for Modern Architectures

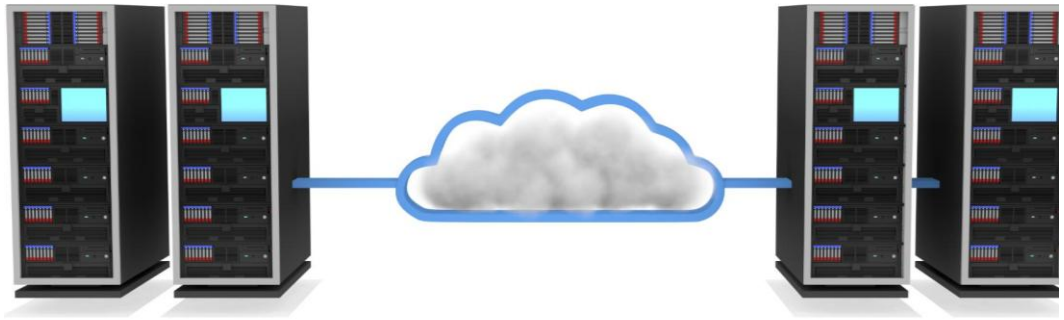
Microservices Ready

Event-Driven by Design

DevOps Integration

Why Now?

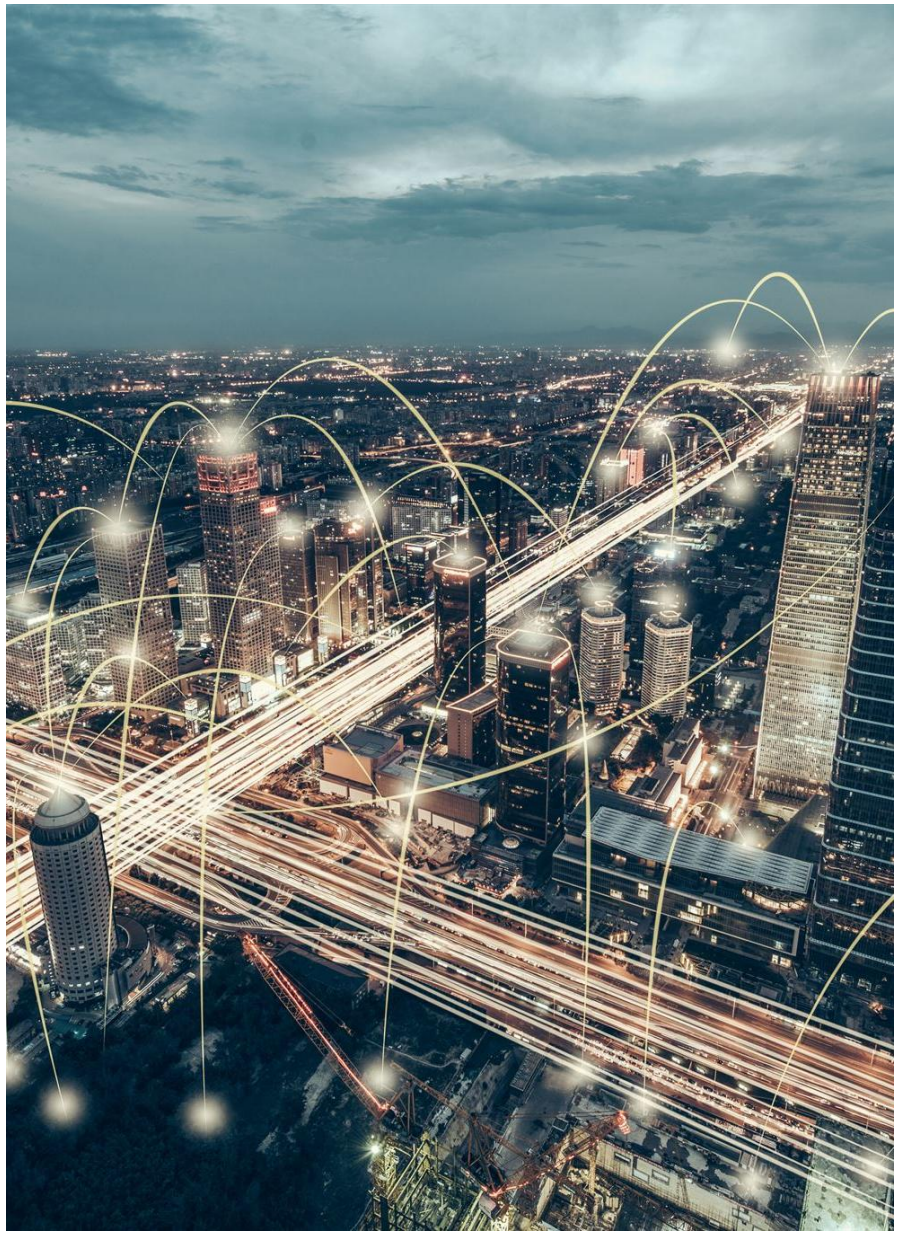
Maturity of Platforms



Extended Runtimes

Advanced Monitoring

Richer Integrations



Mainstream Adoption

- APIs at Scale
- ML & Data Pipelines
- Enterprise Integration

Economic Pressures

Economic Uncertainty

Cost-Efficient Solutions

Serverless Optimization

Developer Productivity Demands

- Faster Time-to-Market
- Focus on Features
- Accelerated Velocity

Conclusion

Operational Simplicity

Serverless computing reduces complexity by removing infrastructure management tasks, enabling focus on development.

Scalability Benefits

Serverless platforms automatically scale applications based on demand without manual intervention.

Cost Efficiency

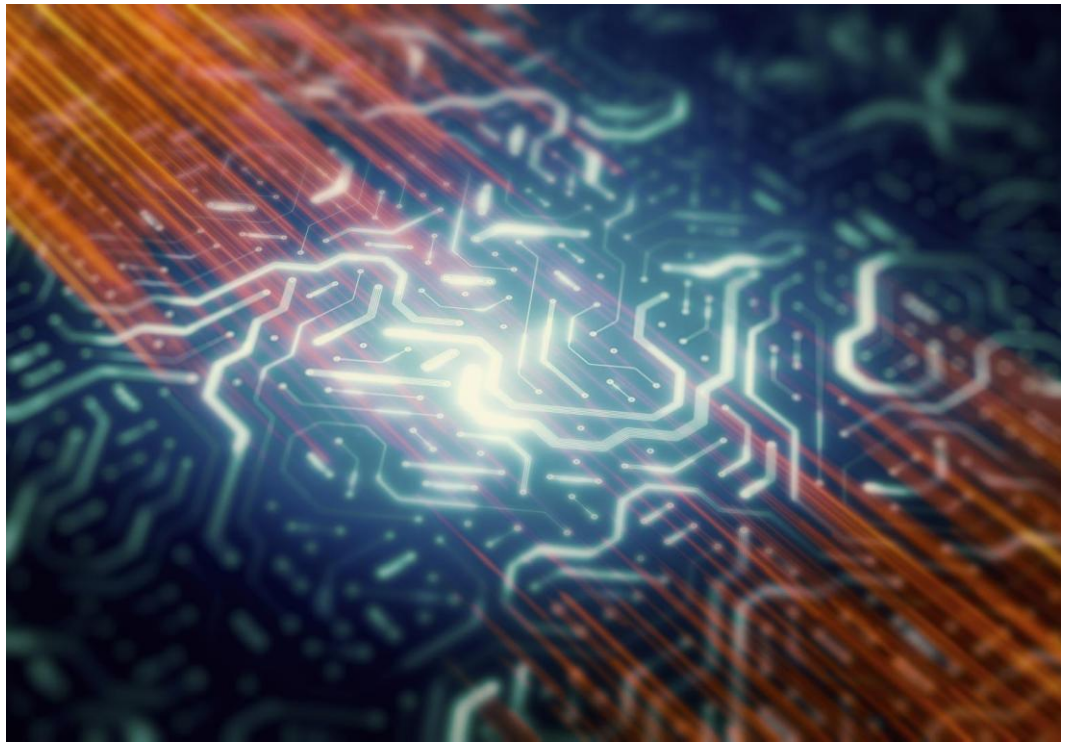
Pay-as-you-go pricing models reduce costs by charging only for actual resource usage.

Faster Innovation

Developers can rapidly build and deploy features, accelerating time to market with serverless architectures.

Serverless Architecture Styles

Event-Driven Architectures



Serverless Function Triggers

In event-driven architectures, functions respond to triggers such as HTTP requests, database changes, or file uploads.

Scalable Reactive Systems

Event-driven models support scalable and reactive systems by leveraging flexible and elastic pipelines.

Circuit Diagram Metaphor

Events are visualized as sparks in a circuit diagram, igniting flows within the architecture.

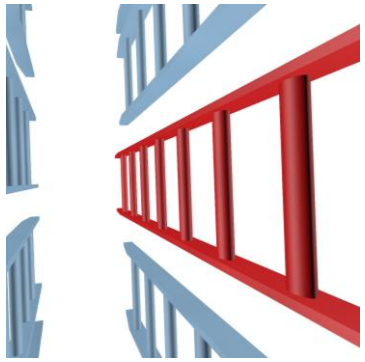
Avoiding Anti-Patterns

Avoid one function handling too many event types to maintain clarity and ease of maintenance.

Microservices with FaaS

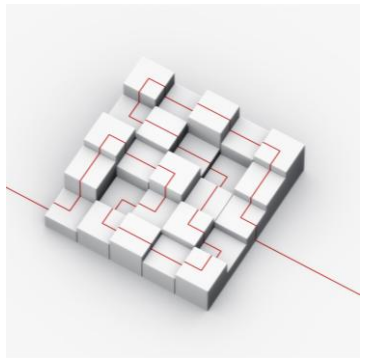
Discrete Single-Responsibility Functions

Microservices built as single, independent functions enable clear modular division and responsibility separation.



Independent Scaling and Modularity

FaaS supports independent scaling of functions, similar to rooms in a building connected by API gateways.



Mini-Monolith Anti-Pattern

Avoid sharing excessive state or logic between functions to prevent the mini-monolith anti-pattern.

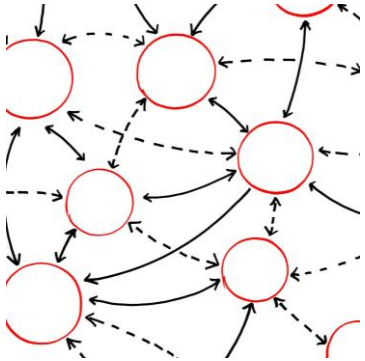
Orchestration and Workflow Patterns

Serverless Orchestration Tools

Tools like Step Functions and Durable Functions manage workflows, ideal for long-running and auditable processes.

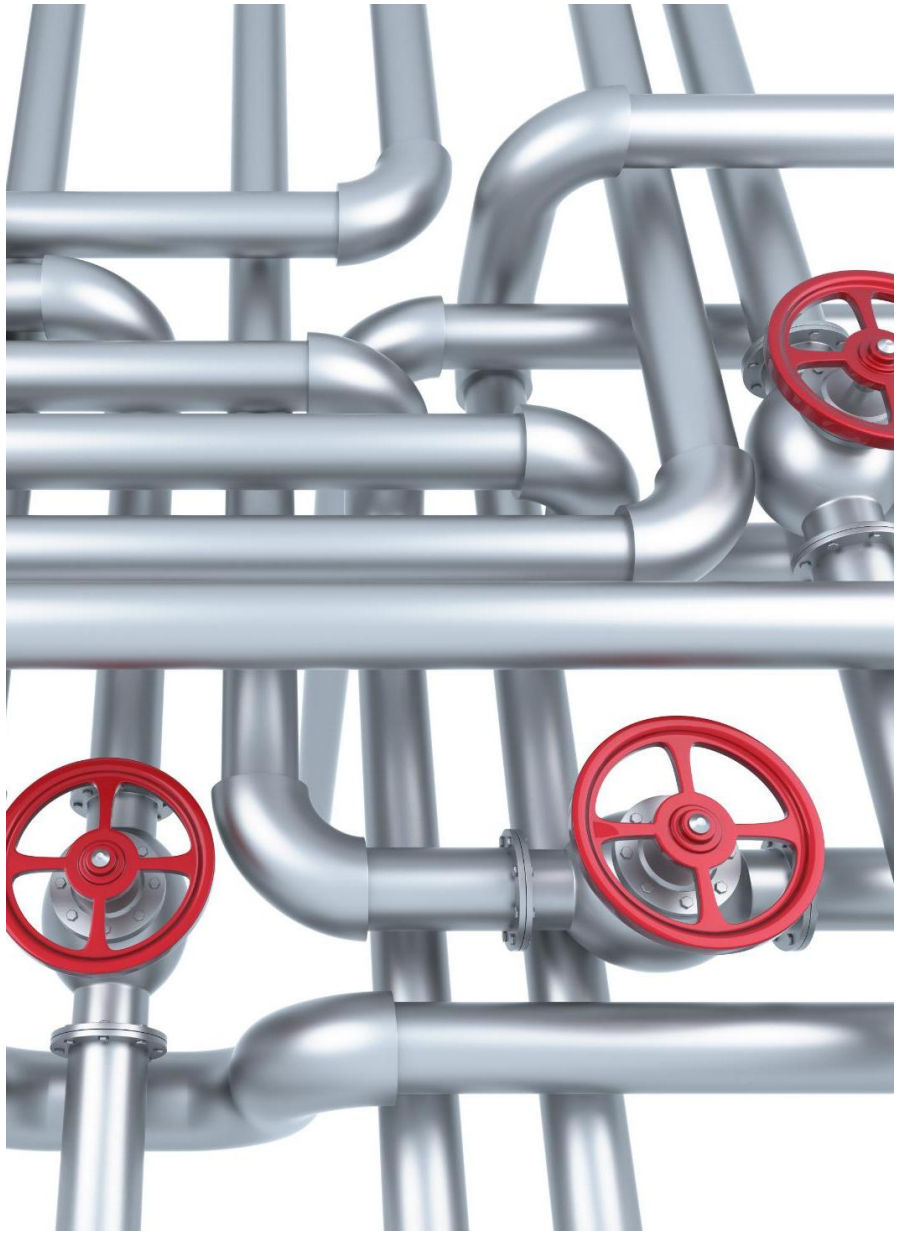
Control Tower Metaphor

The control tower metaphor illustrates central coordination and management of multiple subsystems via a master plan.



Avoiding Unnecessary Complexity

Orchestration should be avoided for simple tasks to prevent overhead and maintain clear system logic.



Data Processing Pipelines

Serverless Pipeline Concept

Serverless pipelines chain functions to process ETL, analytics, or IoT data efficiently in stages.

Real-Time Applications

This pattern supports real-time dashboards and dynamic data transformation for timely insights.

Limitations of Over-Chaining

Excessive chaining leads to latency, debugging challenges, and fragile, hard-to-maintain pipelines.

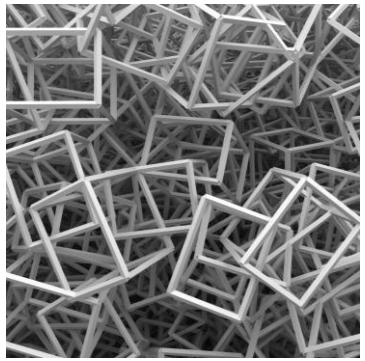
Hybrid / Polyglot Serverless

Blending Serverless and Legacy

Hybrid architectures combine serverless functions with containers, VMs, and legacy systems to support flexible, mixed workloads efficiently.

Workloads and Task Matching

Use serverless functions for bursty, stateless tasks and containers or VMs for long-running, stateful jobs within the same architecture.



Avoid Architectural Fragmentation

Uncontrolled serverless adoption without proper integration can lead to fragmented, inefficient architectures. Strategic planning ensures cohesion.

Core Serverless Patterns

Foundational Request/Response

Core Serverless Pattern

The request/response pattern forms the foundation of serverless architectures with API Gateway and Function integration.

Direct Client Interaction

It handles client requests directly and returns immediate responses, serving as the system’s front door.

Simple Event Flow

The event flow is straightforward: request enters, function executes, then response is sent back.

API Gateway Pattern

Event-Driven Core Patterns

Decoupling Producers and Consumers

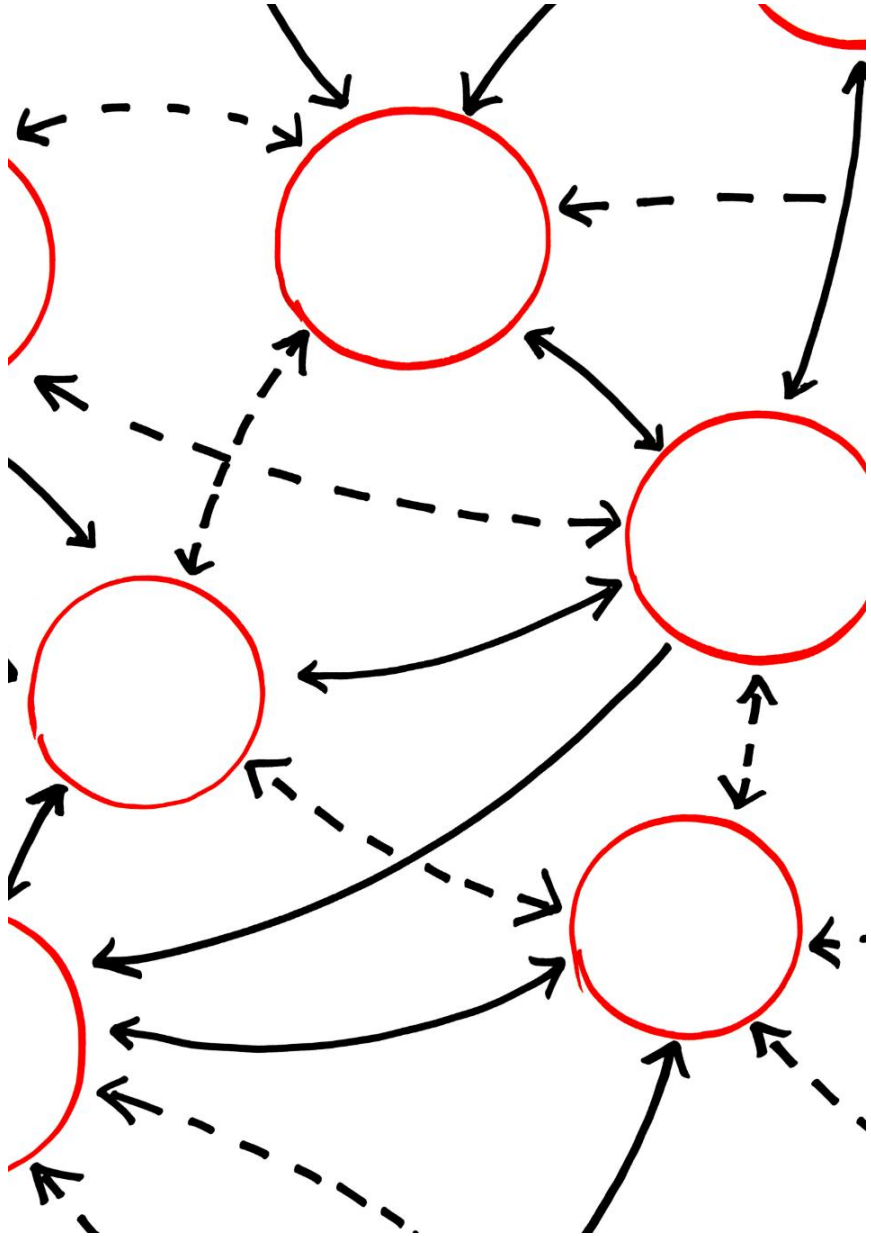
Event-driven processing separates producers and consumers, improving system flexibility and scalability.

Pub/Sub Pattern

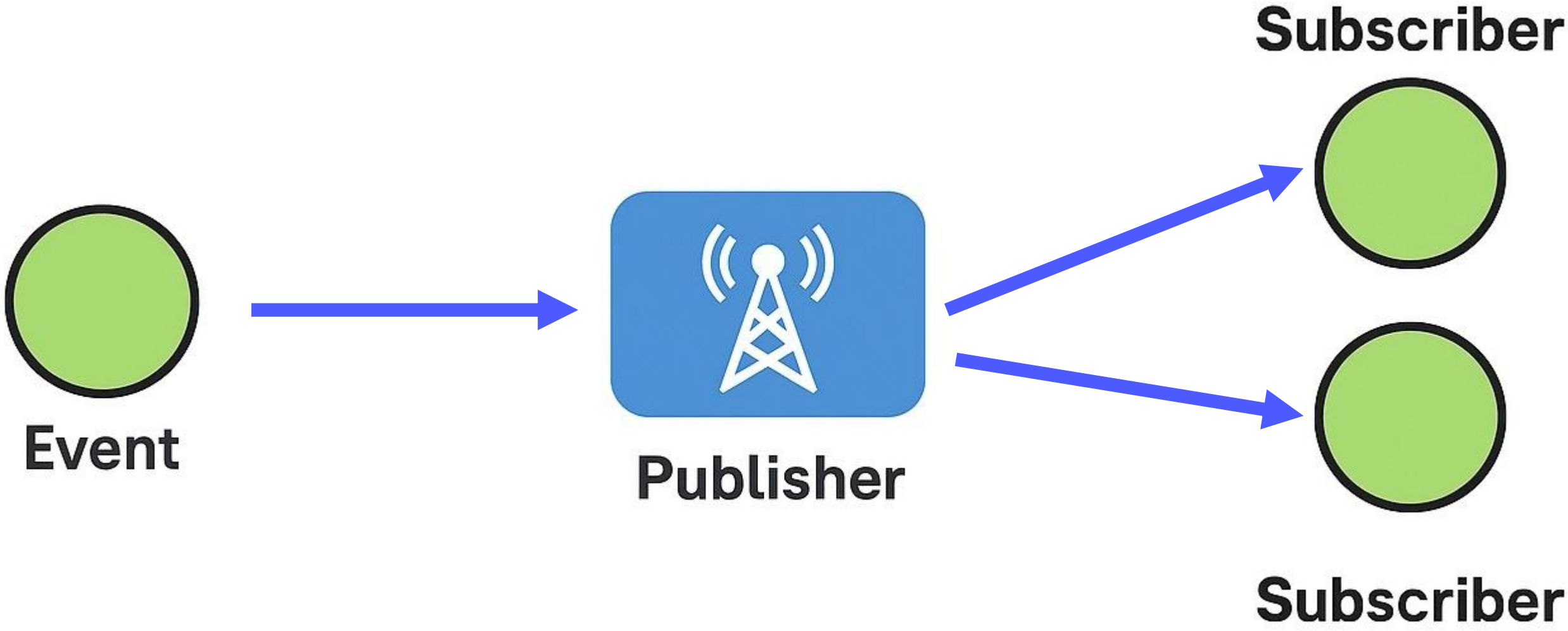
Pub/Sub pattern acts like a signal relay, distributing events to multiple subscribers efficiently.

Fan-Out/Fan-In Pattern

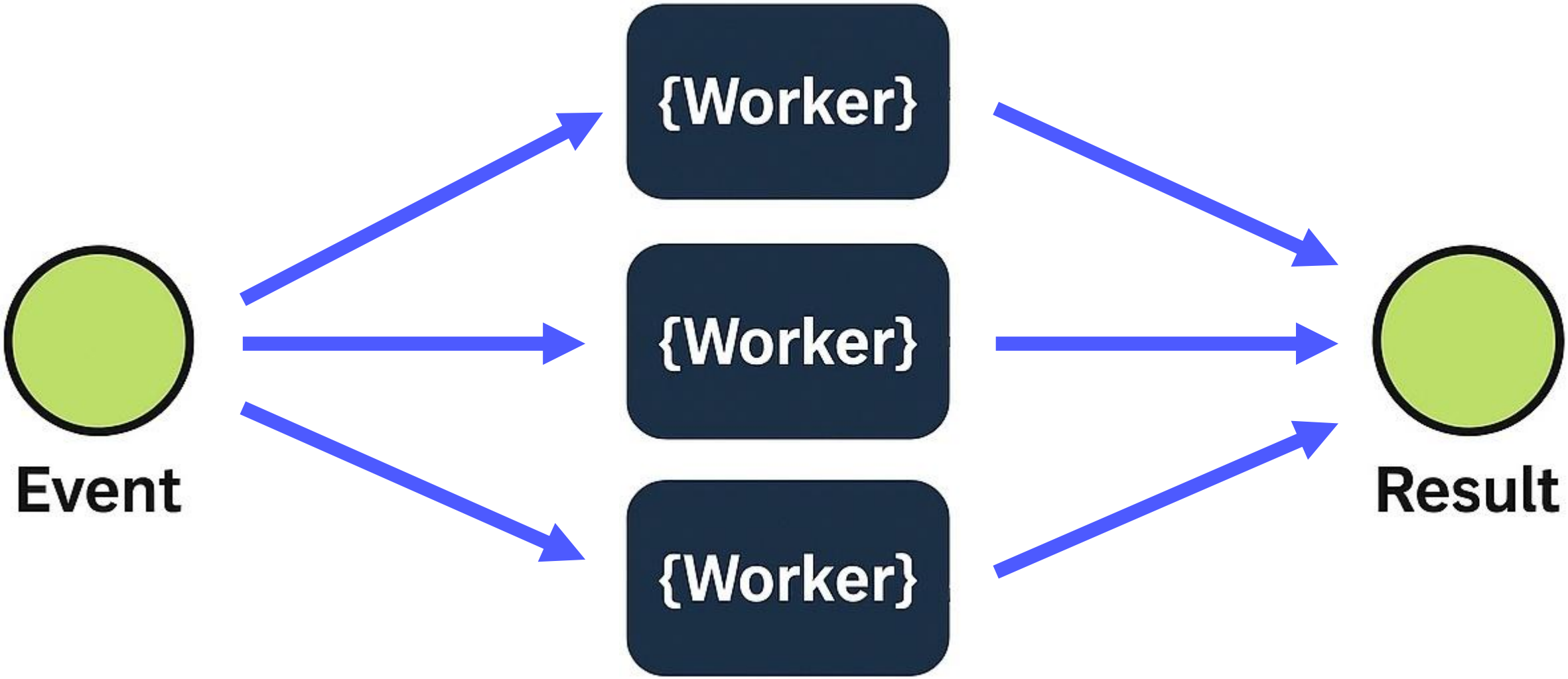
Fan-Out/Fan-In pattern enables branching event streams and parallel workers aggregating results.



Publish/Subscribe (Pub/Sub)



Fan-Out / Fan-In



Migration & Evolution Strategies

Strangler Fig Pattern

This pattern incrementally migrates legacy systems by gradually wrapping old components with new interfaces.

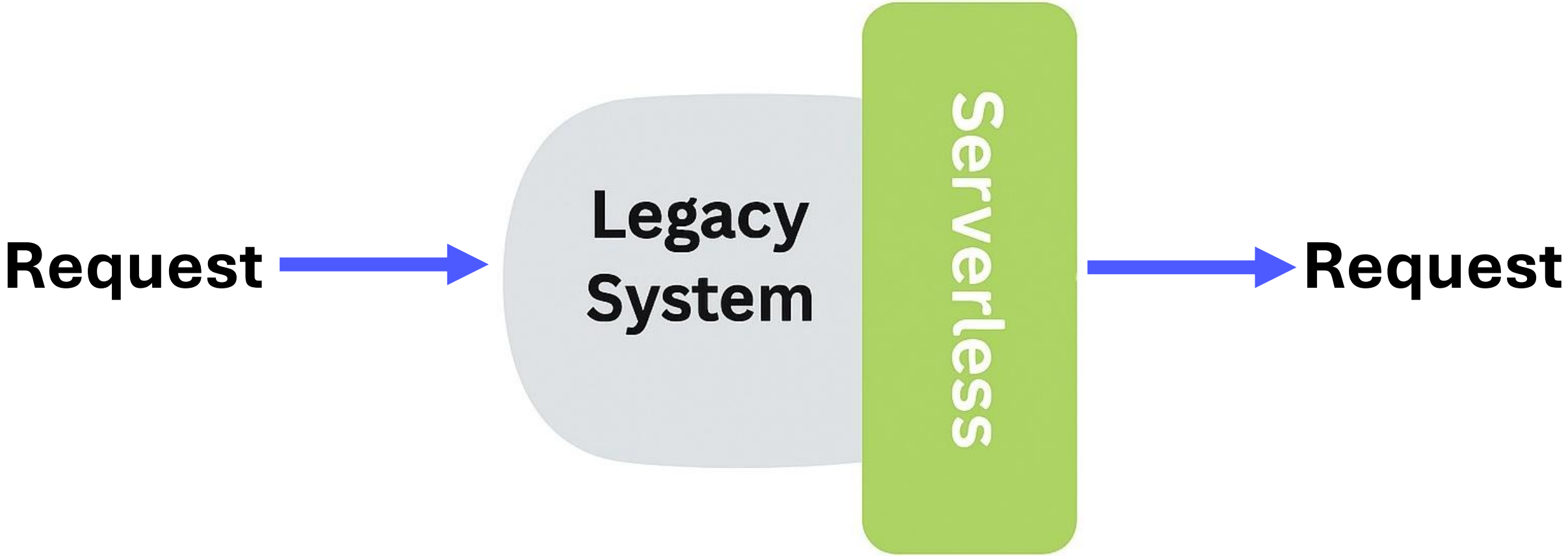
Incremental Migration

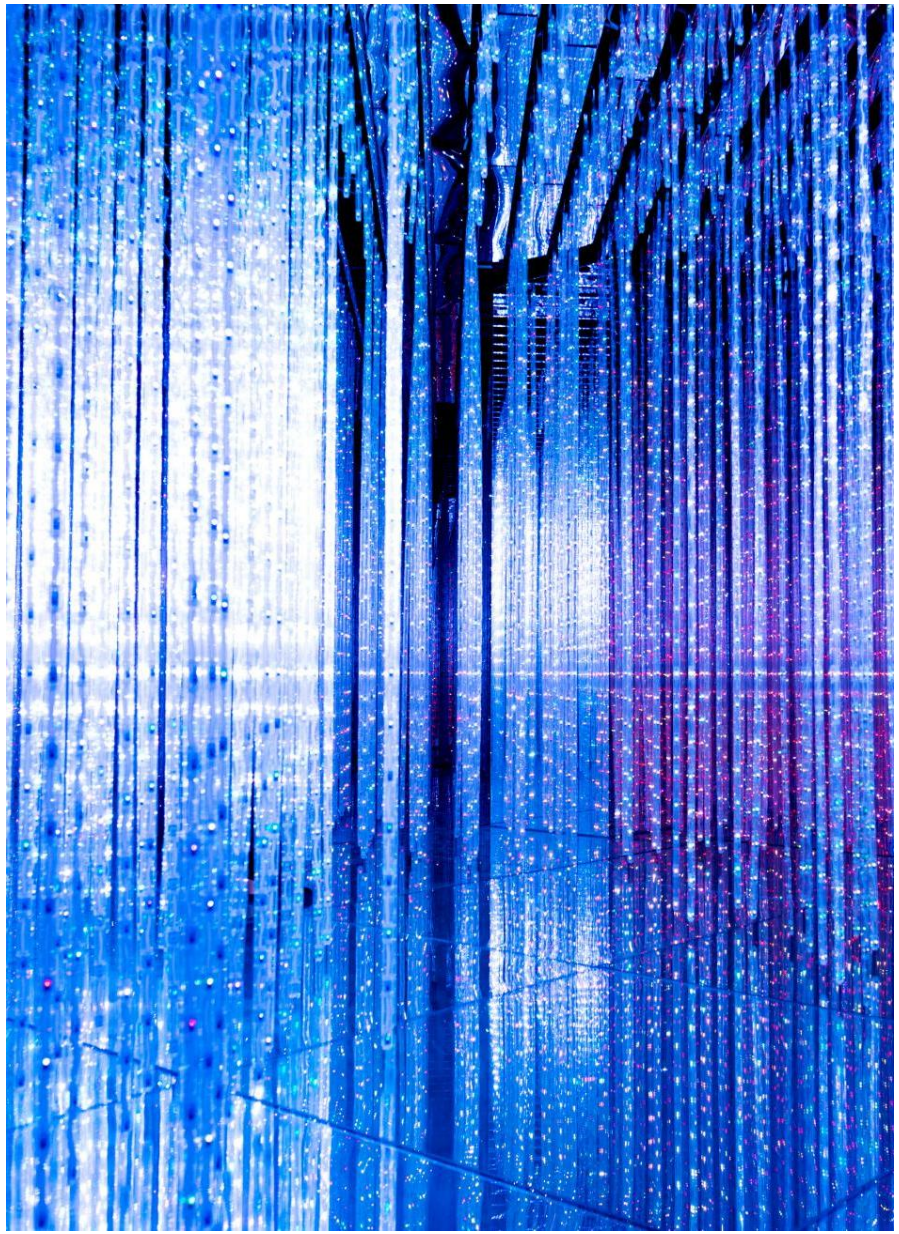
Incremental wrapping allows smooth transition and reduces risks during system modernization.

Event Flow Rerouting

Event flow diagrams illustrate traffic rerouting over time for controlled and safe modernization.

Strangler Fig





Data-Centric Patterns

Event Sourcing

Event Sourcing treats every data change as an immutable event, similar to a ledger or revision log.

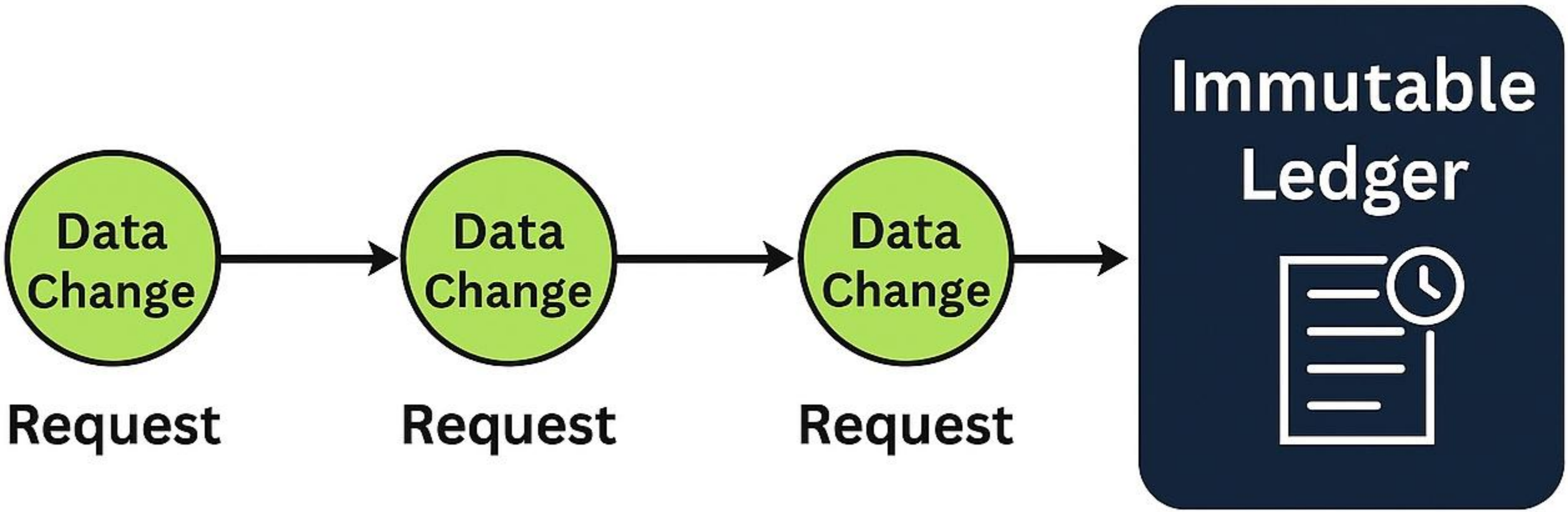
CQRS Architecture

CQRS separates command and query pipelines to improve system scalability and performance.

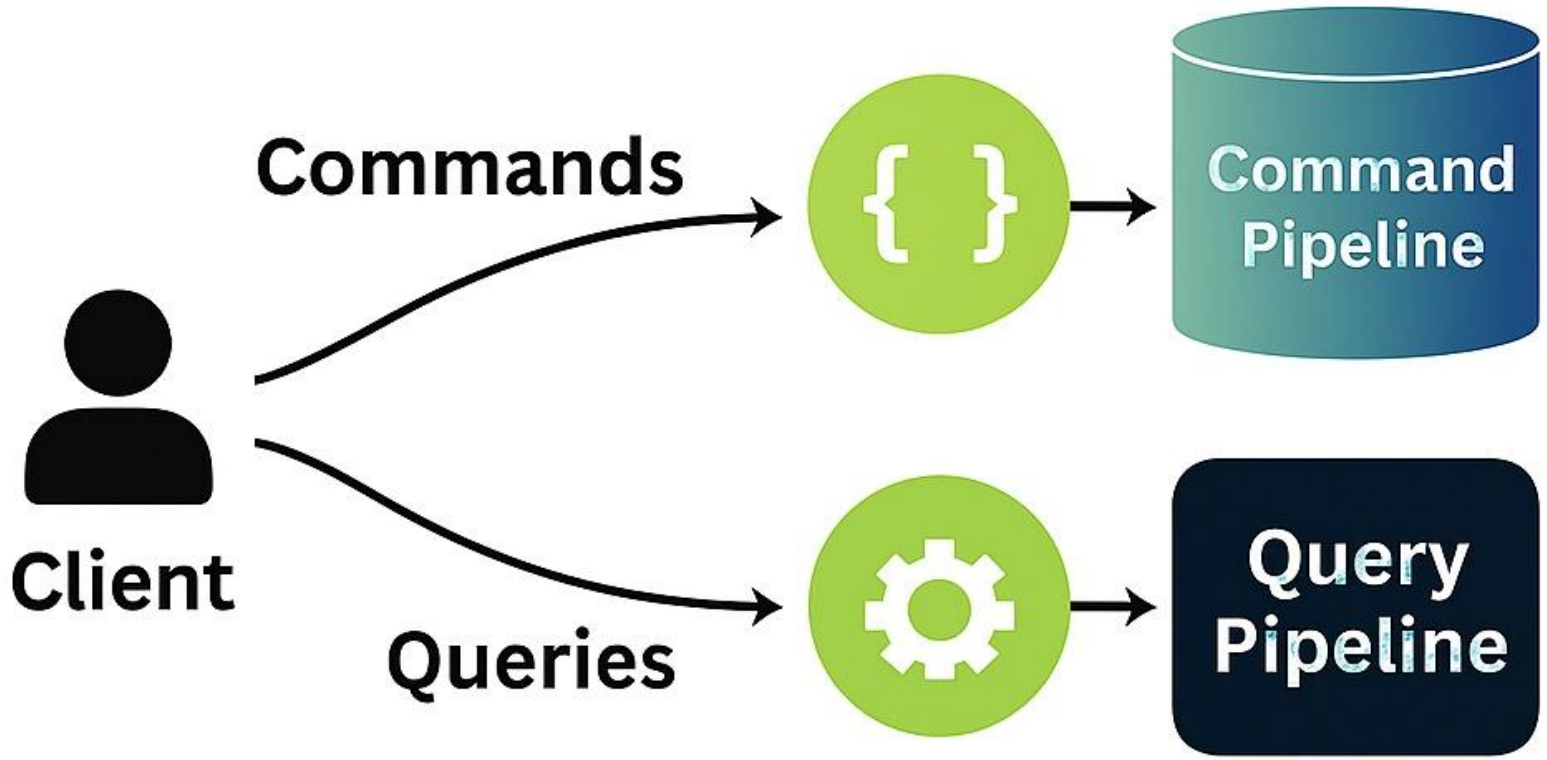
Data Lake Integration

Data Lake Integration uses serverless technologies for efficient data ingestion and transformation.

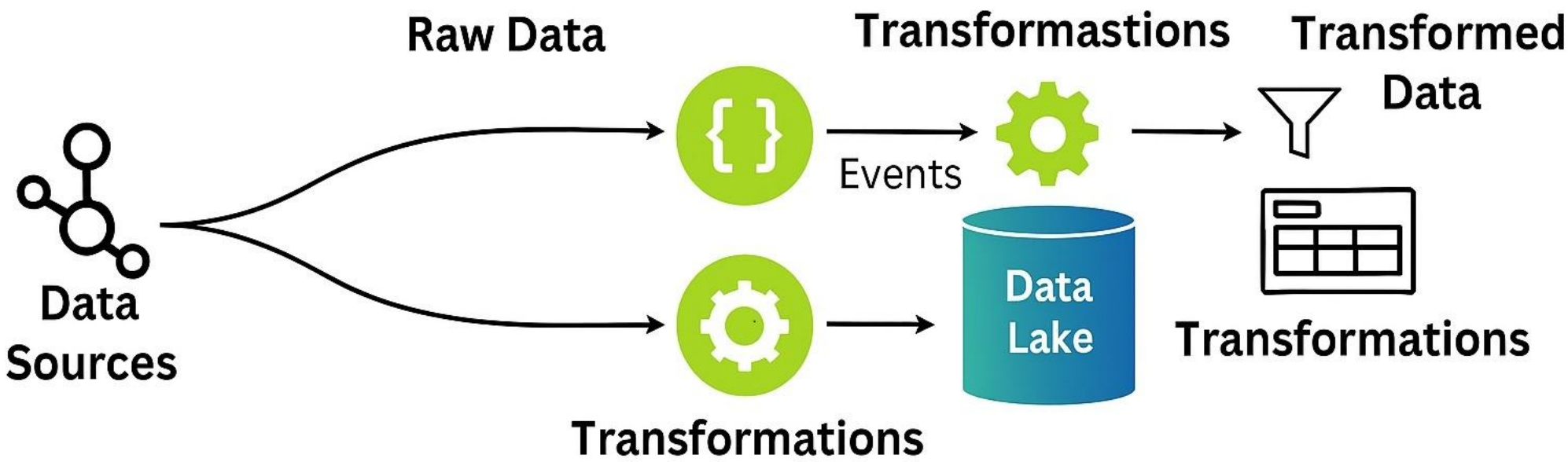
Event Sourcing



Command Query Responsibility Segregation (CQRS)



Data Lake Integration



Time & Coordination Patterns

Time-Based Triggers

Schedulers and cron-like triggers initiate automated functions at precise time intervals for consistent execution.

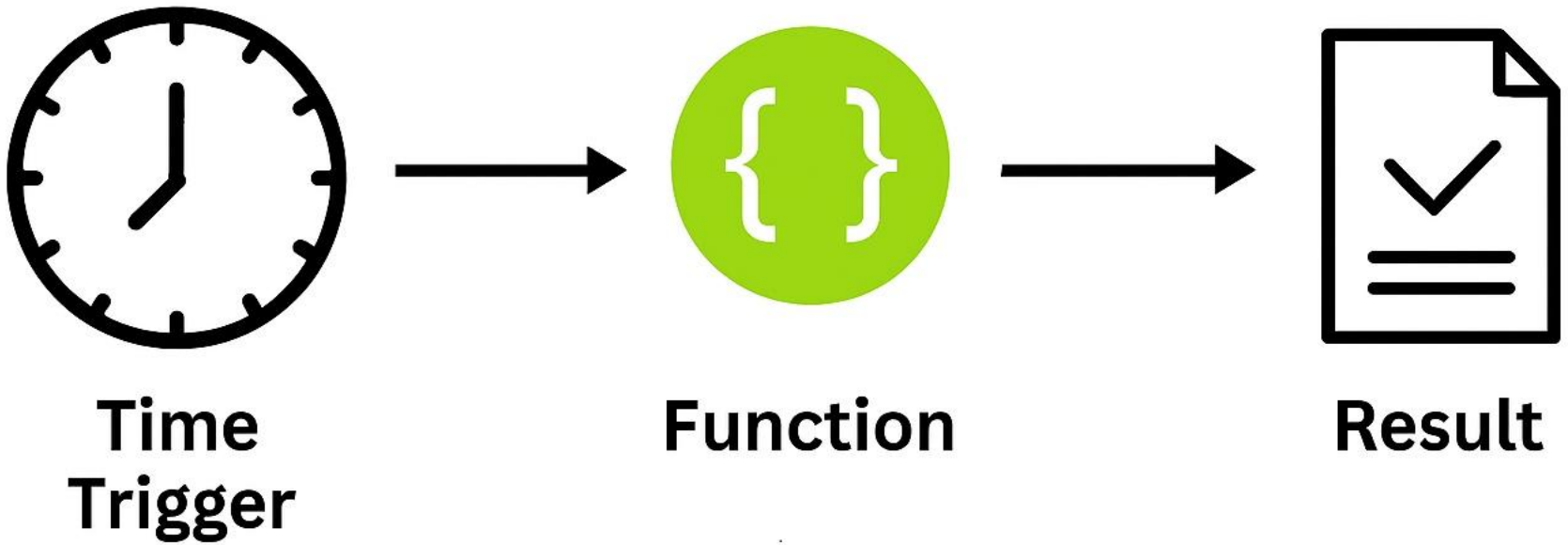
Coordinated Workflows

Orchestration patterns use workflow diagrams or control towers to manage multi-step automated processes efficiently.

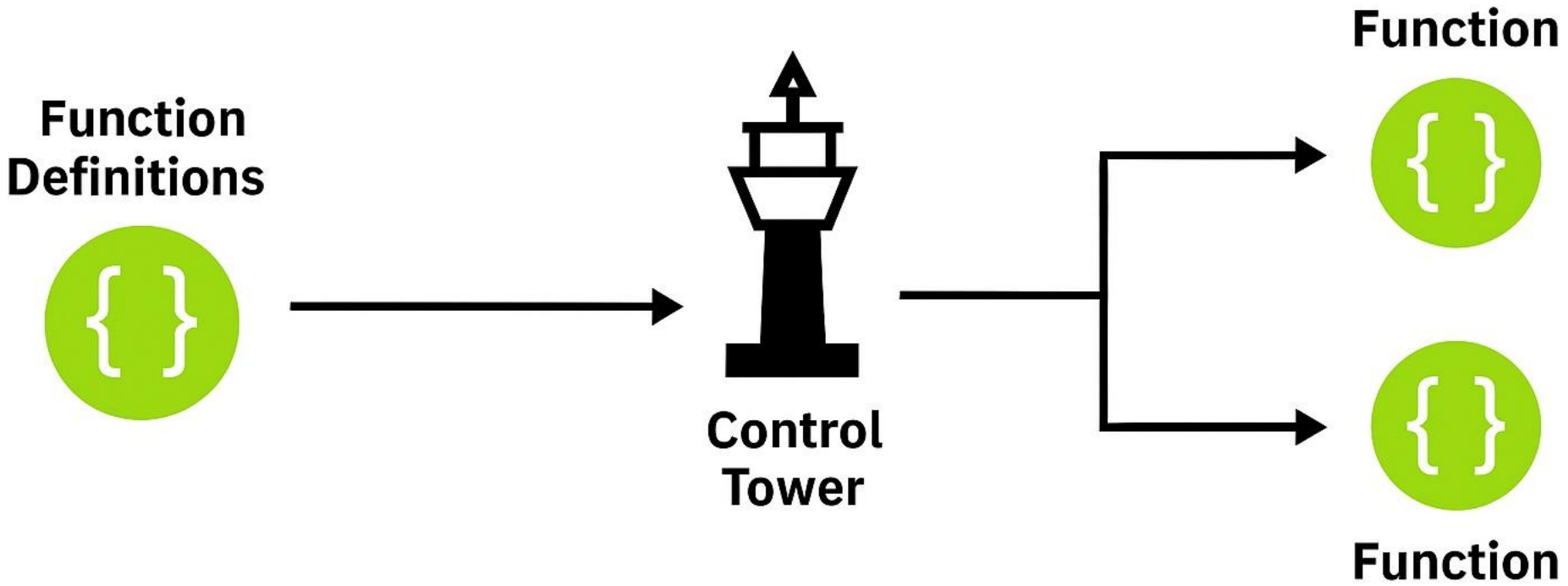
Robust Operations

Retries and compensations in workflows ensure resilient and reliable automation processes.

Scheduler/Cron

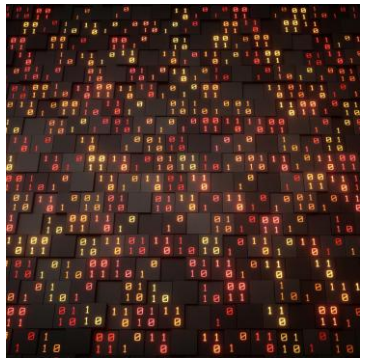


Orchestration (Workflow-as-Code)



Serverless Anti-Patterns & Pitfalls

Monolithic Lambdas (God Functions)



Challenges of Monolithic Lambdas

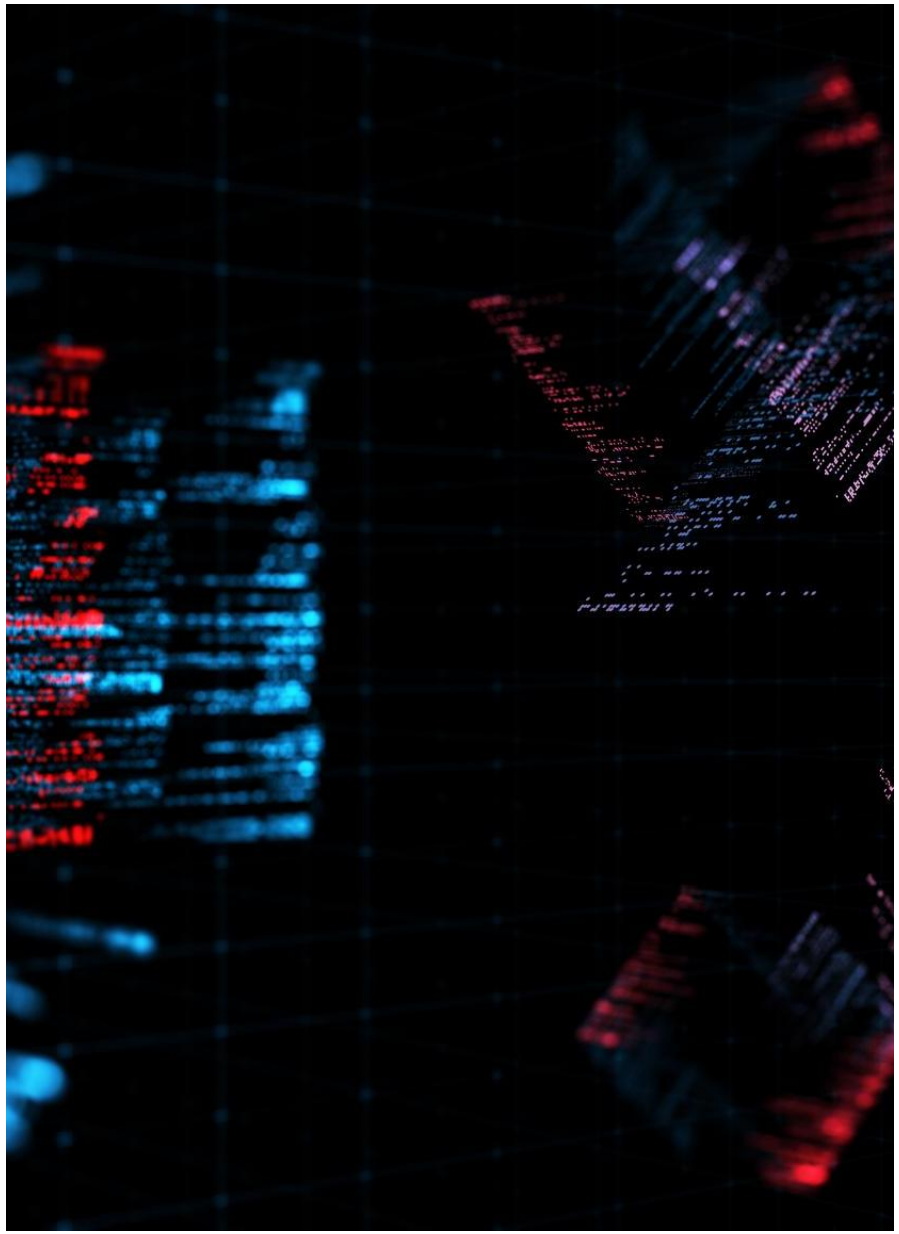
Monolithic Lambdas mix business logic, orchestration, and persistence, causing long cold starts and complex dependencies.

Negative Impact on Testing

Large serverless functions are hard to test and maintain due to intertwined responsibilities.

Adopt Composable Units

Break functions into smaller, single-responsibility units for better maintainability and scalability.



Business Logic in Handlers Only

Handlers Only Anti-pattern

Writing all business logic in serverless handlers leads to tangled code and poor maintainability. This makes it difficult to reuse or extend features.

Challenges in Testing and Reuse

Logic-heavy handlers complicate testing and make reusing code in other contexts or services challenging.

Best Practice: Thin Handlers

Maintain thin handlers by moving core logic to well-structured modules, making your codebase easier to maintain and extend.

Chatty Architectures

Excessive Synchronous Calls

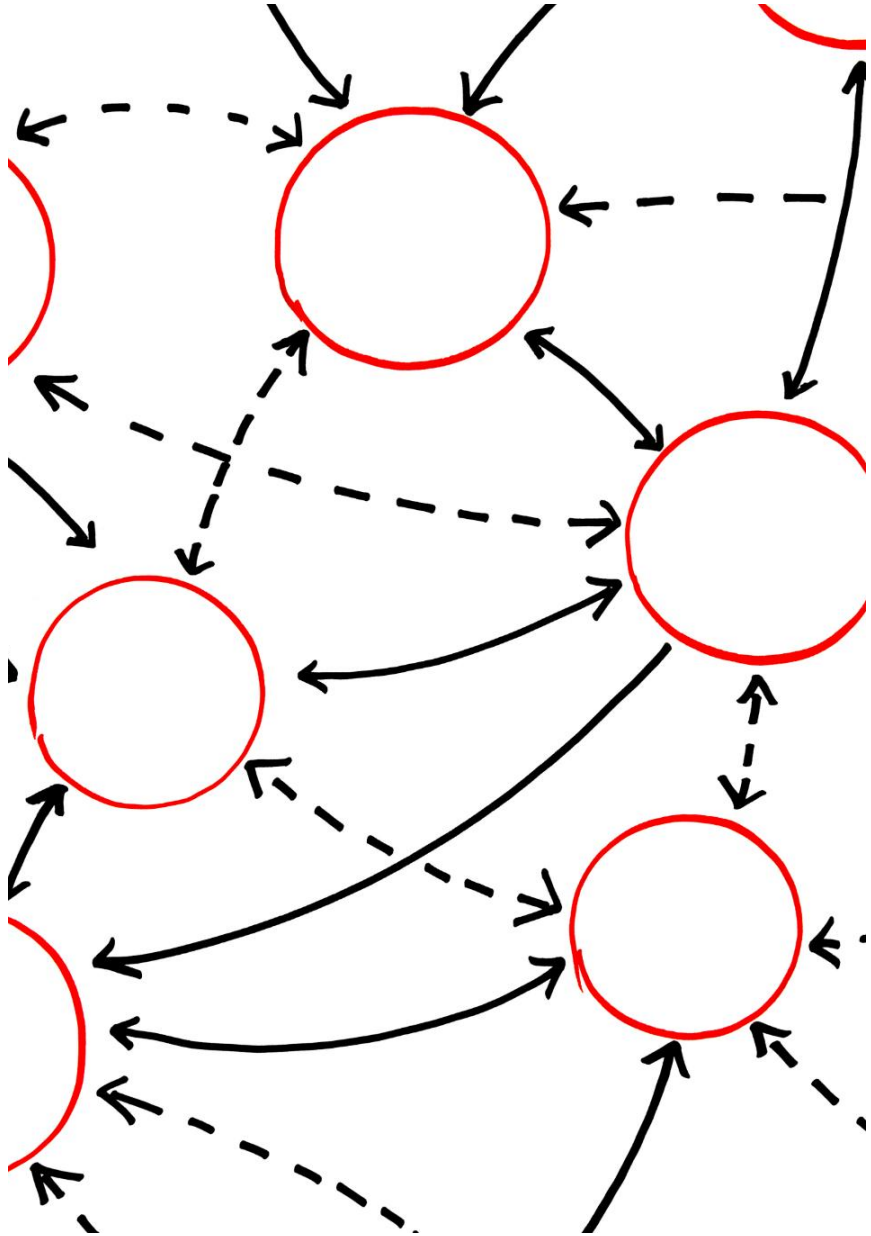
Too many direct synchronous calls between functions or services create tightly coupled dependencies in the system.

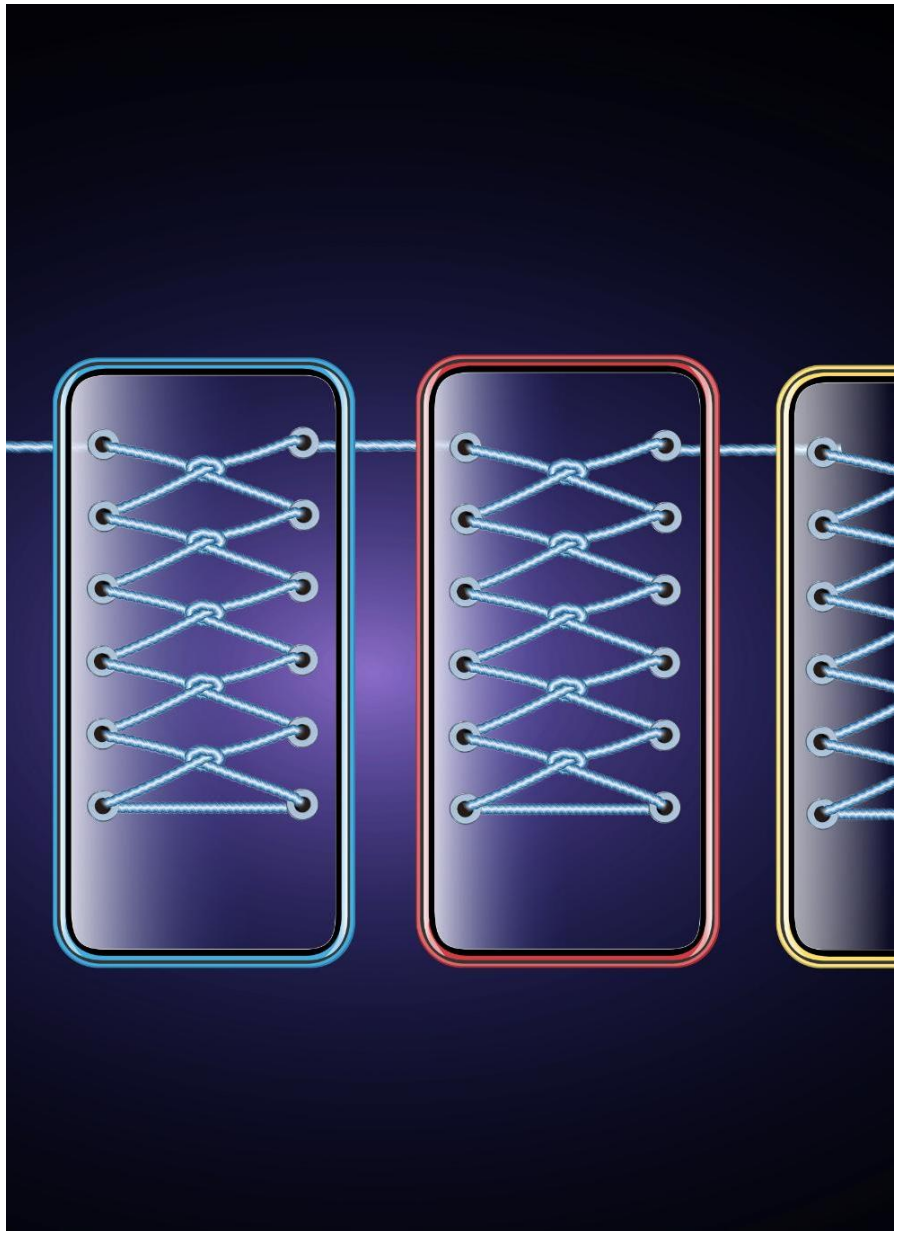
Performance and Reliability Issues

This anti-pattern increases latency, operational costs, and introduces risk of cascading failures across the architecture.

Event-Driven Asynchronous Solutions

Switching to event-driven asynchronous flows using queues or streams improves scalability and system resilience.





Ignoring Cold Starts

Impact of Cold Starts

Cold starts create unpredictable latency spikes in serverless applications, especially when scaling from zero to handle requests.

Breaking SLAs and User Trust

Latency-sensitive APIs may fail to meet SLAs, resulting in poor reliability and diminished user experience.

Mitigation Strategies

Provisioned concurrency, warming routines, and asynchronous buffering can reduce cold start impact for more consistent performance.

Avoid Stateful Assumptions

Stateless Serverless Environments

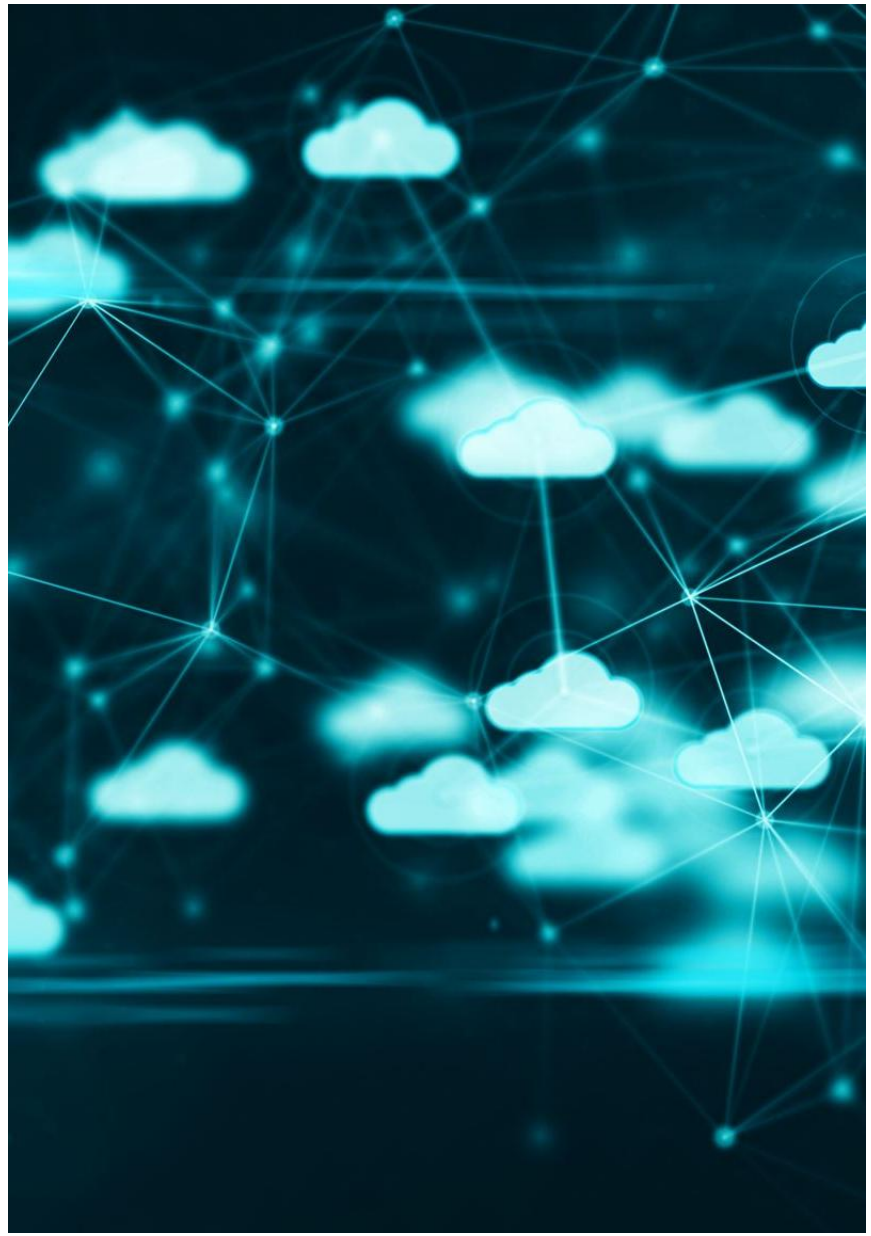
Serverless platforms are stateless by design, meaning function state does not persist between invocations.

Risks of Stateful Assumptions

Expecting persistent state can cause hard-to-detect bugs due to unpredictable container reuse and non-deterministic behavior.

Externalize State Management

Use external services for state management, such as databases or object storage, to ensure reliability.



Overusing Serverless for Long-Running Tasks

Execution Time Limits

Serverless functions performing long-running tasks often reach their maximum execution limits, causing performance bottlenecks and failures.

Increased Retries and Costs

Timeouts often lead to repeated retries, which increases operational costs without improving results or efficiency.

Alternative Solutions

Long-running processes are better handled by step functions, batch processing, or containerized services for reliability and efficiency.

Avoid Vendor Lock-In

Risks of Vendor Lock-In

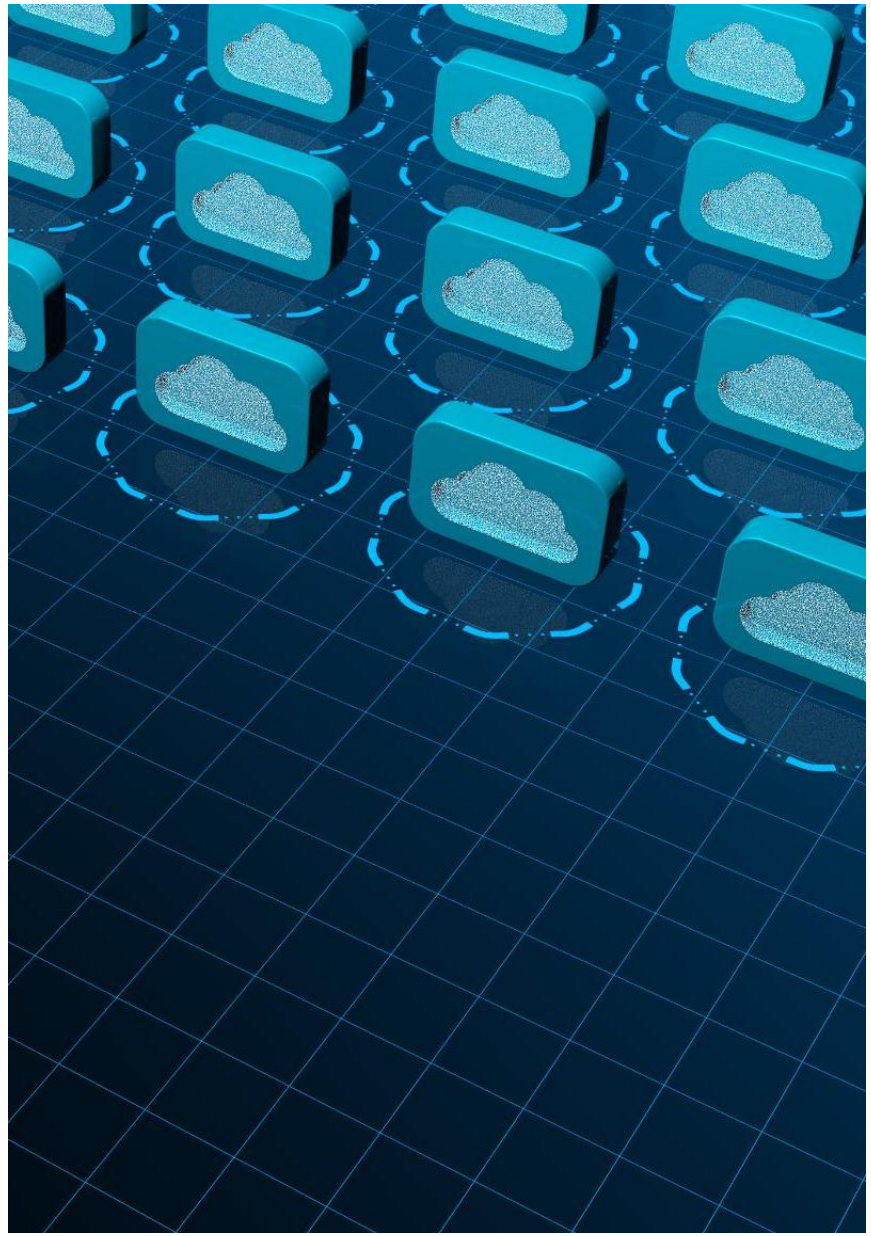
Tight coupling to a single provider’s proprietary services can hinder migration and increase long-term costs for organizations.

Use Technology Abstractions

Applying abstractions like interfaces helps decouple your application from specific providers, enabling easier migration and hybrid cloud strategies.

Infrastructure as Code and Portability

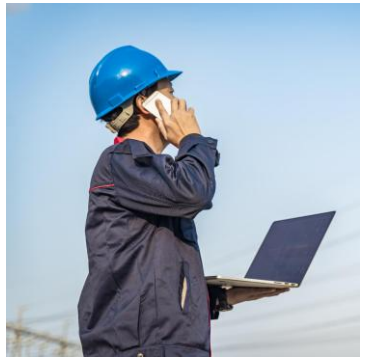
Leveraging Infrastructure as Code and portable event contracts ensures flexible, portable deployments across multiple cloud environments.



Neglecting Observability

Limited Debugging Capabilities

Without observability, serverless systems lack structured logs and tracing, making debugging distributed applications difficult and unreliable.



Guesswork in Troubleshooting

Troubleshooting often relies on guesswork when observability is neglected, increasing downtime and frustration for engineers.

Benefits of Early Observability

Implementing tools like structured logs and dashboards early improves visibility, reduces debugging time, and boosts system reliability.

Cross-Cutting Concerns in Serverless Architectures

Security & Identity: Locks and Keys on Blueprint Doors

Core Security Strategies

Security involves authentication, authorization, secrets management, and enforcing least-privilege access.

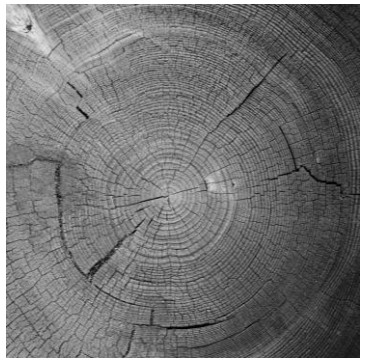
Access Control Visualization

Locks and keys on blueprint doors symbolize the importance of precise access controls in serverless architectures.

Risks of Over-Permissive Roles

Avoid over-permissive IAM roles to reduce exposure to avoidable security risks.

Observability: Measurement Scales and Inspection Windows

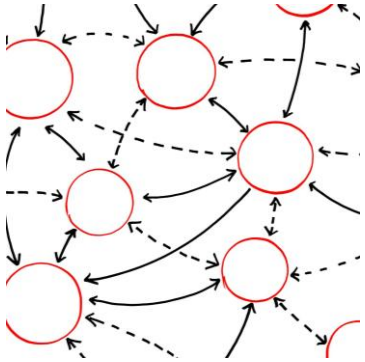


Core Observability Practices

Structured logging, distributed tracing, and metrics setup are essential for monitoring serverless systems effectively.

Measurement Scales and Inspection

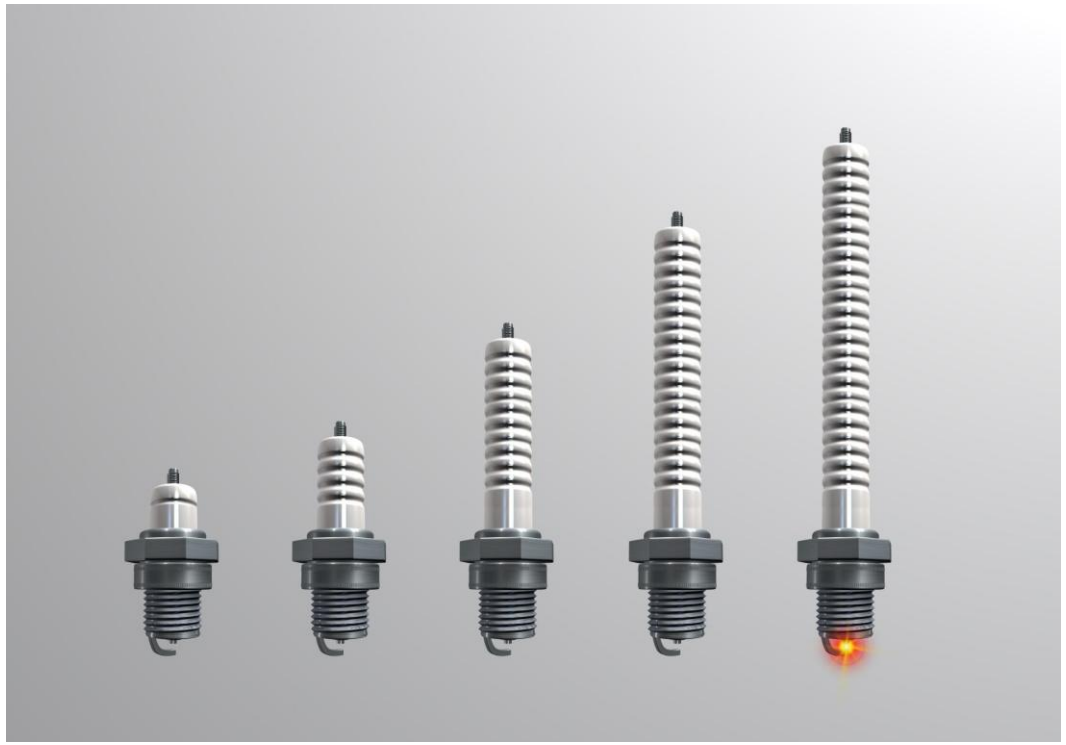
Using metaphors like measurement scales and inspection windows helps visualize gaining insight into system components.



Importance of Correlation IDs

Correlation IDs link logs and traces, preventing guesswork and helping pinpoint root causes quickly.

Resilience & Reliability: Shock Absorbers and Circuit Breakers



Designing for Failure

Resilience involves designing systems to handle failures gracefully and recover smoothly.

Resilience Techniques

Key techniques include retries, Dead Letter Queues, idempotency, timeouts, and circuit breakers.

Role of Shock Absorbers and Circuit Breakers

They act as buffers in architecture to ensure stability in unpredictable environments.

Avoiding Blind Retries

Blind retries can cause retry storms, leading to cascading failures in systems.

Cost, Performance & Governance: Meters, Stamps, and Assembly Lines

Cost Optimization and Performance

Mitigate cold starts and tune resources to optimize cost and performance, using energy meter analogies for visibility.

Governance and Compliance

Ensure governance with audit logs, versioning, and change control, symbolized by stamped approvals.

Operational Consistency

Use CI/CD pipelines and documentation to maintain consistency, like assembly lines or scaffolding in manufacturing.

Preventing Hidden Costs

Avoid shadow deployments and manual configuration errors by enforcing robust process controls.

Case Studies: Mini Blueprints From the Field in Serverless Architectures

Retail E-Commerce – Flash Sale Scalability

Serverless API Gateways

Using serverless API gateways enables automatic scaling to handle traffic spikes during flash sales efficiently.

Fan-out/Fan-in Order Handling

Fan-out/fan-in mechanisms distribute and consolidate order processing to maintain system performance under peak load.

Orchestrated Workflows

Orchestrated workflows coordinate multiple services to ensure resilience and efficiency during high-volume sales events.

Healthcare Analytics – Real-Time Patient Monitoring

Serverless Architecture Adoption

Hospitals leverage serverless architectures for scalable, event-driven real-time patient monitoring with improved responsiveness.

Scheduled Alerts and Processing

Scheduled events and alerts enable timely responses to patient conditions through cron-like job scheduling.

Secure Data Lake Integration

Data lakes securely collect and store patient data, enabling analytics while ensuring privacy and observability

S&S Media – Automated Slide Extraction From Video

Event-Driven Processing

S&S Media utilizes serverless event-driven processing for frame-by-frame video analysis.

Automated Slide Extraction

The system automatically extracts slides from speaker videos, preventing missing conference content.

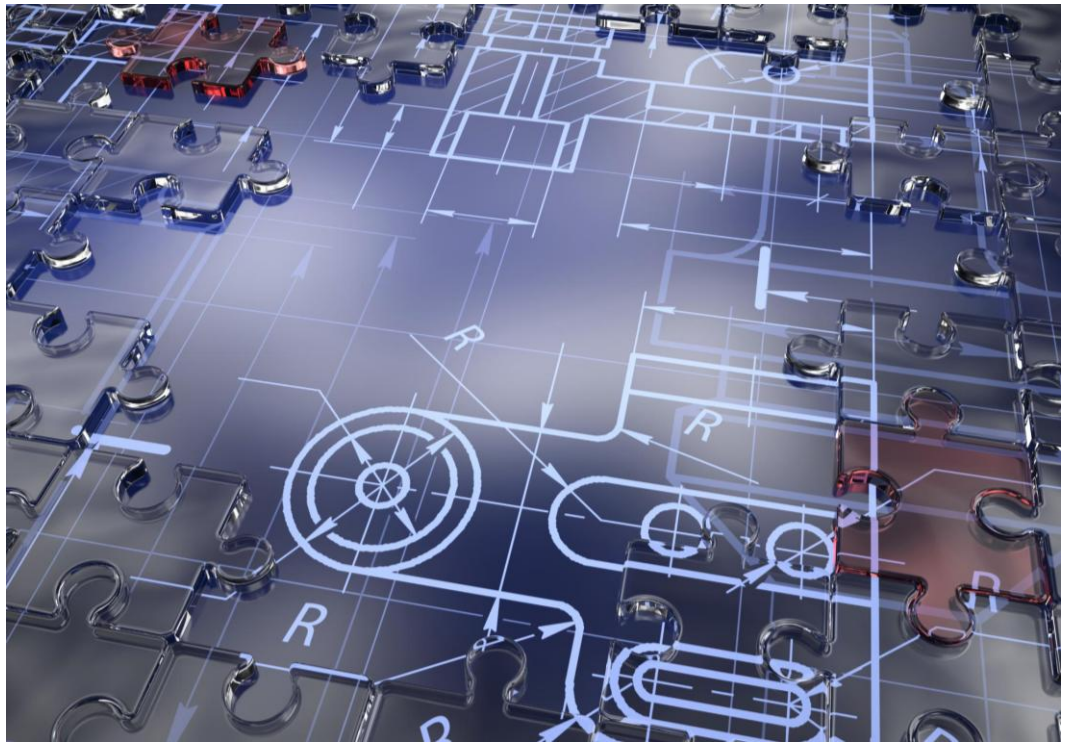
Observability and Cost Control

Cross-cutting concerns include managing operational costs efficiently.

Blueprint Metaphor

The blueprint metaphor illustrates a film reel feeding into a slide projector for automated content delivery.

Jasper Engines & Transmissions – Event-Driven ERP for Remanufacturing



Event-Driven Architecture

Jasper Engines implemented event-driven architecture using event sourcing and CQRS to enhance remanufacturing ERP.

Legacy System Replacement

The new ERP replaced legacy systems to improve operational consistency and governance in remanufacturing processes.

Serverless Integration

Serverless technologies enabled flexible integration and compliance across modular process automation stations.

Real-Time Workflow Tracking

Real-time tracking of events on the assembly line improves remanufacturing workflows and process automation.

Transition to Interactive Workshop: From Blueprints to Practice

Blueprint Index Recap: Toolboxes & Takeaways

Comprehensive Summary

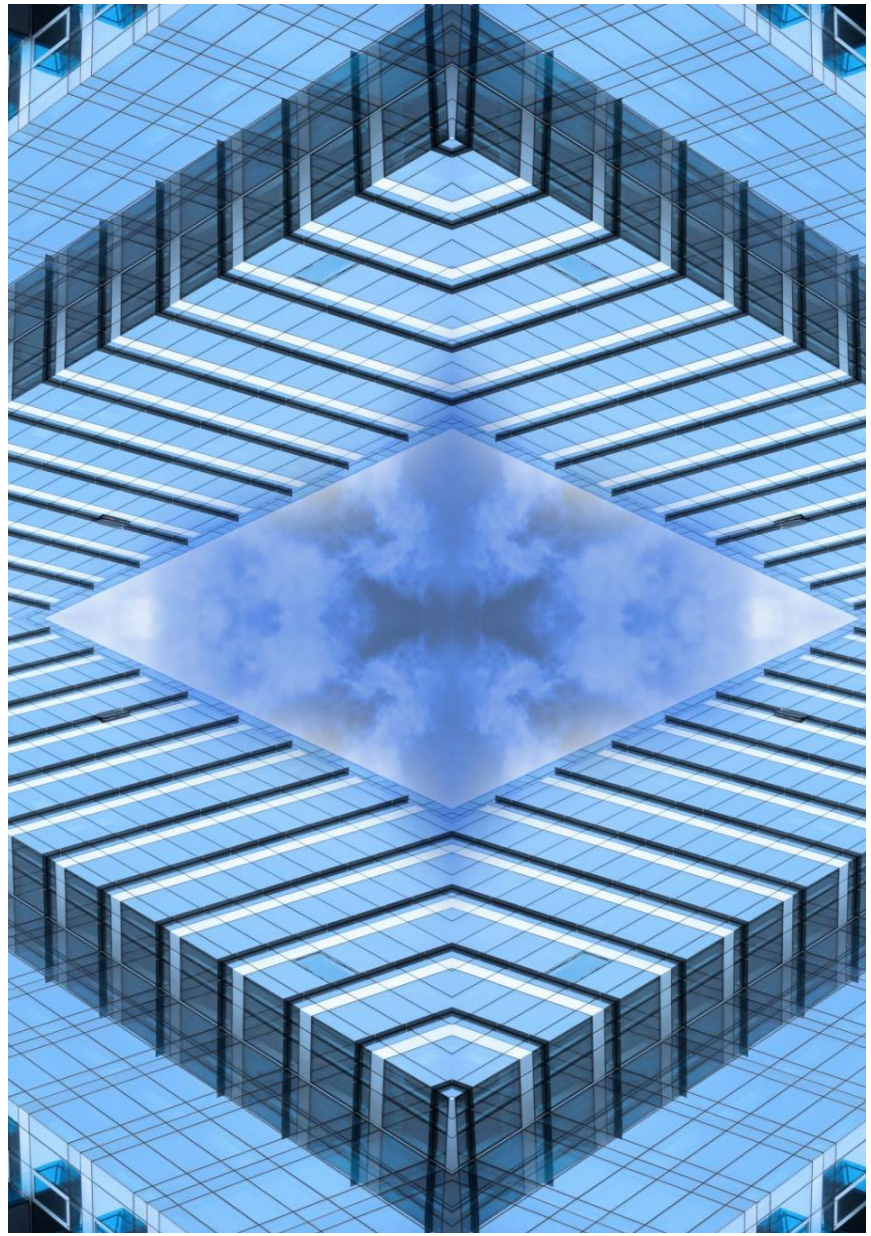
The recap visually summarizes core patterns, anti-patterns, cross-cutting concerns, and case studies for clarity.

Architectural Toolbox

Participants have built a comprehensive set of architectural tools to support design and development.

Transition to Creation

This recap sets the stage for moving from planning to active creation in the workshop environment.



Collaboration Setup: From Lecture Hall to Design Studio

Active Participation

Transition from passive learning to active engagement fosters deeper understanding and skill development.

Scenario-Driven Exercises

Using realistic scenarios helps apply serverless principles in practical and relevant contexts.

Collaborative Diagramming

Diagramming together enhances visualization and problem solving in a dynamic, studio-like environment.

Scenario Framing: Blueprinting Real-World Problems

Scenario-Based Challenges

Two design challenges use real-world scenarios framed as blueprint problem statements to guide decision-making.

Critical Evaluation

Teams evaluate patterns, anti-patterns, and cross-cutting concerns relevant to each challenge for effective solutions.

Cloud-Native Trade-Offs

Participants are encouraged to analyze trade-offs in cloud-native design to develop balanced, practical solutions.

No Single Correct Answer