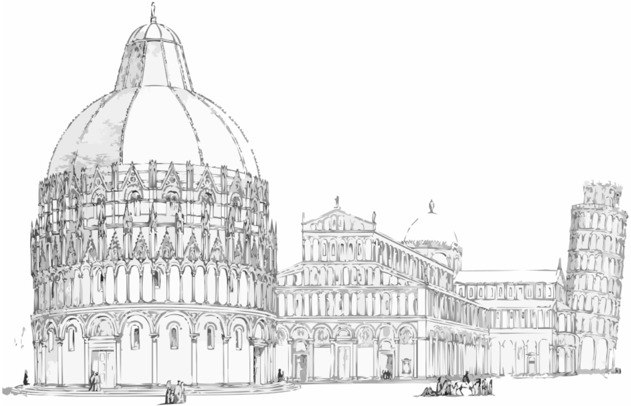
UNIVERSITÀ DI PISA

Bridging the Gap: The Convergence of Cloud-Native Serverless and HPC Workloads

Pesaresi Seminar



Matteo Della Bartola[1]

[1] Department of Computer Science, University of Pisa

February 11, 2026

Who am I?

Matteo Della Bartola
PhD student at the University of Pisa Department of Computer Science.

Working with the Parallel Programming Models (PPM) Group on *NOUS*, an EU-funded project.

Background:

- B.Sc. in Computer Science 2022, University of Pisa
- M.Sc. in Computer Science 2024, University of Pisa
- Research interests: HPC, serverless computing, parallel programming, and more...

What to expect from this presentation?

Today's Topics:

→ Introduction to Cloud Computing

→ Introduction to Serverless Computing

→ High-Performance Serverless Computing SLR

→ HPSC Future Directions & Open Problems

Cloud Computing - A Definition

Cloud Computing

Cloud computing is a utility-oriented and Internet-centric way of delivering IT services on demand. These services cover the entire computing stack: from the hardware infrastructure packaged as a set of virtual machines to software services such as development platforms and distributed applications.[1]

[1] R. Buyya, C. Vecchiola, and S.T. Selvi. *Mastering Cloud Computing: Foundations and Applications Programming*. ITPro collection. Morgan Kaufmann, 2013. ISBN: 9780124095397. URL: `https://books.google.it/books?id=wqKkqHJhPJQC`

Players in Cloud Computing

Provider: Provides resources, services, and applications (cloud services from now on) to consumers.

- Cloud (infrastructure) provider
- Service / application / software provider
- Platform provider

Players in Cloud Computing

Provider: Provides resources, services, and applications (cloud services from now on) to consumers.

- Cloud (infrastructure) provider
- Service / application / software provider
- Platform provider

Consumer: Consumes cloud services provided by the provider.

- a.k.a., cloud **customer**
- Pays the bill

Players in Cloud Computing

Provider: Provides resources, services, and applications (cloud services from now on) to consumers.

- Cloud (infrastructure) provider
- Service / application / software provider
- Platform provider

Consumer: Consumes cloud services provided by the provider.

- a.k.a., cloud **customer**
- Pays the bill

End User: Uses cloud services.

Deployment Models

Public Cloud:

- Provisioned for open use by the general public.
- Owned by an organization and exists on the provider's premises.

[1] Peter Mell and Timothy Grance. *The NIST definition of cloud computing*. Tech. rep. Special Publication 800-145. National Institute of Standards and Technology, 2011

Deployment Models

Public Cloud:

- Provisioned for open use by the general public.
- Owned by an organization and exists on the provider's premises.

Private Cloud:

- Provisioned for exclusive use by a single organization.
- Can be on or off premises; managed by the org or a third party.

[1] Peter Mell and Timothy Grance. *The NIST definition of cloud computing*. Tech. rep. Special Publication 800-145. National Institute of Standards and Technology, 2011

Deployment Models

Public Cloud:

- Provisioned for open use by the general public.
- Owned by an organization and exists on the provider's premises.

Private Cloud:

- Provisioned for exclusive use by a single organization.
- Can be on or off premises; managed by the org or a third party.

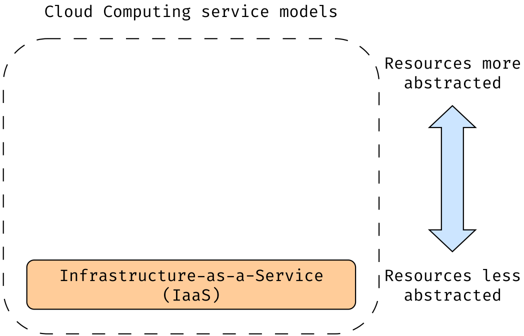
Hybrid Cloud:

- Composition of two or more distinct clouds (Private, Community, or Public).
- Bound by technology enabling data and application portability (e.g., cloud bursting).

[1] Peter Mell and Timothy Grance. *The NIST definition of cloud computing*. Tech. rep. Special Publication 800-145. National Institute of Standards and Technology, 2011

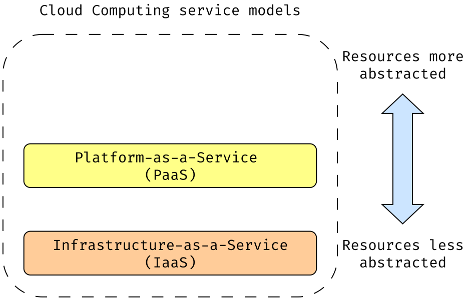
Provisioning Models

- Infrastructure-as-a-Service (IaaS)



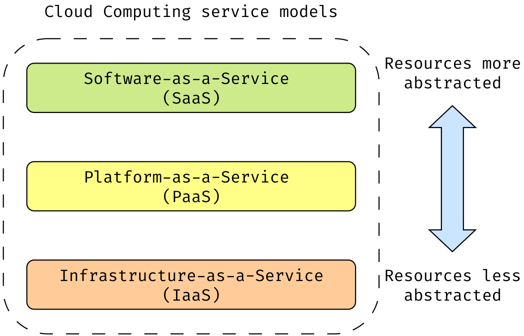
Provisioning Models

- Infrastructure-as-a-Service (IaaS)
- Platform-as-a-Service (PaaS)



Provisioning Models

- Infrastructure-as-a-Service (IaaS)
- Platform-as-a-Service (PaaS)
- Software-as-a-Service (SaaS)



The XaaS Trend

- Infrastructure-as-a-Service (IaaS)
- Platform-as-a-Service (PaaS)
- Software-as-a-Service (SaaS)
- Network-as-a-Service (NaaS)
- Function-as-a-Service (Faas): Serverless Computing
- Security-as-a-Service: Firewall, IDS, authentication,
- DB, storage, caching, queueing, analytics-as-a-Service

Serverless Computing

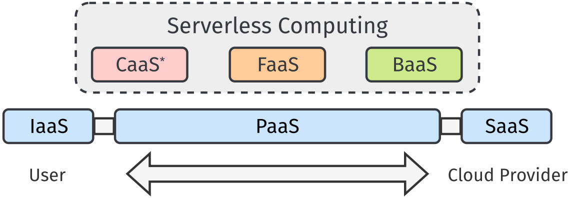
Serverless Computing

Serverless computing is a cloud computing execution model that allows users to deploy and execute fine-grained billed and automatically scaled applications, without having to address the underlying operational logic.[2]

Serverless Computing

Serverless Computing

Serverless computing is a cloud computing execution model that allows users to deploy and execute fine-grained billed and automatically scaled applications, without having to address the underlying operational logic.[2]



Who is responsible for the operational logic?

**serverless-enabled offerings*

Serverless Service Models

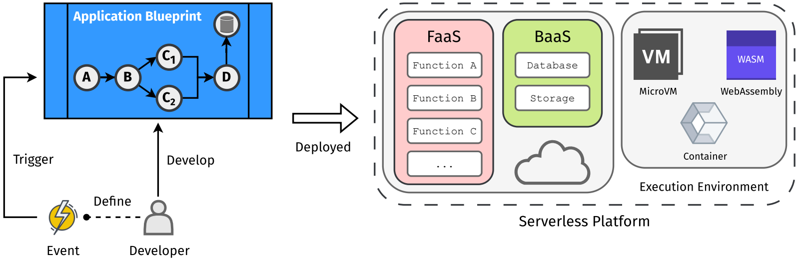
- **FaaS (Function-as-a-Service):** Focuses on managing resource requirements and event-driven triggering of functions.
- **CaaS (Container-as-a-Service):** Allows deployment of containers; considered serverless depending on abstraction levels.
- **BaaS (Backend-as-a-Service):** Provides specialized frameworks for specific needs like storage, databases, or messaging.

Serverless Computing Core Traits

- **No operational logic (NoOps)**
  - Complete abstraction of servers, OS patching, and infrastructure management.
  - Developers focus solely on business logic and code.
- **Utilization-based billing**
  - *Pay-per-use model:* Costs are calculated based on execution time and memory allocated.
  - *Scale-to-zero:* No cost is incurred when functions are idle.
- **Auto-scaling**
  - Rapid, automatic scaling from zero to thousands of instances based on incoming events.
  - Handles ”bursty” traffic without manual intervention or pre-provisioning.

Together, these properties distinguish serverless from PaaS and IaaS, making it ideal for the event-driven workflows shown previously.

Serverless Workflow



Serverless Platforms

Implementing the FaaS and BaaS Layer

The **Serverless Platform** integrates the FaaS (Functions) and BaaS (State) components shown in the previous workflow.

- **Commercial Cloud Providers**
  - **AWS Lambda:** The market leader; deeply integrated with AWS BaaS services (DynamoDB, S3).
  - **Azure Functions:** Focus on enterprise integration and event triggers.
  - **Google Cloud Functions:** Optimized for data processing and Firebase events.
- **Open Source & Kubernetes**
  - **Knative:** Provides the "Serving" and "Eventing" primitives on top of Kubernetes.
  - **Apache OpenWhisk:** A flexible, distributed event-based programming service.
  - **OpenFaaS [6]:** Simplifies deploying functions to Docker Swarm or Kubernetes.

Execution Environments

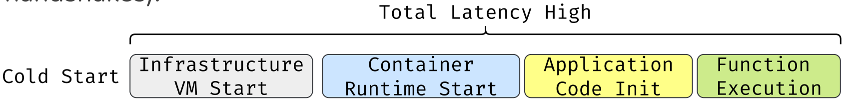
Isolation Technologies

To execute the "Application Blueprint" securely and efficiently, platforms rely on three main isolation technologies:

1. **Containers** (e.g., Docker, OCI)
   - The standard unit of software. Packages code and dependencies.
   - *Trade-off*: High compatibility but slower "cold starts."
2. **MicroVMs** (e.g., Firecracker, gVisor)
   - Lightweight Virtual Machines. They offer the strong security isolation of traditional VMs but with millisecond boot times.
   - Used heavily by AWS Lambda and Fargate.
3. **WebAssembly (Wasm)**
   - A binary instruction format for a stack-based virtual machine.
   - **Key Advantage:** Near-instant startup, platform independence, and extremely high density compared to containers.

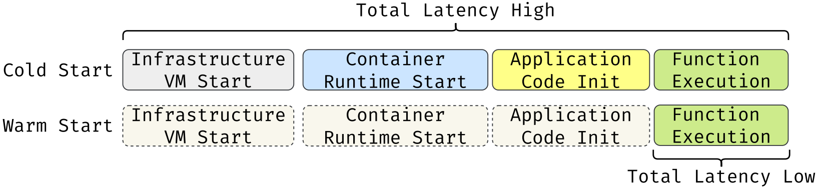
The Cold Start Problem in FaaS

- **Definition:** The latency penalty incurred when a FaaS function is invoked after a period of inactivity or for the first time.
- **Why it happens:** The provider must prepare a fresh isolation layer. Before execution, it must:
  - **Infrastructure Provisioning:** Allocate a **MicroVM (e.g., Firecracker [1])** or a **Container**.
  - **Image/Code Loading:** Download and unpack the container image or deployment package.
  - **Environment Setup:** Start the runtime (e.g., JVM, Node.js) and configure system-level resources.
  - **Application Init:** Run initialization logic (e.g., importing heavy libraries, DB handshakes).



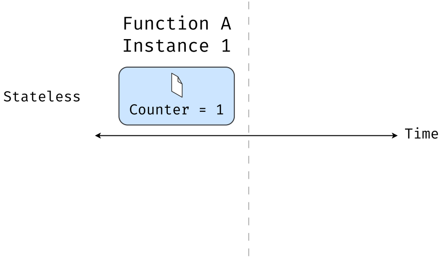
The Cold Start Problem in FaaS

- **Definition:** The latency penalty incurred when a FaaS function is invoked after a period of inactivity or for the first time.
- **Why it happens:** The provider must prepare a fresh isolation layer. Before execution, it must:
  - **Infrastructure Provisioning:** Allocate a **MicroVM (e.g., Firecracker [1])** or a **Container**.
  - **Image/Code Loading:** Download and unpack the container image or deployment package.
  - **Environment Setup:** Start the runtime (e.g., JVM, Node.js) and configure system-level resources.
  - **Application Init:** Run initialization logic (e.g., importing heavy libraries, DB handshakes).



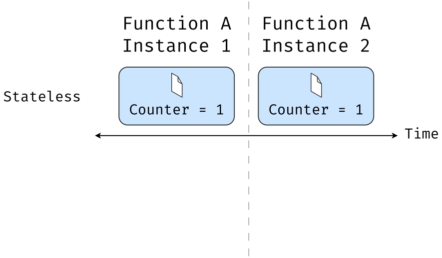
Stateless Computation in Serverless

- **The Core Principle:** FaaS functions are ephemeral (short-lived). They do not retain data between executions.
- **Local Storage is Temporary:**
  - You can write to memory or '/tmp' disk (AWS Lambda gives you 512 MB by default of /tmp storage).
  - However, once the function finishes, that environment may be destroyed immediately.



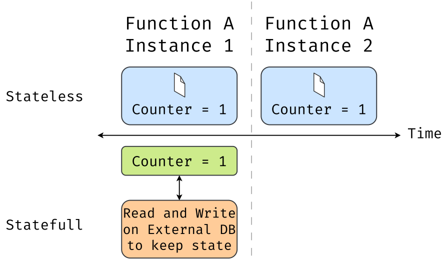
Stateless Computation in Serverless

- **The Core Principle:** FaaS functions are ephemeral (short-lived). They do not retain data between executions.
- **Local Storage is Temporary:**
  - You can write to memory or '/tmp' disk (AWS Lambda gives you 512 MB by default of /tmp storage).
  - However, once the function finishes, that environment may be destroyed immediately.



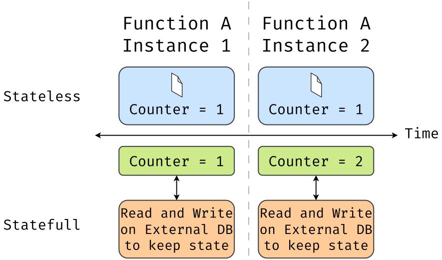
Stateless Computation in Serverless

- **The Core Principle:** FaaS functions are ephemeral (short-lived). They do not retain data between executions.
- **Local Storage is Temporary:**
  - You can write to memory or '/tmp' disk (AWS Lambda gives you 512 MB by default of /tmp storage).
  - However, once the function finishes, that environment may be destroyed immediately.
- **How to Persist Data:**
  - State must be externalized.
  - Use managed services (BaaS): S3 (files), DynamoDB/Redis (data), SQS (queues).



Stateless Computation in Serverless

- **The Core Principle:** FaaS functions are ephemeral (short-lived). They do not retain data between executions.
- **Local Storage is Temporary:**
  - You can write to memory or '/tmp' disk (AWS Lambda gives you 512 MB by default of /tmp storage).
  - However, once the function finishes, that environment may be destroyed immediately.
- **How to Persist Data:**
  - State must be externalized.
  - Use managed services (BaaS): S3 (files), DynamoDB/Redis (data), SQS (queues).



High-Performance Serverless: SLR Overview

Journals & Magazines > IEEE Access > Volume: 13

High-Performance Serverless Computing: A Systematic Literature Review on Serverless for HPC, AI, and Big Data

Datasets Available

Publisher: IEEE Cite This PDF

Valerio Besozzi ; Matteo Della Bartola ; Patrizio Dazzi ; Marco Danelutto **All Authors**

High-Performance Serverless: SLR Overview (Cont.)

Research Goal

To investigate the convergence of **High-Performance Computing (HPC)** and **Cloud-Native** paradigms, analyzing how serverless models support compute-intensive workloads like AI, Big Data, and Scientific Computing.

Scope of Study

- **Timeline:** 2017 – Early 2025
- **Scale:** 122 Primary Studies selected
- **Sources:** ACM, IEEE Xplore, ScienceDirect

Key Contributions

- **Taxonomy:** 8 primary research directions & 9 use case domains.
- **Trends:** Analysis of publication growth and author collaboration.
- **Gaps:** Identification of open problems (e.g., cold starts, heterogeneity).

Research Methodology: Protocol & Strategy

Framework:

- Follows guidelines by Kitchenham et al. [5] for systematic literature reviews in software engineering.
- **Objective:** Analyze the suitability of serverless for HPC, AI, and Big Data workloads.

Research Questions (RQs):

1. **Directions:** Identification of taxonomy and research tracks.
2. **Solutions:** Approaches to specific HPC/Serverless problems.
3. **Use Cases:** Domain-specific applications (e.g., AI inference).
4. **Trends:** Growth, impact, and publication venues.
5. **Community:** Key contributors and collaboration patterns.

Search Strategy:

- **Sources:** ACM Digital Library, Elsevier ScienceDirect, IEEE Xplore.
- **String:** `(Serverless OR FaaS) AND (HPC OR Accelerators)`.
- **Timeframe:** January 2017 – February 2025.

Selection Process & Data Analysis

Inclusion & Exclusion Criteria:

- **Included:** Peer-reviewed papers on Serverless for HPC/AI inference, proposing designs, algorithms, or frameworks.
- **Excluded:** Secondary studies, theses, pre-prints.

The Selection Process:

1. **Initial Search:** 579 papers retrieved.
2. **Screening:** Dual-author review of titles/abstracts.
3. **Manual Additions:** Added 3 seminal works missed by search.
4. **Snowballing:** Followed Wohlin's guidelines (backward/forward), adding 15 relevant studies.
5. **Final Set: 122 primary studies** selected.

Analysis Techniques:

- **Taxonomy:** Constructed via open coding (grouping phrases into categories).
- **Bibliometrics:** Citation analysis and co-authorship networks using VOSviewer.

Current Landscape: Open Problems & Limitations

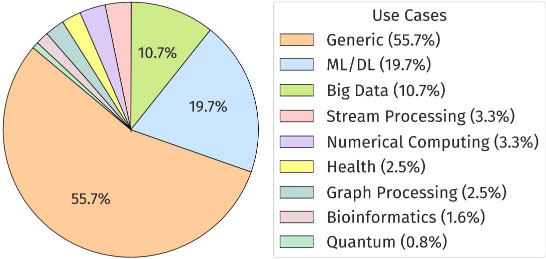
The Challenge: While serverless offers elasticity and scalability, its adoption in HPC is hindered by architectural mismatches between the *stateless*, *ephemeral* nature of FaaS and the *stateful*, *long-running* nature of scientific applications.

Three Fundamental Barriers:

1. **Cold Start Latency:** Unpredictable delays unacceptable for tightly coupled tasks.
2. **Data Communication:** High overhead from mediated storage (e.g., S3) versus direct HPC interconnects.
3. **Hardware Heterogeneity:** Difficulty in efficiently sharing and abstracting accelerators (GPUs, FPGAs, DPUs).

Additionally, critical gaps remain in Sustainability, Security, and Standardization.

Taxonomy of Use Cases: Distribution



Dominance of Generic Research

- **55.7%** of studies focus on general-purpose workloads.
- Indicates a "foundational phase": priority is currently on building robust platforms and frameworks rather than specific applications.

The Leading Specialization

- **ML/DL (19.7%)** has emerged as the primary specialized domain.
- Driven by the natural alignment between stateless, bursty inference tasks and the FaaS model.

Open Problem I: Cold Start Latency

The HPC Context: Unlike web traffic, HPC workloads often lack historical patterns, making standard time-series prediction ineffective.

Current Limitations:

- **Inefficient Mitigation:** Common ”Pre-warming” techniques keep resources idle, contradicting the cost-efficiency of serverless.
- **DAG Complexity:** In workflow-based applications (DAGs), a cold start in one function triggers cascading delays across the entire pipeline.

Future Directions:

- Need for platform-level startup reduction (e.g., snapshotting, lightweight isolation).
- Prediction models tailored specifically for scientific workflow structures rather than historical invocation rates.

Open Problem II: Data Communication

The Bottleneck: Traditional FaaS relies on *mediated communication* (passing data via Object Storage/DBs), introducing latency for parallel HPC tasks.

The ”Serverless Trilemma”: Current systems struggle to achieve three goals simultaneously:

1. Functions as ”Black Boxes” (Composability).
2. Double-billing protection.
3. **High Performance / Low Latency.**

Required Innovation:

- Shift toward **Direct Communication** (e.g., TCP hole punching [3], RDMA support [4]).
- New abstractions that hide communication complexity without sacrificing performance.

Open Problem III: State Management

Stateless vs. Stateful:

- **Paradigm Clash:** FaaS enforces statelessness for elasticity.
- **Reality:** HPC applications (e.g., simulations, training) are inherently stateful and iterative.

Current Inefficiencies:

- Offloading state to external storage (S3/Redis) creates I/O bottlenecks.
- Lack of efficient shared memory models between function instances.

Research Opportunity:

- Development of **Direct Memory Sharing** techniques (e.g., Faasm [8]).
- Efficient, ephemeral storage layers co-located with compute nodes to support "Stateful Serverless."

Open Problem IV: Hardware Heterogeneity

Accelerator Integration (GPUs, FPGAs, NPUs):

- Current platforms often provide coarse-grained ”full device” access, leading to under-utilization.
- Lack of native support for **Resource Disaggregation** (separating compute/memory/accelerators).

Multiplexing Challenges:

- **Techniques:** Using NVIDIA MIG/MPS or Kernel Slicing.
- **Issue:** These introduce significant management complexity and scheduling overheads not handled by standard FaaS providers.

Gap: General-purpose frameworks capable of seamlessly abstracting heterogeneous hardware while maintaining multi-tenant security and isolation.

Open Problem V: Systemic Research Gaps

1. **Sustainability & Energy:**
   - Energy-aware scheduling is largely unexplored.
   - Need to balance ”performance-at-any-cost” with carbon footprint metrics in distributed serverless environments.
2. **Security in HPC FaaS:**
   - Multi-tenant accelerator sharing introduces new attack surfaces (side-channels).
   - Federated workflows require complex authentication/authorization models currently missing.
3. **Benchmarking Standard:**
   - **Current State:** Fragmented, ad-hoc evaluation setups.
   - **Need:** A standardized benchmark suite specifically for *High-Performance* Serverless (beyond simple web microservices).

Conclusions

Summary & Key Findings

The Evolution of Cloud Computing

- We observed a shift from infrastructure-heavy models (IaaS) to pure abstraction (FaaS), where developers focus solely on code.
- **Trade-off:** Serverless offers rapid auto-scaling and pay-per-use but introduces *Cold Starts* and *Statelessness* challenges.

Conclusions

Summary & Key Findings

The Evolution of Cloud Computing

- We observed a shift from infrastructure-heavy models (IaaS) to pure abstraction (FaaS), where developers focus solely on code.
- **Trade-off:** Serverless offers rapid auto-scaling and pay-per-use but introduces *Cold Starts* and *Statelessness* challenges.

SLR Insights (2017–2025)

- **Growing Interest:** Analysis of 122 studies confirms a surge in Serverless for HPC, AI, and Big Data.
- **Dominant Workloads:**
  - **Generic (55.7%):** Foundational research on platforms dominates.
  - **ML/DL (19.7%):** The fastest-growing domain due to bursty inference workloads.

Conclusions

Future Research Directions

To make Serverless fully viable for HPC, we must address these open problems:

1. **Latency:** Mitigating cold starts without inefficient pre-warming (e.g., snapshotting).
2. **Communication:** Moving from mediated storage (S3) to direct communication (e.g., RDMA/TCP hole punching).
3. **State Management:** Developing efficient "Stateful Serverless" and shared memory models.
4. **Heterogeneity:** Better abstraction for hardware accelerators (GPUs/FPGAs).
5. **Systemic Maturity:** Establishing standard benchmarks and energy-aware scheduling.

Q&A

Thank you for your attention!
Any Questions?

Slides proudly made in LaTeX

Bibliography

[1] Alexandru Agache et al. "Firecracker: Lightweight virtualization for serverless applications". In: (2020). URL: https://www.amazon.science/publications/firecracker-lightweight-virtualization-for-serverless-applications.

[2] R. Buyya, C. Vecchiola, and S.T. Selvi. *Mastering Cloud Computing: Foundations and Applications Programming*. ITPro collection. Morgan Kaufmann, 2013. ISBN: 9780124095397. URL: https://books.google.it/books?id=wqKkqHJhPJQC.

[3] Marcin Copik et al. "FMI: Fast and Cheap Message Passing for Serverless Functions". In: June 2023, pp. 373–385. DOI: 10.1145/3577193.3593718.

Bibliography

[4] Marcin Copik et al. "rFaaS: Enabling High Performance Serverless with RDMA and Leases". In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2023, pp. 897–907. DOI: 10.1109/IPDPS54959.2023.00094.

[5] Barbara Kitchenham and Stuart Charters. "Guidelines for performing Systematic Literature Reviews in Software Engineering". In: 2 (Jan. 2007).

[6] Dac-Nhuong Le, Souvik Pal, and Prasant Kumar Pattnaik. "OpenFaaS". In: *Cloud Computing Solutions*. John Wiley Sons, Ltd, 2022. Chap. 17, pp. 287–303. ISBN: 9781119682318. DOI: https://doi.org/10.1002/9781119682318.ch17. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119682318.ch17. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119682318.ch17.

Bibliography

[7] Peter Mell and Timothy Grance. *The NIST definition of cloud computing*. Tech. rep. Special Publication 800-145. National Institute of Standards and Technology, 2011.

[8] Simon Shillaker and Peter Pietzuch. “Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing”. In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, July 2020, pp. 419–433. ISBN: 978-1-939133-14-4. URL: `https://www.usenix.org/conference/atc20/presentation/shillaker`.