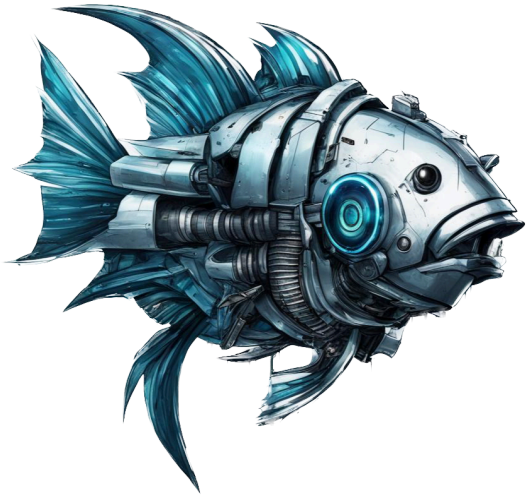
Technology Radar (2024 edition) ¦ Dmitry Kolesnikov

https://tech.fog.fish ¦ dmkolesnikov@gmail.com ¦ https://www.linkedin.com/in/dmkolesnikov/

What is the Tech Radar?



ThoughtWorks maintains highlights about currently interesting technologies in software development. As they said - “things in motion that we think you should pay attention to and consider using in your projects”. It inspires teams to pick the best technologies and tools to get the job well done.

Here is my personal technology radar. It helps me to follow up and assess upcoming technologies, to innovate on them and study news things. Despite the fact that the radar is written using impersonal language, it provides value via visibility of used technologies and gained experience curve at my professional life cycle.

How is it made?

Quadrants. The radar is built on topic areas - quadrants. Each quadrant maintains blips and a list of knowledge materials which helps progressing with development.

- **Techniques and Theories**. These include elements of a software engineering process, such as theories (e.g. queueing, functional programming), architecture practices and patterns, experience design, etc.
- **Platforms and Systems**. Things that we build software on top of such as cloud platforms, operating systems, or generic kinds of platforms like distributes or actor systems.
- **Tools and Technologies**. Standalone software components such as databases, software development tools, such as versions' control systems; or more generic categories of productivity tools.
- **Languages and Frameworks**. These include programming languages like Go, Java and Python and frameworks Flutter, React.js and Gin, etc.

Rings. Blips belong to one of the rings on the radar, emphasizing its importance and applicability for my projects.

- Adopt I do have strong confidence for that and use it when appropriate in my projects.
- Trial The technology is under evaluation. I am still testing its limits or learning new things about it.
- Assess Worth exploring and building an understanding of how it impacts my projects.

↦ Hold Proceed with caution. The technology does not have a significant impact.

How do I maintain it?

Every two years, I diligently update the Tech Radar by reflecting on my projects and conducting thorough studies of emerging technologies. The categorization of technologies into different rings is based on subjective assessment, influenced by the significance of each technology to my personal projects and interests. This process involves evaluating the relevance, impact, and potential of various technologies, ensuring that the Tech Radar remains a valuable resource tailored to my specific needs and objectives.

Techniques and Theories

These include elements of a software engineering process, such as theories (e.g. queueing, functional programming), architecture practices and patterns, experience design, etc.

Functional, Typesafe Programming ¦ Adopt

Functional Programming is a declarative style of software development that relies on side-effect-free functions to articulate solutions within the problem domain. Its foundational principles include first-class and higher-order functions, along with immutability. Additionally, composition plays a pivotal role in functional programming, enabling the construction of new components from smaller, reusable elements. Remarkably, this approach is not confined to purely functional languages but can also be effectively applied in imperative ones. Scala serves as a prime illustration, leveraging an imperative runtime while offering data structure implementations that internally safeguard against mutation.

Static type compilers serve as a robust defense against type errors and the detection of undesirable program behavior during compilation. Through the utilization of type classes abstractions and higher-order types, developers can formalize the core domain in algebraic terms, providing a means to "prove" the correctness of algorithms and thus ensure type safety throughout implementation.

Protocols Engineering ¦ Trial ⇒ Adopt

Protocols stand as pivotal components in constructing interoperable distributed systems. They establish the essential communication frameworks necessary for seamless interaction between disparate entities. Among the plethora of protocols imperative to distributed systems are the likes of IP, TCP, UDP, TLS, HTTP/1, HTTP/2, HTTP/3, WebSocket, OAuth2, MQTT, and CDN, each serving specific functions tailored to diverse network requirements.

In addition to their functional attributes, non-functional aspects such as latency play a crucial role in protocol design, particularly in the context of heterogeneous wireless networks. Ensuring efficient transmission and reception of data across varying network conditions demands meticulous consideration of latency, packet loss, and bandwidth constraints. The adaptability of protocols to accommodate these challenges significantly influences the overall performance and reliability of distributed systems, thereby emphasizing the paramount importance of incorporating non-functional aspects into protocol design methodologies.

Distributed Systems | Adopt

Addressing consensus challenges within a network of unreliable processors stands as a fundamental hurdle in distributed computing. Protocols such as Paxos and Raft have emerged as cornerstone solutions tailored to tackle coordination and synchronization obstacles inherent in distributed systems. These protocols facilitate the establishment of consensus, ensuring coherence among distributed nodes despite potential failures or discrepancies.

Complementing traditional consensus mechanisms, Conflict-Free Replicated Data Types (CRDTs) offer a novel approach to achieving strong eventual consistency without the need for explicit synchronization. Mathematically proven to resolve concurrent updates across disparate replicas, CRDTs provide a robust foundation for maintaining data integrity in distributed environments.

The absence of a global clock poses a significant challenge in distributed systems. Approaches such as Lamport and Vector clocks employ partial ordering to navigate temporal discrepancies and establish a coherent sequence of events across distributed nodes. Additionally, innovative techniques like Twitter Snowflake and k-ordered numbers offer decentralized event ordering, enabling efficient coordination and synchronization without relying on a central authority.

By leveraging a diverse array of consensus protocols and synchronization mechanisms, distributed systems can overcome the inherent complexities of unreliable processors and ensure robustness, coherence, and consistency across networked environments.

- CRDT in Erlang https://github.com/fogfish/crdts
- K-ordered unique identifiers in lock-free and decentralized manner https://github.com/fogfish/guid
- Consistent hashing data structure https://github.com/fogfish/ring

Event-driven architectures | Adopt

Event-driven architecture (EDA) represents a software architecture paradigm focused on the generation, propagation, and handling of events, as well as the underlying principles governing their processing. Central to EDA is the concept of asynchronous processing, which serves as a fundamental abstraction necessary for designing resilient and scalable systems. By decoupling components and enabling non-blocking communication between them, asynchronous processing facilitates responsiveness and fault tolerance, making it indispensable in modern system designs.

Additionally, event-driven architectures empower systems to respond dynamically to changing conditions and user interactions, enabling real-time processing and decision-making. This adaptability is achieved through the continuous flow of events, which trigger actions and workflows in a loosely coupled and decentralized manner.

In essence, embracing event-driven architecture enables organizations to build agile, scalable, and responsive systems that can effectively handle the complexities of modern applications and meet the evolving demands of users and stakeholders.

- Go channels for distributed queueing and event-driven systems
  https://github.com/fogfish/swarm

Highly Scalable Systems | Adopt

Scalability is commonly defined as "the property of a system to handle a growing amount of work by adding resources to the system." This essential characteristic underpins the engineering of systems that can dynamically accommodate increasing workloads while maintaining performance and availability. Achieving scalability involves the design and implementation of horizontally scalable architectures, which facilitate the seamless addition of resources to meet growing demands. Additionally, ensuring high availability is paramount, necessitating robust redundancy and fault tolerance mechanisms.

Central to scalability is the practice of workload decomposition, wherein complex tasks are partitioned into smaller, manageable units that can be distributed across multiple nodes or resources. This approach enables efficient parallel processing and resource utilization, thereby enhancing system performance and responsiveness.

Traffic shaping also plays a crucial role in scalability, allowing organizations to regulate the flow of data and requests within the system. By implementing traffic shaping techniques such as rate limiting and prioritization, service providers can prevent bottlenecks and ensure equitable resource allocation, even under heavy loads.

Furthermore, the principles of scalable design and validation must permeate the daily activities of service teams. Continuous monitoring, performance testing, and capacity planning are essential practices that ensure the scalability and resilience of distributed systems. By adhering to these principles, organizations can effectively address the evolving needs of their users and adapt to changing market conditions while maintaining optimal performance and reliability.

Fault Tolerant Systems | Adopt

Software systems inevitably encounter failures over time; it's an inherent aspect of their operation. These failures often manifest inconsistently, and observers typically acquire imperfect information about their occurrence. Such failures are categorized as Byzantine faults, presenting significant challenges to system reliability and performance.

Fault tolerance is the measure of a computer system's ability to operate effectively and maintain functionality in the face of these adversities. It encompasses a range of techniques and strategies designed to mitigate the impact of failures and ensure system resilience. These measures include redundancy, error detection and correction mechanisms, graceful degradation, and failover capabilities, among others.

In essence, fault tolerance is a critical aspect of system design and engineering, particularly in mission-critical environments where system downtime or failure can have severe consequences. By implementing robust fault tolerance mechanisms, software engineers can enhance system dependability and minimize disruptions caused by unforeseen failures.

Security and Privacy of Internet Services | Trial ⇒ Adopt

The increasing adoption of SaaS as an operating model has underscored the paramount importance of prioritizing security and privacy throughout the development process. Cybersecurity is pivotal in mitigating a multitude of threats, including data breaches, theft, and privacy violations. Fortunately, a dedicated cadre of security experts continuously assess and address these risks. They have devised robust controls and benchmark protocols to safeguard against these challenges.

However, leveraging these tools often demands specialized knowledge and effort from software engineers. Compliance with mandatory controls such as those outlined by the Center for Internet Security (CIS) and the General Data Protection Regulation (GDPR) is imperative. Furthermore, integrating security threat automated scanning tools into the deployment pipeline has become a standard practice. These tools serve as proactive measures, running automatically to detect and address vulnerabilities throughout the development lifecycle.

Microservices | Adopt

Microservices have evolved into a foundational design approach for shaping modern system architectures. They offer a methodical framework to distill and encapsulate core business concepts, enabling the evolution of solutions in parallel while promoting uniformity and consistency across the entire system. This architectural paradigm proves particularly advantageous for large enterprises, empowering them to decompose intricate systems into manageable and cohesive units. This facilitates parallel evolutions and enhances scalability, enabling large companies to navigate complex software landscapes with agility and efficiency.

Conversely, for smaller organizations, microservices provide the agility to pivot swiftly and efficiently dispose of components as needed. This flexibility enables small businesses to adapt quickly to changing market dynamics and seize emerging opportunities with nimbleness.

Effective utilization of microservice patterns demands a holistic consideration of various factors. Domain-driven design principles are paramount for defining clear service boundaries aligned with business objectives. Establishing a common data model fosters interoperability and facilitates seamless communication between microservices. Thoughtful interface design is crucial for defining stable and consistent interfaces that facilitate integration and maintainability.

Furthermore, attention to non-functional requirements such as scalability, reliability, and security is essential. Practices such as immutable deployments, where services are deployed as self-contained units, enhance deployment agility and reduce the risk of configuration drift or unintended side effects.

In essence, microservices represent a transformative approach to software architecture, empowering organizations of all sizes to build resilient, scalable, and agile systems. By embracing domain-driven design, adhering to common data models, and prioritizing non-functional requirements, businesses can unlock the full potential of microservices to drive innovation and deliver value to their customers.

- Migrate Your Application To Cloud Age
  https://tech.fog.fish/2017/05/24/migrate-you-application-to-cloud.html

- Microservices https://martinfowler.com/articles/microservices.html

Infrastructure as a Code | Adopt

The paradigm of strictly implying that everything is pure code, developed with a general-purpose programming language, is pivotal in modern infrastructure management. In this approach, any changes to the infrastructure must be defined within the code itself. Declarative Infrastructure as Code (IaC) describes the desired state of the infrastructure without explicitly defining the commands needed to achieve it. Developers utilize generic programming languages to articulate target configurations, which are subsequently materialized by the IaC engine.

The adoption of tools like AWS CDK with TypeScript exemplifies how infrastructure development can be elevated to a typesafe, purely functional, and higher-order process. Leveraging these technologies, developers can design cloud components with increased confidence, ensuring reliability and scalability while maintaining code quality and consistency.

In the realm of software engineering, continuous delivery pipelines have become mainstream, serving as a cornerstone for efficient development workflows. Extending these principles to DevOps represents a natural progression, enabling organizations to seamlessly integrate development and operations activities. Continuous Integration/Continuous Deployment (CI/CD) pipelines for IaC play a crucial role in ensuring quality validation and automation during environment provisioning. By automating the testing and deployment of infrastructure changes, CI/CD pipelines streamline development processes, minimize errors, and accelerate time-to-market, ultimately driving greater agility and innovation within organizations.

IaC enables us with Immutable deployment to address critical aspects of availability and fault tolerance by mitigating the risks associated with software regression. The key principle is to refrain from altering infrastructure or application configurations on running systems. Instead, deploying a new copy as a parallel stack ensures stability and resilience, minimizing the impact of potential failures and downtime.

“Everything is Continuous” | Adopt

Continuous Integration, Continuous Delivery, and Continuous Deployment are the cornerstones of modern software development. The ethos of "Everything is Continuous" epitomizes the philosophy and dedication to maintaining code and services in a perpetually release-ready state. This approach entails the implementation of pipelines to automatically deploy every commit to a feature sandbox environment, with subsequent promotion to production.

Major players in enabling these practices within cloud-native environments include GitHub Actions and AWS CodeBuild. These platforms facilitate seamless integration and automation of CI/CD pipelines, empowering teams to streamline development processes, accelerate delivery cycles, and ensure the reliability and scalability of their software solutions.

Open Source Software | Adopt

Open Source development and contribution to communities.

Technical Hiring ¦ Adopt

Practices to attract techy talents into companies and processes to validate their skills-and-abilities.

Test Driven Development ¦ Adopt

KISS ("keep it simple, stupid") is a guiding principle that, when coupled with test-driven development (TDD), serves as a powerful technique for maintaining control over the quality, addressing regressions, and ensuring the maintainability of software products.

Service Design & Agile Methodology ¦ Adopt

In the rapidly evolving landscape of software engineering, delivering accessible and user-friendly services presents a significant challenge. Service Design encompasses a collection of engineering patterns that span across cloud infrastructure, databases, and protocols. These patterns are meticulously crafted to address the complexities of modern software systems.

Service design processes guide the entire product lifecycle, employing agile methodologies to adapt to changing requirements and facilitate greenfield development. By leveraging these processes, software engineers can navigate the dynamic nature of their projects and ensure the effective delivery of accessible and usable services to users.

Continuing on the theme of service design, Continuous Delivery involves the iterative delivery of small increments, fostering continuous learning and adaptation to customer demand. This approach emphasizes the continuous evolution of products and processes, embracing the principles of "everything" and "everywhere."

In the realm of Agile development, the Kanban style stands out as particularly well-suited for service teams. With its emphasis on real-time communication, full transparency regarding units of work, and a philosophy that acknowledges the potential for change at any moment, Kanban provides an ideal framework for navigating the dynamic nature of service-oriented projects within the context of service design.

Procurement and Technology acquisition ¦ Adopt

Collection of processes to make a selection and on-boarding the technology into the company's portfolio from external sources. It involves licensing, acquiring, technology deep-dives, proof-of-concepts and other tech study activities.

User Experience ¦ Trial ⇒ Adopt

Full-stack polyglot service developers are expected to possess the proficiency to craft and simulate User Experiences effectively. A deep understanding of usability principles not only enhances the development process but also ensures the creation of intuitive and user-friendly internal and consumer-facing applications.

In the realm of software engineering, the command line remains a ubiquitous and indispensable tool. It holds unparalleled significance and familiarity among developers. Thus, it is imperative that we strive to optimize its functionality and ease of access, maximizing its utility and empowering developers to accomplish tasks efficiently.

- Command Line Interface Guidelines https://clig.dev
- Human Interface Guidelines https://developer.apple.com/design/human-interface-guidelines

Observability Trial

Observability, rooted in control theory, is a fundamental property of a system that gauges its capacity to infer internal states from external outputs. To achieve effective observability, applications must actively externalize key events through logs, metrics, and alerts, facilitating comprehensive monitoring strategies. While cloud-based applications can leverage unified solutions like AWS CloudWatch and X-Ray for event collection, other environments rely on tools such as Prometheus, ELK Logstash, ChaosSearch, or ServiceNow.

For visualization and analysis of collected data, Grafana and Kibana serve as powerful tools for building dashboards and visualizing metrics, enabling stakeholders to gain insights into system behavior. Additionally, low-level toolkits like D3.js (https://d3js.org) and Cubism.js (https://bost.ocks.org/mike/cubism/intro/#0) provide options for constructing custom dashboards tailored to specific requirements, further enhancing the observability and diagnostic capabilities of the system.

Linked Data Trial

Linked Data serves as a prevalent method for disseminating structured information, enabling its interconnection and machine interpretation. Typically, data domains are constructed and processed as sets of grounded facts, encapsulating knowledge statements that leverage machine-interpretable vocabularies such as schema.org. These vocabularies provide a standardized framework for defining and organizing data elements, facilitating seamless integration and comprehension by automated systems.

In the realm of representing Linked Data, protocols such as RDF (Resource Description Framework) and JSON-LD (JSON for Linked Data) play integral roles. RDF offers a flexible and extensible model for expressing relationships between resources in a graph format, while JSON-LD provides a lightweight and easily parsable serialization format that aligns with the conventions of JSON, making it particularly suitable for web-based applications. The adoption of these protocols underscores the importance of interoperability and accessibility in the dissemination and utilization of Linked Data across diverse digital ecosystems.

- Method and apparatus for providing edge-based interoperability for data and computations https://patents.google.com/patent/US20140067758A1/en

Well Architected Review Trial

Well-Architected Review is a comprehensive framework that equips cloud architects with the necessary techniques to craft infrastructure that is secure, high-performing, resilient, and efficient, capable of supporting a wide range of applications and workloads. By adhering to these best practices, architects ensure that the designed workload is meticulously prepared to handle production traffic seamlessly.

“Code as Crime Scene” Assess

"Code as Crime Scene" is a methodology developed by Adam Tornhill. It borrows principles from forensic science and applies them to software development to understand and improve code quality and maintainability. The central idea behind "Code as Crime Scene" is that software repositories contain valuable insights into the health, structure, and evolution of a codebase, much like how a crime scene contains clues that can help investigators understand what happened. By analyzing various aspects of a codebase, such as code churn (frequency of changes), code complexity, and code ownership, developers can uncover patterns and trends that reveal areas of high risk, technical debt, or inefficiency.

To apply "Code as Crime Scene" effectively, developers use tools and techniques to gather and analyze data from version control systems, issue trackers, code repositories, and other sources. By visualizing this data and identifying patterns, developers can prioritize areas for refactoring, bug fixing, or improvement, ultimately leading to a more maintainable and resilient codebase.

Overall, "Code as Crime Scene" provides developers with a systematic approach to understanding and improving code quality, enabling them to make informed decisions and drive continuous improvement in software development processes.

Semantic, Knowledge and Ontology Assess

Semantic and Ontology methodologies represent powerful techniques for mitigating the information retrieval complexities inherent in unstructured knowledge. These approaches offer effective solutions to the challenges encountered in the extraction, storage, retrieval, and equivalence matching of facts within diverse production systems spanning eCommerce, Fashion, AppStore, Consumer Behavior, and the Search domain. Leveraged notably by IBM Watson, these methodologies provide invaluable frameworks for structuring and harnessing unstructured data, enhancing the efficiency and efficacy of information processing and decision-making processes across various industries.

Data Engineering Assess

Data engineering is a field within computer science and information technology that focuses on designing, building, and maintaining systems and infrastructure to support the processing, storage, and analysis of large volumes of data. It encompasses a wide range of tasks, including data ingestion, transformation, storage, and retrieval. At its core, data engineering involves managing the lifecycle of data within an organization, from its acquisition or generation to its storage, processing, and analysis. Focus on security measures to protect sensitive data from unauthorized access or breaches and ensure compliance with relevant regulations such as GDPR or HIPAA.

Teletraffic & Performance Engineering Adopt ⇒ Hold

Modern software engineering focuses on empirical approached.

Teletraffic engineering harnesses statistical analysis, queueing systems, and simulations to accurately forecast the capacity requirements of network equipment. This multidisciplinary approach is essential for ensuring the optimal performance and scalability of telecommunications infrastructure. In many cases, service teams complement these techniques with performance engineering—an empirical method that addresses specific challenges identified by various branches of teletraffic engineering.

Analytical models play a crucial role in understanding the dynamics of highly loaded Internet services. By establishing relationships between business requirements and system capacity, organizations can leverage these models as valuable tools for investment planning. This strategic approach enables informed decision-making regarding infrastructure upgrades and resource allocation, ultimately enhancing the reliability and efficiency of telecommunications networks.

Software as a Service | Adopt ⇒ Hold

Suspended development of SaaS solutions

Software as a Service (SaaS) epitomizes a delivery model in which web-based software is hosted and managed by a dedicated service team. Stemming from the realm of cloud computing, SaaS emerged alongside infrastructure as a service (IaaS) and platform as a service (PaaS), revolutionizing the way software is accessed, deployed, and maintained.

Platforms and Systems

Things that we build software on top of such as cloud platforms, operating systems, or generic kinds of platforms like distributes or actor systems.

AWS Cloud Computing | Adopt

AWS Cloud Computing services serve as a foundational platform for delivering workloads in the cloud. This versatile platform caters to a diverse array of workloads, ranging from batch processing and stream processing to online transaction processing, content distribution, machine learning, and beyond.

AWS Serverless | Adopt

AWS Serverless consolidates a myriad of techniques for deploying and executing workloads in the cloud, eliminating the need for managing virtual machines. This encompassing approach includes serverless functions, batch processing, delivery pipelines, storage solutions, and more. A standout feature of AWS Serverless is the extensibility of serverless function runtimes, enabling the integration of custom runtimes to cater to diverse programming needs and use cases.

- Serverless Runtime for Erlang https://github.com/fogfish/serverless

Linux | Adopt

Linux is a versatile open-source operating system renowned for its monolithic kernel architecture. Low-level system development often involves patching the Linux kernel to accommodate specific requirements due to its monolithic nature. Proficiency in Linux kernel internals is crucial for tasks such as driver development, advanced IP networking, and routing.

Originally, application development for Linux was primarily accomplished using the GNU toolchain, which included essential components like the GNU Compiler Collection (gcc). However, in modern times, programming languages have evolved to offer cross-platform implementations specifically tailored for Linux environments, further enhancing the development experience and enabling seamless integration with the operating system's ecosystem.

Containerization | Trial

Containerization effectively tackles challenges related to transparent compute infrastructure utilization, workload co-allocation, and streamlining development pipelines. Despite their continued prominence in

software delivery pipelines, questions have arisen regarding the suitability of containers in cloud environments.

To address this, AWS provides a range of container services, including Elastic Container Service (ECS), Fargate, and Elastic Kubernetes Service (EKS), offering flexibility and scalability for containerized workloads. However, containers have faced criticism for their limitations in deploying storage solutions effectively.

The rise of serverless solutions presents a formidable challenge to the dominance of containers in workload management. Serverless computing offers a compelling alternative by abstracting away infrastructure management and providing on-demand execution of code, thus intensifying the competition for container workloads in the cloud.

Elastic (ELK) stack Adopt ⇒ Hold

Search is invalidated by LLM, AI and Vector DBs

The Elastic Stack is a highly scalable platform designed for developing robust analytics, search, and data visualization solutions. Renowned for its adaptability, the platform has demonstrated exceptional suitability for constructing diverse applications such as data meshes, Geographic Information System (GIS) tools, and Consumer Search applications. Its flexibility and comprehensive feature set make it a preferred choice for organizations seeking powerful and customizable solutions for managing and analyzing their data effectively.

Robotic OS Hold

Robotic OS (ROS) is a versatile and comprehensive framework designed for developing robot software. It comprises a diverse array of tools, libraries, and conventions meticulously crafted to streamline the creation of sophisticated and resilient robot behaviors. ROS not only simplifies the development process but also fosters interoperability and scalability, enabling seamless integration across a broad spectrum of robotic hardware platforms. Moreover, ROS facilitates collaboration within the robotics community by providing standardized interfaces and protocols, fostering innovation and accelerating advancements in the field of robotics.

- https://www.ros.org

Actor Systems Hold

Serverless is considered as the replacement of actor systems.

Actor Systems serve as indispensable platforms for addressing scalability challenges in various domains such as data processing, connectivity, and Internet of Things (IoT). These systems, exemplified by frameworks like Erlang, Akka, and Vert.X, provide a powerful paradigm for building highly scalable and fault-tolerant applications.

However, with the advent of serverless computing, there has been a shift in priorities, leading to a deprioritization of the traditional homogenous actor systems. Serverless functions, which are essentially heterogeneous actors, leverage scalable connectivity fabric provided by cloud providers to achieve similar scalability benefits without the need for managing infrastructure.

Despite this shift, actor systems continue to play a vital role in certain use cases where fine-grained control over concurrency, fault tolerance, and distributed processing is required. Moreover, the principles underlying actor systems, such as message-passing concurrency and isolation, remain relevant and influential in the design of modern distributed systems.

iOS | Assess ⇒ Hold

The priority is shifted towards multi platform development with Flutter

iOS, developed by Apple, is a proprietary operating system and integrated environment tailored for application development exclusively on Apple devices.

Android | Assess ⇒ Hold

The priority is shifted towards multi platform development with Flutter

The Android platform serves as the mobile operating system and development environment for Android devices. Originally conceived by Google in response to Nokia's smartphone initiatives, Android has since become one of the most widely adopted mobile platforms worldwide.

Nokia Smartphone Platform | Hold

The world's first mobile OS for smartphones, Symbian OS, was designed and developed by Nokia as a proprietary solution. The platform served as the foundation for Nokia's family of S60 devices and was also licensed to other manufacturers.

TelCo (5G, LTE, 3G and Tetra) | Hold

5G, the fifth generation of wireless technology, promises significantly faster data speeds, lower latency, and greater network capacity compared to its predecessors.

LTE, or Long-Term Evolution, is a standard for high-speed wireless communication, often referred to as 4G technology, which offers improved performance and efficiency over previous generations.

3G, or third-generation technology, marked a significant advancement in mobile communication, introducing faster data transfer speeds and enabling services such as video calling and mobile internet browsing.

Tetra, short for Terrestrial Trunked Radio, is a digital mobile radio communication system widely used by public safety and emergency services for secure and reliable voice and data communication.

Each of these technologies represents a milestone in the evolution of mobile communication, offering unique capabilities and catering to diverse needs across different industries and sectors.

Tools and Technologies

Standalone software components such as databases, software development tools, such as versions' control systems; or more generic categories of productivity tools.

AWS Cloud Development Kit | Adopt

The AWS Cloud Development Kit (CDK) harnesses Infrastructure as Code (IaC) principles to achieve idempotence, ensuring that regardless of how many times the code is executed, it consistently yields the

same outcome. Developed using TypeScript, a strictly type-safe language, CDK allows developers to precisely declare the target infrastructure. Employing type-safe constructs, each corresponding to an AWS concept, facilitates efficient development. Familiar tools and techniques expedite the process, while compile-time static type checks help mitigate potential errors. Additionally, CDK transpiles TypeScript code into AWS CloudFormation, enabling seamless orchestration of cloud component provisioning.

Relational Database Management Systems | Adopt

Relational databases leverage relational algebra as a foundational framework for modeling data, with the majority of RDBMS employing B+ Tree indexing and SQL for data management tasks. Nevertheless, certain implementations opt for alternative indexing approaches like log-merge or fractals to optimize performance and scalability. Postgres stands as a prominent open-source example of an RDBMS, renowned for its robustness, extensibility, and support for advanced features (e.g. JSONB, full-text search, vector search, etc).

Key Value Datastores (NoSQL) | Adopt

The rapid evolution of scalable Internet services has revolutionized data retrieval requirements, sparking a resurgence in NoSQL technology around 2010. Unlike traditional RDBMS systems, which rely heavily on B+ tree indexing, Key-Value Storages offer a distinct approach, leveraging associative arrays (hash-tables) for low-latency data access. This technology has seen widespread adoption, with platforms like AWS DynamoDB, AWS S3, Redis, Memcached, Coherence, LevelDB, RocksDB, Riak, and MySQL Handler Socket leading the charge.

The absence of relational structures necessitates a sophisticated approach to data modeling. To address this challenge, leveraging Linked Data patterns proves invaluable in defining complex data domains and structuring data in a meaningful way. This enables organizations to efficiently manage and query diverse datasets while maximizing the performance and scalability of their systems.

- Key-value abstraction to store algebraic, linked-data data types | https://github.com/fogfish/dynamo, https://github.com/fogfish/ddb
- Method and apparatus for providing applications with shared scalable caching | https://patents.google.com/patent/US8977717B2/en

Real-time event processing | Adopt

The technology represents a natural evolution of queueing systems, incorporating elements of the publish/subscribe paradigm and asynchronous communication. Event systems have emerged in response to the demand for reactive computing and the proliferation of decentralized systems development. These systems often serve as the most effective approach to scaling data intake pipelines, facilitating real-time processing and analysis of vast volumes of data.

It's crucial to highlight key patterns such as Event Sourcing, Command Query Responsibility Segregation (CQRS), and Inside-out databases within event-driven architectures. These patterns offer valuable strategies for managing data flow, maintaining consistency, and enabling efficient processing of events.

When considering implementations, Apache Kafka, AWS Kinesis, and AWS EventBridge stand out as robust solutions that provide scalable and reliable event processing capabilities. However, traditional queueing systems like AWS SQS can serve as lightweight alternatives to event systems, particularly when event ordering is not critical for applications.

In the realm of programming abstractions, Go channels emerge as fundamental tools for building event processing systems, offering lightweight and efficient mechanisms for communication and synchronization between concurrent processes.

- Inside-out database | https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/
- Command Query Responsibility Segregation | https://martinfowler.com/bliki/CQRS.html
- Go channels for distributed queueing and event-driven systems | https://github.com/fogfish/swarm

Reactive computing ¦ Adopt

Reactive computing can be conceptualized as the client-side counterpart of real-time event processing, where data flows through a series of operations, reacting to changes in real-time. At the core of reactive computing lies the concept of streams (http://srfi.schemers.org/srfi-41/srfi-41.html), which are sequential data structures containing computed elements on demand. These streams enable developers to model and process data flows in a reactive manner, facilitating dynamic and responsive applications.

Frameworks such as Apache Camel, Akka Streams, and RxJS are widely recognized for their support of reactive programming paradigms. These frameworks provide developers with powerful tools and abstractions for working with streams and managing asynchronous data processing.

An important aspect of streams is that they adhere to the monad laws, providing consistency and predictability in how data is transformed and manipulated. By enforcing functional purity, streams abstract away the complexities of data processing, eliminating side effects and facilitating composability and maintainability in codebases.

- The stream monad | https://patternsinfp.wordpress.com/2010/12/31/stream-monad/
- Pure functional stream implementation | https://github.com/fogfish/datum/blob/master/src/stream/stream.erl
- Standard I/O build over streams | https://github.com/fogfish/stdio
- Apache Camel | https://camel.apache.org
- Akka streams | https://doc.akka.io/docs/akka/current/stream/index.html
- RxJs | https://rxjs-dev.firebaseapp.com

Data Meshes and Warehouse ¦ Adopt

Analytics and System of Records (SoR) represent the core use cases for Data Warehouses, historically posing challenges in terms of scalability and management. Decades ago, on-premises Data Warehouses necessitated the adoption of complex community-centric solutions like MapR or Apache Spark to address scalability issues. However, with the emergence of cloud computing, solutions such as Data Bricks (Delta Lake) have revolutionized Big Data Analytics for SoR systems. Leveraging services like

AWS S3, EFS, and RedShift, organizations can now effortlessly establish data warehouses in the cloud, streamlining scalability and management processes.

Moreover, the evolution of technology has transitioned Data Warehouses from traditional warehouses to dynamic data lakes and, more recently, towards the concept of data meshes. This evolution emphasizes the significance of linked data and distributed data mesh architectures in addressing the complexities of modern data management and analytics. By adopting these advanced paradigms, organizations can harness the power of interconnected data ecosystems to drive innovation, efficiency, and insight generation.

- Distributed Data Meshes | https://martinfowler.com/articles/data-monolith-to-mesh.html

Identity and Access Management | Adopt

Identity and Access Management (IAM) serves as a comprehensive framework encompassing policies and technologies designed to guarantee that authorized users within an enterprise ecosystem have suitable access to technology resources. Leading IAM solutions such as AWS Cognito, Auth0, Google Account, OAuth2, and Keycloak exemplify effective implementations of this framework. These platforms offer robust features for managing user identities, authentication, and authorization, ensuring seamless and secure access to resources while maintaining strict control over user privileges. By leveraging IAM solutions, organizations can safeguard sensitive data, streamline user access workflows, and fortify their overall cybersecurity posture.

Graph Databases | Trial

Graph databases constitute a specialized class of storage solutions finely tuned for executing semantic queries rooted in graph theory concepts such as nodes, edges, and properties. Unlike traditional relational models that rely on strict schemas and data normalization to support ACID transactions, graphs facilitate the exploration of associative data sets without the overhead of costly join operations. Furthermore, they offer greater flexibility by relaxing schema definition requirements.

This storage technology has given rise to various applications, including Knowledge Graphs, Intent Graphs, and Consumption Graphs. Knowledge Graphs are instrumental in advancing search solutions by enabling a deeper understanding of relationships between entities. Intent and Consumption Graphs facilitate real-time consumer analytics, enabling organizations to track user behavior and make immediate decisions based on interactions with user experiences. Through the seamless integration of graph databases, organizations can unlock valuable insights, drive innovation, and enhance user experiences across diverse domains.

Batch processing | Trial

When discussing batch processing, our minds often gravitate towards on-premises technologies like Hadoop, Spark, and Flink, which demand extensive infrastructure, dedicated engineering teams, and complex management. However, cloud computing, particularly AWS, has transformed batch processing into lightweight ETL (Extract, Transform, Load) pipelines that operate seamlessly with ad-hoc provisioned infrastructure. AWS Batch stands as a prime example of a service designed to construct and manage batch processing pipelines efficiently in the cloud. This paradigm shift offers organizations the flexibility, scalability, and cost-effectiveness needed to handle batch processing tasks effectively without the burdens associated with traditional on-premises solutions.

Geographic Information Systems | Trial

Many applications, particularly those in environmental domains, extensively process spatial data. These applications encompass the capture, storage, analysis, and visualization of geographical information. Horizontally scalable data meshes (warehouses) for GIS are constructed using the Elastic Stack, enabling efficient management and analysis of spatial datasets. AWS Location Services is an off the shelf cloud alternative. Tile38 (https://tile38.com) is an ultra fast geospatial database & geofencing server. It is an in-memory geolocation data store, spatial index, and real time geofence.

For browser-based web applications, tools like LeafletJS and MapBox are commonly employed to visualize geospatial data, providing interactive and intuitive maps. Hybrid applications leverage technologies such as React Native Maps for similar visualization capabilities across different platforms. However, recent experimentation advises me to focus exclusively on Flutter for building cross platform apps (including GIS domain).

Custom indexing of geospatial data often requires dimension transformation techniques like Z-Curve, GeoHash, or S2 (https://s2geometry.io) geometry. These methodologies facilitate efficient storage and retrieval of spatial information, ensuring optimal performance in spatial data processing applications.

The presence of GIS platforms significantly enhances the capability to develop robust applications. However, it's crucial to resist the temptation of relying solely on a single GIS provider. While major players like MapBox, Here, Google, AWS, and Apple offer valuable tools and services, none of them offers a comprehensive solution. Instead, each provider excels in specific use-cases; for instance, Google is renowned for navigation services, MapBox for its map tiles and customization, Here for GIS expertise, AWS for geofencing capabilities, and Apple for cost-effectiveness. Consequently, effective GIS engineering necessitates an integration effort across multiple vendors, leveraging the strengths of each to create a more versatile and resilient application.

Zero Trust | Trial

Zero Trust is a security framework based on the principle of "never trust, always verify." In essence, it assumes that both internal and external threats may exist within a network and mandates strict verification of every user, device, and application attempting to connect to resources, regardless of their location or network environment. Unlike traditional security models that rely on perimeter-based defenses, Zero Trust operates on the premise that trust should not be granted implicitly based on network location or user credentials alone. Instead, it requires continuous authentication and authorization mechanisms, granular access controls, and robust encryption to ensure that only authenticated and authorized entities can access sensitive resources. This approach helps organizations strengthen their security posture, mitigate the risk of data breaches, and protect against insider threats, making Zero Trust a vital framework in today's dynamic and evolving cybersecurity landscape.

Artificial Intelligence & Machine Learning | Trial

Artificial Intelligence encompasses various disciplines, including advanced web search engines, recommendation systems, and generative tools. My expertise and interest lies in knowledge engineering, which involves organizing concepts within a domain and their relationships. The goal is to enable AI programs to intelligently answer questions and deduce real-world facts. Formal knowledge

representations (ontologies) play a crucial role in content-based indexing and retrieval. An ontology defines the objects, relations, concepts, and properties specific to a domain of knowledge.

Successfully adopting AI necessitates establishing MLOps, short for Machine Learning Operations. MLOps is a paradigm designed to “deploy and maintain machine learning models in production reliably and efficiently”. My objective is to facilitate the creation of machine learning products by leveraging best practices of engineering.

The integration of pre-trained commercial and open-source models has become indispensable in contemporary software engineering, enabling the development of intelligent experiences across various applications. Leveraging established solutions like OpenAI and AWS Bedrock models offers a swift and efficient route to construct and scale generative AI applications. These pre-trained models provide a solid foundation, empowering developers to focus on application-specific functionalities while benefiting from cutting-edge AI capabilities.

Alternatively, for those seeking open-source alternatives, platforms like Llama 2 (https://llama.meta.com/llama2/) and projects like Kandinsky (https://github.com/ai-forever/Kandinsky-3) offer viable options. These open-source initiatives provide flexibility and transparency, allowing developers to tailor models to their specific needs and contribute to the wider AI community. Whether opting for commercial or open-source solutions, the availability of pre-trained models significantly accelerates the development cycle and enhances the overall intelligence of software systems.

Vector Databases & Embeddings Assess

Vector databases are specifically designed to efficiently store and retrieve vector embeddings, which serve as mathematical representations of data in a high-dimensional space. These embeddings encapsulate various forms of content such as images, text, videos, and more, converting them into fixed-length numerical arrays. At the heart of vector database architecture lies the utilization of AI models for content encoding.

One fundamental indexing technique employed within vector databases is the Hierarchical Navigable Small World (HNSW) algorithm (https://arxiv.org/pdf/1603.09320.pdf), belonging to the class of approximate nearest-neighbor search (ANN) algorithms. This algorithm plays a central role akin to the B+ tree in traditional relational database management systems (RDBMS), facilitating efficient search operations within the vast repositories of vector embeddings. In the Open Source, there are the following solutions https://weaviate.io, https://milvus.io, https://github.com/pgvector/pgvector and https://github.com/facebookresearch/faiss. AWS Aurora for Postgres supports vector search out of the box.

The capability to conduct similarity searches within massive datasets stands out as a pivotal feature of vector databases. Such databases find extensive utility across various domains, including but not limited to recommendations, content retrieval, semantic search, retrieval-augmented-generation (RAG), classification, and anomaly detection.

Word2vec, Doc2vec, and http://rdf2vec.org represent open-source solutions tailored for calculating embeddings. These methods provide efficient means to transform textual data into dense numerical

representations, facilitating various downstream tasks such as semantic analysis, clustering, and recommendation systems.

In addition to these open-source solutions, numerous cloud providers offer commercial-grade embedding services. For instance, offerings such as OpenAI Embeddings and AWS Bedrock Titan Text model are specifically engineered to handle large-scale embedding computations efficiently. These commercial solutions often leverage advanced machine learning models and infrastructure optimizations to deliver robust performance and scalability for diverse applications in natural language processing and beyond.

- Blazing Fast Text Embedding With Word2Vec in Golang to Power Extensibility of Large Language Models (LLMs) | https://medium.com/@dmkolesnikov/blazing-fast-text-embedding-calculation-with-word2vec-in-golang-to-power-extensibility-of-large-ed05625dea62
- adapter over various popular vector embedding interfaces: AWS BedRock, OpenAI, word2vec | https://github.com/kshard/embeddings

Distributed Hash Table ┆ Adopt ⇒ Hold

due commercially available NoSQL

A Distributed Hash Table (DHT) is a type of KeyValue storage system deployed across a distributed environment, enabling efficient data storage and retrieval across multiple nodes. Key implementations of DHT technology include platforms such as AWS DynamoDB, Cassandra, and Riak. These systems distribute data across a network of nodes, allowing for scalable and fault-tolerant data management. By leveraging DHT, organizations can achieve high availability, resilience, and performance in handling large volumes of data in distributed environments.

- Dynamo: Amazon's Highly Available Key-value Store https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
- Robust and Efficient Data Management for a Distributed Hash Table https://pdfs.semanticscholar.org/6862/d2099203e4dcd4627ca2128115b4bd3d2fdb.pdf

Peer-to-Peer Networking ┆ Adopt ⇒ Hold

due to L7 protocols (e.g. HTTP/3)

Topology management plays a pivotal role in peer-to-peer networking, ensuring efficient communication and resource utilization among interconnected nodes. Several algorithms have been developed to facilitate effective peer management in such networks. Consistent Hashing, Chord, Kadmelia, and BitTorrent are among the prominent algorithms utilized for this purpose. These algorithms enable nodes to efficiently locate and communicate with one another, contributing to the scalability, robustness, and performance of peer-to-peer networks. Additionally, they help optimize resource allocation and facilitate load balancing, further enhancing the overall efficiency of the network.

Messaging and Queuing Systems ┆ Adopt ⇒ Hold

The rise of event processing de-prioritise importance of queueing systems at the data plane.

Queueing systems are essential components in computer systems, protocols, and I/O operations, enabling the implementation of "waiting" lines to manage tasks and requests efficiently. Beyond mere task management, queues also play a crucial role in enabling reliable asynchronous communication through messaging systems. Platforms such as RabbitMQ, eJabberd, and MongooseIM exemplify

messaging systems, facilitating the seamless exchange of messages between distributed components. By leveraging queues and messaging systems, organizations can enhance the reliability, scalability, and responsiveness of their systems while promoting decoupling and flexibility in communication architectures. Despite the family of queuing systems having got on-hold, AWS SQS is still a prominent technology.

Embedded Queuing Systems | Hold

Embedded Queueing Systems operate similarly to embedded databases, functioning within the application process itself without the need for external processes. These systems are implemented as linkable libraries, seamlessly integrated into the application's runtime environment. By leveraging this technology, applications can efficiently communicate with peers and orchestrate traffic flows without the overhead of managing separate processes.

Several reference implementations of Embedded Queueing Systems exist, including ZeroMQ, nanomsg, and Erlang Simple Queue. These implementations provide developers with versatile tools for building scalable, resilient, and efficient communication pipelines within their applications. Whether handling inter-process communication, distributed messaging, or asynchronous task coordination, Embedded Queueing Systems offer a lightweight and flexible solution for a wide range of use cases.

- ZeroMQ | https://zeromq.org
- Nanomsg | https://nanomsg.org
- Erlang Simple Queue | https://github.com/fogfish/esq

Consumer Search and Discovery | Adopt ⇒ Hold

The rise of vector databases eliminates traditional search solutions, chatbots powered by LLM are a game changer of search and its experience .

Search and Discovery are two concepts frequently intertwined, yet they serve distinct purposes. Discovery facilitates direct access to product catalogs, offering an implicit contextual search triggered programmatically in response to consumer interaction. On the other hand, Search transforms explicit consumer intent into precise or fuzzy pattern match requirements, facilitating the retrieval of relevant items from the catalog.

- Using microservices to power fashion search and discovery | https://engineering.zalando.com/posts/2017/02/using-microservices-to-power-fashion-search-and-discovery.html

Apache Solr | Assess ⇒ Hold

Solr and SolrCloud are open-source search engines developed by the Apache community. These powerful tools are widely utilized for building a variety of search solutions across various domains, including but not limited to fashion, forensic investigations, and more. With their robust features and flexibility, Solr and SolrCloud empower developers to create sophisticated search applications tailored to meet diverse needs and requirements.

Embedded Databases | Trial ⇒ Hold

Databases typically operate within isolated processes, with applications communicating with them via communication sockets to query or store data. Embedded databases, however, are implemented as libraries linked directly to the same process as the application code. While this reduces communication overhead, it also prohibits the sharing of data between multiple instances of the same application.

Embedded databases find utility in standalone applications or serve as storage toolkits in distributed systems, offering lightweight and efficient data management solutions.

Examples of embedded databases include SQLite, LevelDB, RocksDB, Apache Lucene, and ETS, each offering unique features and performance characteristics tailored to different use cases.

Smart Sockets | Adopt ⇒ Hold

Real-time event processing has substitute needs for advanced smart socket concepts.

Smart Sockets represent an innovative technology designed to abstract Layer 4 (L4) and higher communication protocols through a unified interface, resembling either BSD sockets or message queuing systems. Notably, WebSocket, gRPC, NanoMsg, and ZeroMQ are commonly cited examples of smart sockets. These technologies facilitate efficient and seamless communication between distributed systems, enabling real-time data exchange and interoperability across diverse network environments.

Additionally, smart sockets offer numerous advantages, including enhanced performance, scalability, and flexibility in handling various communication patterns. They empower developers to build robust and resilient applications by providing reliable communication channels and simplifying the integration of complex networking functionalities. With their versatility and ease of use, smart sockets play a pivotal role in modernizing communication infrastructures and driving innovation in distributed systems architecture.

LiDAR | Assess ⇒ Hold

Suspended any active development.

LiDAR, short for Light Detection and Ranging, is a cutting-edge technology utilized to create precise 3D representations of physical targets. Its end-to-end data processing pipeline enables the construction of advanced environmental analytics. Particularly prevalent in robotic and autonomous vehicles, including self-driving cars, LiDAR plays a pivotal role in enhancing spatial awareness and navigation capabilities.

In environmental analytics, leveraging LiDAR demands sophisticated pipelines for processing and normalizing raw data. This encompasses tasks such as point cloud storage, indexing, search visualization, and 3D machine learning techniques like feature extraction, segmentation, object recognition, and classification. By harnessing LiDAR technology, organizations can gain invaluable insights into complex environmental landscapes and drive innovation in various fields, from urban planning to natural resource management.

OpenGL ES | Hold

OpenGL ES, short for OpenGL for Embedded Systems, is a specialized graphic hardware acceleration technology tailored for embedded platforms, such as smartphones. It serves as a powerful tool for implementing immersive 3D graphics in various applications, notably including 3D games. Additionally, OpenGL ES finds application in creating gamified user experiences, such as interactive social networking platforms reminiscent of Second Life. By leveraging OpenGL ES, developers can unlock the full potential of embedded systems, delivering captivating visual experiences and innovative user interfaces across a range of mobile and embedded devices.

Languages and Frameworks

These include programming languages like Go, Java and Python and frameworks Flutter, React.js and Gin, etc

Golang | Adopt

Golang (or Go) is a programming language renowned for its versatility in general-purpose, type-safe development across various domains. It excels in building backend services, cloud applications, and command-line utilities with its efficient concurrency model and robust standard library.

Backend Services: Golang is widely adopted for developing scalable and high-performance backend services. Its simplicity, speed, and built-in support for concurrency make it ideal for building microservices, RESTful APIs, and distributed systems. Gin and its middleware is the default selection. However, Go combinator library for building containerized and serverless HTTP services (https://github.com/fogfish/gouldian) has shown its potential.

Cloud Applications: Golang's lightweight runtime and efficient memory management make it well-suited for cloud-native development. It seamlessly integrates with cloud platforms, enabling developers to build resilient and scalable applications for the cloud environment. Thus, Golang is the primary selection for serverless applications.

Command-Line Utilities: Golang's ease of use and powerful standard library make it an excellent choice for building command-line tools and utilities. Its cross-platform compatibility and efficient execution speed ensure that command-line applications written in Golang are fast and reliable. Cobra (https://github.com/spf13/cobra) and https://clig.dev/. GoReleaser (https://goreleaser.com) is the defacto standard solution for automation of cli applications.

Type Safety: Golang's static typing system provides strong guarantees about the correctness of code at compile-time, reducing the likelihood of runtime errors and enhancing code maintainability. Addon libraries (e.g. https://github.com/fogfish/golem) provide various functional abstractions to improve type safety. Despite this, the linter for Golang is a necessary tool - https://staticcheck.dev just does its job (golangci-lint does not support generics).

Concurrency: Golang's native support for concurrency through goroutines and channels enables developers to write concurrent programs with ease, making it particularly suitable for building highly concurrent and parallel systems.

Overall, Golang's simplicity, performance, and concurrency support make it a popular choice for a wide range of development tasks, from backend services to cloud-native applications and command-line utilities. Its growing ecosystem and vibrant community continue to propel its adoption across various industries and use cases.

Flutter | Adopt

Flutter is a versatile cross-platform front-end development framework renowned for its ability to support development across Web, iOS, and Android platforms. The major components of Flutter comprise the Dart platform, the Flutter engine, and the Foundation library. Additionally, Flutter offers a rich set of tools

and widgets, facilitating rapid and efficient development of visually stunning and performant applications across various devices and platforms.

BLOC, which is a predictable state management library for Dart, is the default state management solution for me.

TypeScript ¦ Adopt

For me, TypeScript is a versatile language utilized for Infrastructure as Code (IaC) development and cloud integrations, employing pure-functional and higher-order type-safe techniques. Notably, AWS CDK leverages TypeScript to define target infrastructure, enabling developers to express cloud resources and configurations with clarity and precision.

The language accelerates development by providing robust static type checks at compile time, reducing the likelihood of errors and enhancing code reliability. Examples like Purely Functional and higher-order cloud components (https://github.com/fogfish/aws-cdk-pure) and Security-compliant AWS CDK components (https://github.com/SSHcom/c3) exemplify TypeScript libraries that extend the basic functionality of AWS CDK, offering enhanced features and capabilities for building cloud-native applications.

JavaScript, React & React Hooks ¦ Adopt

JavaScript is only associated with front-end development in my cases, where it is often paired with frameworks like React to accelerate the development of web and hybrid mobile applications.

React is primarily utilized for User Experience (UX) development, emphasizing a pure functional approach to building interfaces. This involves leveraging React in a manner that eschews object-oriented programming paradigms, opting instead for a functional programming style that emphasizes the use of functions over classes.

React Hooks represents a significant advancement in React development, offering a technique for composing high-order components and moving away from class-based approaches. With Hooks, developers can create more modular and reusable components, facilitating the development of micro-frontends and enhancing code maintainability.

In addition to component composition, React Hooks also serve as a powerful tool for abstracting side effects in UX development. Projects like react-hook-oauth2 demonstrate how Hooks can be used to manage complex side effects, such as authentication flows, in a concise and reusable manner.

Overall, React and React Hooks enable developers to build dynamic and interactive user interfaces with a focus on simplicity, modularity, and maintainability. By embracing functional programming principles and leveraging Hooks for component composition and side-effect management, developers can create more robust and flexible UX solutions.

Personally, I've put Redux on-hold as a state management solution. React hooks + context is sufficient for small projects, the large one benefits from BLOC pattern.

C | Adopt

No active development, mainly GCO bindings.

C is a tool for embedded systems, drivers development and Linux kernel development. It was one of the first adopted languages but higher-order language like Go has substituted daily development on C.

C++ | Adopt

C++ old-good but Go is a preferred choice.

Bash, Zsh and Shell scripting | Adopt

Shell scripting and Makefiles serve as fundamental tools for orchestrating heterogeneous workloads. Leveraging these tools helps avoid the need for custom domain-specific languages in continuous integration (CI) systems, enabling the implementation of testable automation pipelines. Their simplicity and versatility make them indispensable for managing complex tasks and streamlining development workflows across various environments and platforms.

Material UI | Adopt

Material UI is a leading library for UI development, offering developers a multitude of advantages including simplicity, extensibility, visually appealing designs, and a comprehensive set of UI components. It serves well for Web (React) and Hybrid (Flutter) development.

Python | Trial

In my perspective, Python stands out as the predominant language for machine learning and data science tasks, often paired with influential frameworks such as TensorFlow or Gensim. These tasks are typically executed within Jupyter notebooks or Databricks environments, enhancing the efficiency and flexibility of the development process.

TensorFlow stands as the de facto framework for constructing machine learning workloads, often paired with its high-level API Keras. Successfully delivering machine learning pipelines necessitates a blend of engineering and data science activities. Leveraging technologies like AWS Batch with GPU support proves instrumental in building robust production-grade ML training pipelines with TensorFlow.

Datalog | Trial

Datalog is a versatile logic programming language that serves as a simplified database query tool. Its primary advantage lies in its ability to efficiently query linked data, graphs, and other recursive data structures. Datalog has demonstrated its power in production systems, particularly in deducing real-time consumer behavior, as a showcase of datastore like Datomic (https://www.datomic.com). The language is supported by multiple runtimes, including Datalog in Erlang (https://github.com/fogfish/datalog), which enhances its versatility and adoption across different platforms. Thanks to its simplicity, Datalog can be seamlessly integrated as a query language into existing storage solutions such as Redis (https://github.com/fogfish/relog), DynamoDB, and the Elastic (ELK) Stack (https://github.com/fogfish/elasticlog), enabling developers to leverage its capabilities within their existing infrastructure.

- Deduct consumer behavior in real-time | https://tech.fog.fish/2015/03/17/real-time-consumer-engagement.html

JSII | Assess

Integrating TypeScript with other languages, such as Golang, often entails a significant amount of configuration and boilerplate code. However, in my recent experience, I've found that for Infrastructure as Code (IaC) development, I lean towards utilizing JSII with TypeScript, particularly for L3 library development.

JSII (JavaScript Interoperability Interface) simplifies the process of integrating TypeScript with other languages, allowing for seamless interoperability while minimizing the overhead of configuration and boilerplate code. This approach streamlines the development of infrastructure components, enabling developers to focus more on building robust and scalable solutions without being encumbered by tedious setup tasks.

By leveraging JSII with TypeScript, developers can harness the expressive power of TypeScript for IaC development while benefiting from the flexibility and interoperability provided by JSII. This combination facilitates efficient and effective development workflows, ultimately leading to the creation of high-quality infrastructure solutions with ease.

GraphQL | Assess

Communities widely promote GraphQL as a modern API Gateway due to its advanced capabilities. Paired with micro frontend techniques, GraphQL greatly facilitates development within distributed teams, offering enhanced flexibility and efficiency in building and managing APIs across various services and applications.

Erlang | Adopt ⇒ Hold

Suspended any active development

Erlang stands out as an exceptional language for the development of scalable, fault-tolerant, and soft-real-time systems. Its unique features and capabilities make it well-suited for a variety of use cases:

Scalability: Erlang's lightweight processes, known as "actors," and built-in support for concurrency and distribution enable the creation of highly scalable systems. It excels in handling large numbers of concurrent connections and processing tasks efficiently.

Fault Tolerance: Erlang's "let it crash" philosophy, combined with its supervision tree architecture, empowers developers to build fault-tolerant systems that can recover from failures gracefully. This makes it ideal for building resilient applications that require high availability.

Soft real-time: Erlang's low-latency message passing and predictable garbage collection make it suitable for soft-real-time applications, where responsiveness and consistency are crucial, such as telecommunication systems and real-time collaboration platforms.

Distributed systems: Erlang's built-in support for distribution enables seamless communication between nodes, making it an excellent choice for building distributed systems. It is widely used in building distributed fault-tolerant storage solutions, distributed databases, and distributed message queues.

Time Series databases: Erlang's performance and fault-tolerance capabilities make it well-suited for building time series databases, which require efficient storage and retrieval of timestamped data points. Erlang-based databases like Riak Time Series leverage these features to provide scalable and reliable storage solutions for time-series data.

Workload management: Erlang's lightweight processes and efficient scheduling mechanisms make it ideal for building workload management systems that can handle high-throughput and bursty workloads while maintaining low latency and high availability.

Overall, Erlang's unique combination of features makes it a powerful tool for building robust, scalable, and fault-tolerant systems, particularly in domains where reliability, scalability, and real-time responsiveness are paramount. Its proven track record in building telecommunications systems, distributed databases, and real-time messaging platforms underscores its suitability for a wide range of demanding applications.

Elixir Hold

Elixir is a possible "modern" replacement for Erlang but its Ruby syntax leaks alot of Erlang abstractions.

Hamler Assess ⇒ Hold

Hamler (https://www.hamler-lang.org) is Haskell-style functional programming language running on Erlang VM.

Scala Trial ⇒ Hold

Suspended any active development

Scala stands as a versatile language ideal for microservice development, offering several advantages over other JVM-based languages. With its expressive syntax and powerful features, Scala has emerged as the primary choice for building microservices in the Java ecosystem. Notably, Scala continues to evolve with advancements in pure functional and higher-order type abstractions, enabling developers to write concise, expressive, and type-safe code. Additionally, Scala's seamless interoperability with existing Java libraries and frameworks enhances its appeal for microservice architecture. Overall, Scala's combination of flexibility, scalability, and modern language features makes it a compelling option for developing robust and maintainable microservices.

My Scala experience was only within the microservice domain, primary building solutions with Shapeless around Finch.

Shapeless (https://github.com/milessabin/shapeless) was a default choice for me that facilitates generic programming in Scala, empowering developers to write code that operates efficiently across a wide range of data types and structures. By leveraging Shapeless, Scala programmers can achieve greater flexibility and expressiveness in their code, allowing for the creation of highly reusable and composable components. With its innovative approach to type-level programming and powerful features like heterogeneous lists and type-safe generic derivations, Shapeless opens up new possibilities for solving complex problems and building elegant, type-safe abstractions. Overall, Shapeless represents a significant advancement in Scala development, enabling developers to write more concise, modular, and maintainable code.

Finch (https://github.com/finagle/finch) is a powerful combinator library in Scala designed specifically for building HTTP services. With Finch, developers can leverage a rich set of combinators to define HTTP endpoints and handle requests and responses with ease. By composing these combinators, developers can create complex routing logic and middleware in a concise and expressive manner, making it ideal for building scalable and maintainable HTTP services. Additionally, Finch provides robust support for JSON serialization and deserialization, enabling seamless integration with modern web APIs. Overall, Finch offers a straightforward and type-safe approach to building HTTP services in Scala, making it a popular choice among developers for web development projects.

Haskell ¦ Hold

Fun but unnecessary complexity.

For me, Haskell is a primary reference to explore functional programming patterns and abstractions.

Java ¦ Hold

Java is only applicable to maintain legacy codebases.

C# ¦ Hold

C# is only applicable to maintain legacy codebases.

Perl ¦ Hold

Per is used to be a king for advanced shell scripting. Nowadays, usage of Go for cli development is a preferred choice.

Octave ¦ Hold

Octave is an open source replacement for Matlab but modern data science is all about Python.

R ¦ Hold

R used to be dominant in data science communities a decade ago.

PHP ¦ Hold

PHP is the language for web app development. Lack of type system(s) makes it outdated in modern development.

React Native ¦ Assess ⇒ Hold

Replaced by the Flutter due to unnecessary complexity and high community fragmentations.

React Native serves as a versatile framework tailored for hybrid application development across the iOS and Android mobile platforms. With its robust set of tools and components, React Native empowers developers to build cross-platform applications that offer a seamless user experience on both iOS and Android devices. By leveraging React Native, developers can ~~streamline the development process~~, reduce code duplication, and reach a broader audience with their mobile applications.

BlueprintJS ¦ Adopt ⇒ Hold

Replaced by the Material UI, not well scalable for mobile.

BlueprintJS stands out as a prominent library for UI development, offering a comprehensive set of tools and components. Renowned for its versatility and ease of use, BlueprintJS has emerged as a top choice among developers, rivaling other popular UI development frameworks such as Bootstrap, Metro UI, and

Dress Code. With its robust feature set, BlueprintJS continues to be a preferred solution for crafting intuitive and visually appealing web applications.

UiKit | Assess ⇒ Hold

Replaced by the Material UI, not well scalable for mobile.

UIKit (https://getuikit.com) is a lightweight and modular front-end framework for developing fast and powerful web interfaces.

Jekyll | Adopt ⇒ Hold

Due to Ruby and inconsistency with Gems, it is hard to maintain a project if ruby is not your primary development environment.

Jekyll is a versatile framework renowned for its efficacy in creating static websites and online project documentations. Leveraging Jekyll empowers users to effortlessly generate and manage web content, making it an ideal choice for projects that prioritize simplicity, speed, and ease of maintenance. With its intuitive structure and powerful templating capabilities, Jekyll simplifies the process of building and updating websites, allowing users to focus more on content creation and less on technical intricacies. As a result, Jekyll remains a popular solution for individuals and organizations seeking a straightforward yet robust platform for their online presence.

Akka & Typed Akka | Trial ⇒ Hold

Unnecessary complexity to compare with Erlang, suspended active development.

Typed Akka represents a robust and type-safe actor system, offering developers a powerful framework for building highly scalable and resilient applications. With Typed Akka, developers can define actors with strong type safety, ensuring greater clarity and reliability in their code. This approach provides an intuitive and expressive way to define actors, making it easier for developers to reason about and manage complex concurrent systems. Additionally, Typed Akka offers advanced features such as supervision strategies and message protocols, further enhancing the robustness and resilience of actor-based applications. Overall, Typed Akka serves as a valuable tool for building fault-tolerant and scalable systems in a type-safe and intuitive manner.