Christopher Smith

https://github.com/cwismif/cistek.it
https://linkedin.com/in/cistek

Email: cistek.it@gmail.com
Mobile: 07974851339

Summary

Versatile software engineer with over 11 years of experience developing applications used by millions. Specialised in AWS, Terraform, CI/CD and delivering high-quality, well-tested solutions.

Experience

- **Sainsburys Plc** — Holborn, London, UK
  DevOps Java Engineer — *Oct 2022 - Apr 2025*
  - **Platform migration project**: Led migration of 30 microservices between two Internal Developer Platforms based on Kubernetes.
  - **Migration benefits**: The Internal Developer Platforms being migrated from was decommissioned and saved the company $30,000 a year in AWS costs. 2 senior engineers saved hundreds of working hours a year.
  - **Continuous Integration/Deployment migration**: Led the CI/CD transformation from Jenkins to GitHub Actions, reducing the code line count by 80%.
  - **Upgraded Primary Order Database**: Planned and implemented an upgrade from AWS Aurora 2 (MySQL 5) to 3 (MySQL 8) the 'Single Source of Truth' database for customer orders. Using a blue/green deployment strategy meant there was no downtime, nor data loss/corruption .
- **Ford UK** — London, UK
  Full-stack Software Developer — *Aug 2021 - Sep 2022*
  - **Refactoring as part of the feature development**: Advocated for clean code, refactoring while developing features to decrease tech debt incrementally.
  - **BDD using Test Con**: Pushed for BDD integration testing as part of the local build.
- **Career Break** — Remote
  Personal Development — *Sep 2020 - Aug 2021*
  - **Family Responsibilities / Personal Development**: Took time to support family members requiring care.
- **Sky** — London, UK
  Senior Software Engineer — *Mar 2020 - Sep 2020*
  - **Cloud Engineering using Iac**: Advocated for best practices, making use of official public modules, tflint, tfsec scans. Frequently needed to assist other engineers with Terraform and AWS issues.
  - **Kotlin non-blocking Performance**: Decreased latencies by 80-50% by using Kotlin and non-blocking I/O libraries.
- **Royal Bank of Scotland** — London, UK
  Software Engineer — *Sep 2018 - Feb 2020*
  - **Ownership over microservice**: Led the design and implementation of a microservice that secured 6 million customer transactions per day.
  - **Designed and implemented Event Driven Java frameworks**: Decreased resource consumption by 50% by using Vert.x and non-blocking I/O.
- **Sky** — London, UK
  Software Engineer — *Nov 2017 - Sep 2018*
  - **RatPack - a non-blocking HTTP Server**: Reduced resource consumption and decreased latency by 50% by using non-blocking I/O that doesn't require a single OS thread to handle a single request. This means many more requests can be served concurrently but with less context switching of those OS threads
  - **Designed and scaled our infrastructure**: Using Terraform and IaC - making use of system design patterns for fault-tolerance. Used circuit breakers (Netflix Hystrix) to limit the blast radius of failures and gracefully handle downstream error
- **Amalytics Ltd** — London, UK
  Software Engineer — *Mar 2016 - Nov 2017*
  - **A small startup of 3-5 engineers**: A single MVP was being developed that ingested accountancy data.

- **FinOps Role**: Reduced AWS costs by 75% by scheduling pausing or otherwise disabling services not in use
- **DevOps Role**: Researched, designed and implemented a few Jenkins nodes for CI.
- **Performance Tuning**: Used JVM profiling tools such as JMeter and Java Flight Recorder to reduce computation time by 95% (caused by frequent GC). Remedies were to prefer primitives over Objects, reuse Objects via Object pooling, not using libraries that result in the heap growing in size.

- **British Airways** London, UK
  Junior Full-stack Software Developer *Mar 2014 - Mar 2016*
  - **Implemented and ELK stack**: Used to determine application user information and usage patterns.
  - **Failover architecture**: Two identical stacks in 2 different data centres - both functioned independently but a LB would perform health checks and forward traffic to a healthy stack.

Education

- **University of Portsmouth** Portsmouth, UK
  Master of Pharmacy (2:2) *Oct 2003 - Jun 2008*

Certifications

- **AWS Certified Advanced Networking - Specialty (In Progress)** AWS
  Core: *VPC, Transit Gateway, Direct Connect, Hybrid Networking, Route 53, AWS WAF* *Expected Autumn 2025*
- **Oracle Certified Professional, Java SE 8 Programmer II** Credential ID OC1671391 - Oracle
  Core Java SE 8 • Streams API • Concurrency • JDBC • Java NIO.2 API *Aug 2017*
- **AWS Certified Solution Architect Associate** AWS
  Core: *EC2, EBS, S3, RDS, DynamoDB •* ***Networking:*** *VPC, LB, Routing, Route53* *Jan 2016*

Programming

- **BDD**: Prefer to write service-level black-box tests to identify requirement gaps early.
- **TDD**: Follow the cycle: Write test, make it fail, write code to pass, refactor.
  - Fast feedback and comprehensive regression tests
  - Influences internal design
  - Keeps code focused and concise
  - Unit testing abstractions helps decouple tests from implementation, making refactoring easier and enforcing cleaner abstractions in the production code
- **Design Patterns**: Regularly use Factory Method, Builder, Observer, and Strategy patterns.
- **Languages**: Java (Spring, Hibernate, RxJava), Kotlin (ktor, http4k), JavaScript (AngularJS, Node.js), Python, Bash.

DevOps

- **Containerized Builds**: Build in containers for portability and consistency with CI environments.
- **Local Build Stages**: Run as many build stages locally as possible: compilation, linting, unit/integration/service tests, security scans, static analysis, and performance tests.
- **Reproducible Environments**: Use containers for local databases, message queues, LocalStack, WireMock, etc., to mirror CI/CD environments.
- **Transition to CI/CD**: Once local builds pass, hand off to remote CI for further validation.
- **Internal Developer Platforms**: Reduce app team workload and cognitive load by enforcing standardisation and reducing duplication. Not every team can have a specialist for every DevOps tool.
- **Technologies**: Python, Bash, Java, Groovy; AWS, Terraform, Docker, Kubernetes, GitHub Actions.

System Design

- **Non-Functional Requirements**: Design systems to be scalable, highly available, reliable, and fault-tolerant to prevent outages.
- **Scaling**: Horizontal: Add more servers to maintain performance. Vertical: Simpler but limited by hardware.
- **Databases**: Choose based on data model/access patterns. Relational DBs (e.g., MySQL) are ACID-compliant but harder to scale horizontally; NoSQL offers variants with different trade-offs.
- **Caching**: Select cache types and strategies based on access patterns and consistency needs.
- **Load Balancers**: L4: Faster, less information; L7: Slower, more request detail.
- **Message Queues**: Improve performance via async delegation and add resilience by buffering and replaying messages.
- **Pub/Sub**: Push events to a topic/channel to fan out data to multiple consumers.