Christopher Smith

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

Email: cistek.it@gmail.com
Mobile: 079-7485-1339

Experience

- **Sainsburys Plc** — Holborn, London, UK
  DevOps Java Engineer — *Oct 2023 - Apr 2025*
  - **Platform migration project**: Led migration of 30 microservices between two Internal Developer Platforms (IDP) based on Kubernetes.
  - **Migration benefits**: The IDP 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**: I planned and implemented an upgrade from AWS Aurora 2 (MySQL 5) to 3 (MySQL 8) the 'Master of Record' 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 2023 - Sep 2024*
  - **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.
- **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 2013 - Mar 2015*
  - **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 — *Oct 2003 - Jun 2008*

Certifications

- **Oracle Certified Professional Java SE 8 Programmer II** — 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 — Serverless: Lambdas — *Jan 2016*

Programming

- **BDD**: Prefer to write microservice black-box tests. It helps me identify any gaps in my understanding of the requirements early.
- **TDD**: Write Test, make it fail, write just enough code make it pass, refactor. This has great advantages in:
  - Granting fast feedback and comprehensive regression tests
  - Influencing internal design
  - Keeping code focused and concise

  Although it can couple the tests with the implementation, which require lots of changes to unit tests when refactoring. Unit testing the abstractions helps decouple and makes refactoring require fewer changes. It also promotes cleaner abstractions in the production code
- **Design Patterns**: I use the factory method, Builder, Observer and Strategy patterns most regularly
- **Languages**: Java (Spring, Hibernate, RxJava), Kotlin (ktor, http4k), Javascript(AngularJS, node.JS), Python & Bash (for scripting).

DevOps

- **The need for distributed system design**: Building in a container is portable and reproduces the CI environments. Runtime dependencies a database instance, MQ, LocalStack, etc can be hosted in a container for any testing.
- **Execute the build on your machine as reasonably possible**: shift left when running build stages - lint, run unit, integration and BDD tests with the aid of containerised dependencies, run security scans, and static code analysis, maybe even performance tests. Build and push the image to CI alongside config.
- **Internal Developer Platforms (IDP)**: Reduce App team workload. Enforce standardised, good practices, reduce duplication of work, reduce the cognitive load on the App team
- **Languages**: Python, Bash, Java, Groovy **Technologies**: AWS, Terraform, Docker, Kubernetes, GitHub Actions

System Design

- **Why the complexity?**: The system design has to meet the non-functional requirements (NFRs) of an application. IT system outages can cause huge disruption. If the systems we depend on are so important, they must be: Scalable, Highly available, Reliable, Fault-tolerant
- **Types of Scaling**: Horizontal scaling: Adding more servers to the system. Systems must maintain performance through horizontal scaling.
  Vertical scaling: Simpler but limited by CPU and memory of even the most powerful machines.
- **Databases**: Analysis of data storage and access patterns determines database technology choice:
  - Relational databases (eg. MySQL): ACID compliant but more complex to scale horizontally
  - NoSQL databases: More flexible schema and easier horizontal scaling, but with various trade-offs
- **Caching**: There many variants of cache and many places they can be used. Access patterns and data consistency are considerations.
- **Load balancers**: Two types exist: L4 LBs are Faster but have less information for forwarding decisions and L7 LBs are Slower but can use client request details in decisions
- **Message Queues**: Provides two key benefits: Improves performance through asynchronous workload delegation adds system resilience by buffering temporary processing fluctuations and allowing the message to be replayed.
- **Content Delivery Networks**: Improves performance by serving data from geographically closer servers to users
- **Pub/Sub**: A single push event to a topic/channel fans out data to multiple consumers