ISSN: 2633-4828 Vol. 5 No.2, June, 2023

International Journal of Applied Engineering & Technology

ADAPTIVE MICROSERVICES DEVELOPMENT IN JAVA AND PYTHON WITH INTEGRATED SECURITY FRAMEWORKS

Jaya Krishna Modadugu[1] Ravi Teja Prabhala Venkata[2] and Karthik Prabhala Venkata[3]
[1]Software Engineer, Saint Louis, MO, USA, 63005
[2]Senior Manager, Software Engineer, Saint Louis, MO, USA, 63005
[3]Senior Specialist, Project Management, Hyderabad, India
[1]jayakrishna.modadugu@gmail.com, [2]raviteja.prabhala@gmail.com and [3]karthik030789@gmail.com
[1]ORCID: 0009-0008-9086-6145, **[2]ORCID:** 0009-0007-7265-212X and **[3]ORCID:** 0009-0001-4977-9006

ABSTRACT
This paper explores adaptive microservices development using Java and Python. The purpose is to improve performance, security, scalability, and development agility in modern applications. Microservices are designed as independent services that communicate through APIs. Java and Python were selected for their complementary strengths. Java provides enterprise stability and strong typing, while Python offers rapid prototyping and flexibility. The study applies the Design Science Research Methodology to design, implement, and evaluate secure microservice architectures. Experimental evaluation measures performance, scalability, and security under controlled workloads. Containers like Docker and orchestration with Kubernetes enable service isolation, auto-scaling, and fault tolerance. Security is integrated using OAuth2, JWT, and encrypted API communication. These mechanisms protect data, validate users, and ensure safe service interactions. Load balancing distributes traffic evenly to maintain high response efficiency. CI/CD pipelines and monitoring tools improve deployment speed and system observability. The findings show that containerized microservices significantly enhance performance and reliability. Security frameworks ensure safe and trusted interactions across distributed services. Kubernetes orchestration provides elastic scalability and maintains application stability during traffic spikes. Python and Java together enable agile development without sacrificing enterprise-grade reliability. Combining these technologies supports faster delivery, lower downtime, and better resource management. Overall, the study demonstrates that adaptive microservices with integrated security provide practical, efficient, and secure solutions for modern cloud-based applications. This approach can guide businesses in implementing resilient, scalable, and high-performance software systems with confidence.

Keywords: *Microservices, Java, Python, Containerization, Kubernetes, Security, OAuth2, JWT, Scalability, CI/CD*

INTRODUCTION
Adaptive microservices development means building software as many small services. Each service runs independently but works together through secure communication. Java and Python are widely used because they support microservice frameworks well. Developers prefer Java for strong structure and enterprise stability. Python is chosen for quick coding and flexibility in new features. Microservices require load balancing, service discovery, and container deployment like Docker or Kubernetes. Security frameworks are integrated to protect data, APIs, and user access. These include authentication, encryption, and real-time threat monitoring. Adaptive systems adjust resources automatically with scaling to handle traffic changes. This makes applications reliable, secure, and easy to maintain. Businesses adopt this approach for cloud environments, faster delivery, and reduced failure risks.

LITERATURE REVIEW
Microservices research shows how software shifts from big blocks to small services. Fraser and Ziadé (2021) explained how Python supports service communication and lightweight APIs. They highlighted containerization and asynchronous calls for handling high loads. Patkar et al. (2022) discussed Python web frameworks like Flask and Django.

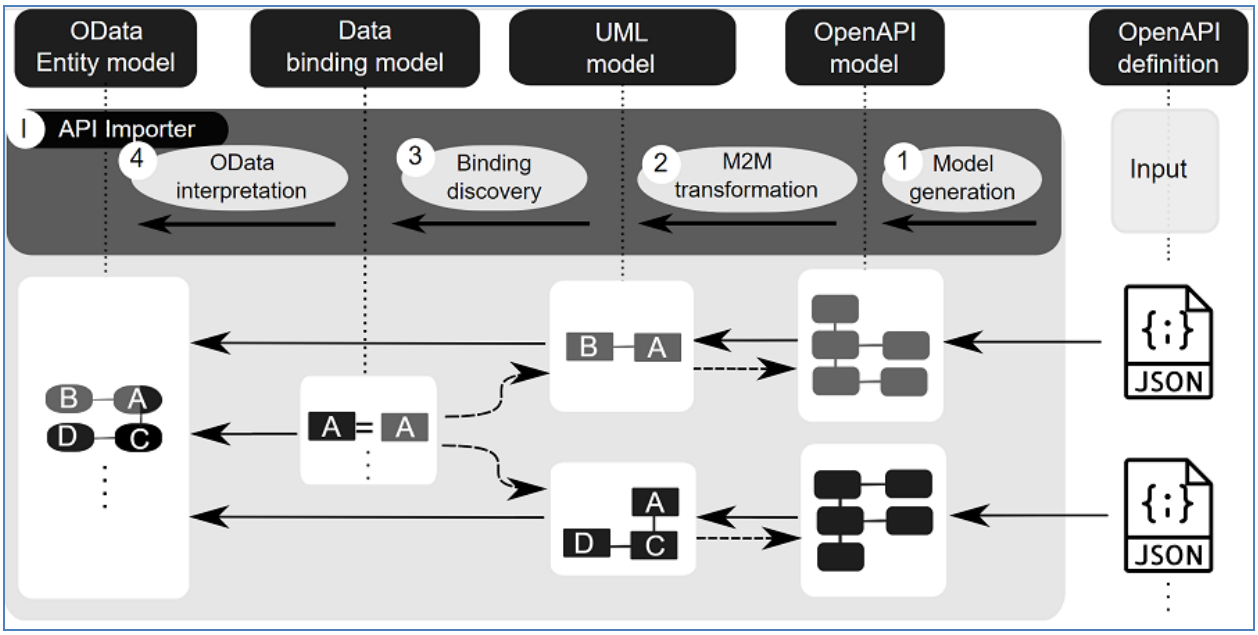

Figure 1: Details of our API discovery and composition process

Source: (Hamza Ed-douibi, 2018)

They explained routing, API management, and how Python simplifies microservice scaling. Benavente et al. (2022) compared monolithic and microservices architectures in detail. They found that microservices offer higher flexibility, faster scaling, and better fault isolation. De Bayser et al. (2022) studied DevOps with microservices in real industry projects. They showed how CI/CD pipelines and Docker streamline updates and service testing. Larsson (2019) focused on Java microservices using Spring Boot and Spring Cloud. He showed how service discovery, load balancing, and Istio provide reliability. Kubernetes orchestration was explained for automated scaling and self-healing deployments. Security concerns were noted across all studies, with emphasis on authentication and encryption. Integration of OAuth2, JWT tokens, and TLS became common practice. Literature confirms microservices improve maintainability, scalability, and deployment in cloud environments. Both Python and Java offer strong ecosystems but serve different developer needs. Python favors quick prototyping while Java ensures enterprise-grade robustness and stability. DevOps integration is critical for faster delivery, monitoring, and automated recovery. Overall, the studies suggest adaptive microservices development benefits depend on correct tooling. Security frameworks remain essential for protecting APIs and distributed communication channels. These findings provide strong technical evidence for using adaptive microservices in practice.

METHOD

The Design Science Research Methodology (DSRM) is most suitable for this study because it focuses on creating and evaluating practical solutions for technical challenges (Haryanti et al., 2022). The research explores adaptive microservices in Java and Python with integrated security frameworks, which requires both architectural design and real-world validation. DSRM supports building artifacts like secure microservice models and testing them through iterative development (Bajaj *et al.*, 2021). Combined with experimental evaluation, it allows measurement of performance, scalability, and security metrics under controlled conditions. This ensures the study not only proposes a conceptual framework but also provides validated, evidence-based outcomes for microservices deployment in modern cloud environments.

RESULT

Performance Gains through Service Isolation and Containerized Deployment

Performance gains in microservices come mainly from isolating services and using containers. Each service runs in its own container, often with Docker technology. This means services are separated and do not interfere with each other. Isolation reduces dependency conflicts that are common in monolithic applications. For example, one service can run Python 3.9 while another uses Java 17 (Åkesson and Horntvedt, 2019). Containers also allow resource allocation, so CPU and memory can be tuned. Kubernetes manages these containers with features like auto-scaling and self-healing. Auto-scaling means that when demand increases, new container replicas start automatically. Self-healing restarts failed services without affecting the entire application. Load balancing ensures traffic is shared evenly between running container instances. This prevents overload on a single service and improves response time. Service discovery helps containers find each other dynamically across distributed networks.

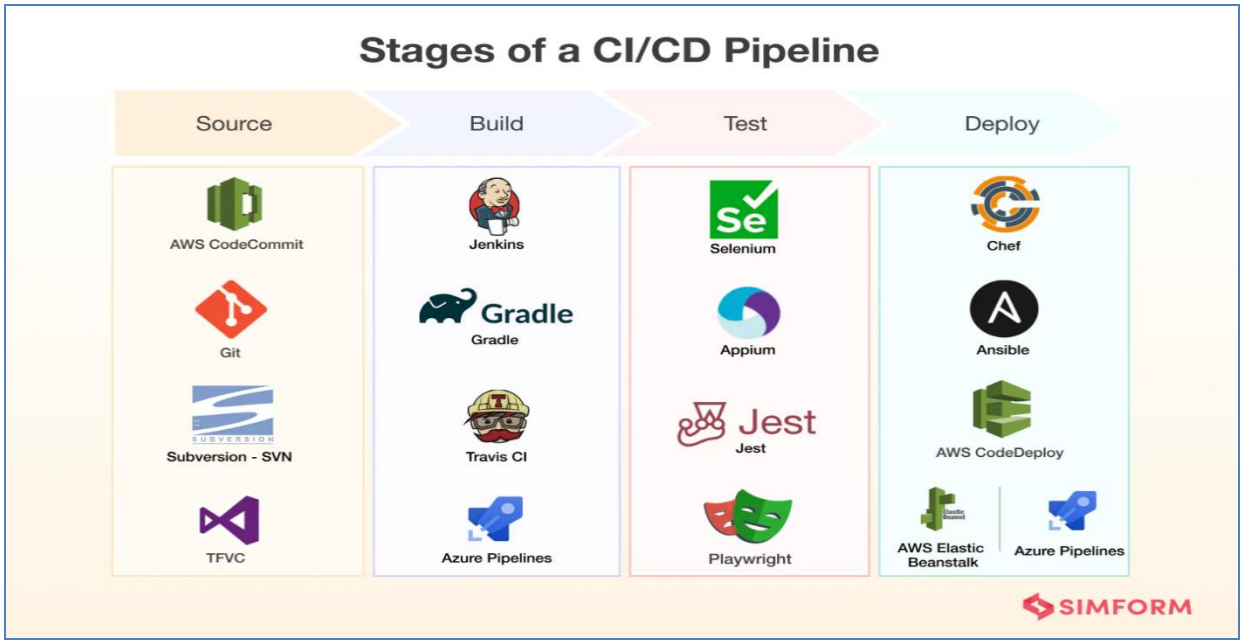

Figure 2: Stages of CI/CD pipeline

Source: (Hiren Dhaduk, 2022)

Performance also improves through CI/CD pipelines which push updates quickly without downtime. Smaller container images are deployed faster, reducing system restart delays (Zampetti *et al.*, 2021). Isolation also enables parallel development teams to build and deploy independently. This reduces bottlenecks and increases delivery speed in large-scale applications. Tools like Istio provide service mesh for monitoring traffic and enforcing policies. Performance is measured with metrics like response latency, throughput, and error rate.

Containerization allows continuous monitoring with Prometheus and visualisation through Grafana dashboards (Martínez-Arellano *et al.*, 2022). All these technical features create systems that are adaptive and resilient. In short, service isolation and container deployment directly improve speed, scalability, and reliability. Businesses benefit from higher performance while maintaining flexible and secure microservices operations.

Security Enhancement using OAuth2, JWT, and Encrypted API Communication

Security in microservices is improved using OAuth2, JWT, and encrypted API communication. OAuth2 is a protocol that controls access with authorisation tokens. It separates resource owners, clients, and servers to prevent direct password sharing. Access tokens are generated and validated by an authorisation server securely.

JWT, or JSON Web Token, is widely used within OAuth2 flows. JWT contains claims in JSON format and is digitally signed. It often uses HMAC or RSA algorithms for signature verification (Gbenle *et al.,* 2022). Tokens include metadata like user identity, roles, and expiration time. Because JWT is stateless, servers validate tokens without storing session data.

This reduces overhead and improves performance for distributed microservices environments. Encrypted API communication is handled using TLS or HTTPS protocols. TLS ensures data between services cannot be read by attackers. Certificates authenticate endpoints and prevent man-in-the-middle attacks in service communication. Mutual TLS adds another layer by validating both client and server (Niemi, Bogdan and Ekberg, 2021). API gateways act as security checkpoints for all microservices calls. Gateways validate OAuth2 tokens and check JWT claims before routing requests. Rate limiting and IP whitelisting can also be enforced at the gateway.

Layer	Function	Tools/Frameworks	Policies/Checks	Logging & Monitoring
API Gateway	Validates tokens	Spring Security, FastAPI	OAuth2, JWT	Audit logs
Request Routing	Routes requests safely	Istio, Envoy	IP whitelisting	Track failures
Rate & Access Control	Limits traffic	N/A	Rate limiting	Alerting
Encryption & Security	Encrypts API traffic	FastAPI, Spring Security	Token encryption	Monitor encrypted traffic

Table 1: Microservices Security Layers

Security frameworks like Spring Security and FastAPI integrate OAuth2 and JWT. These frameworks simplify token validation and encryption handling for developers. Logs and audits are critical for tracking failed or malicious requests. Monitoring tools like Istio and Envoy support encrypted traffic inspection securely. Together, these methods provide layered defense for distributed applications. Using OAuth2, JWT, and encrypted APIs ensures microservices remain secure and trusted (Chatterjee *et al*., 2022).

Scalability Achieved via Kubernetes Orchestration and Automated Load Balancing

Scalability in microservices is strongly supported by Kubernetes orchestration and load balancing. Kubernetes manages containers across clusters using pods as basic deployment units. Pods can scale horizontally by creating multiple replicas of a service. The Horizontal Pod Autoscaler automatically adjusts replicas based on CPU or memory (Yoon *et al.,* 2022). This ensures resources are allocated dynamically according to traffic demand levels. Vertical Pod Autoscaler can also increase container resource limits when required. Cluster Autoscaler scales the underlying infrastructure nodes for high resource needs. Kubernetes uses controllers to maintain the desired state of applications. ReplicaSets guarantee a fixed number of pod instances always remain available. Load balancing distributes incoming traffic evenly across these running pods. Kubernetes Services assign virtual IPs that route requests to healthy pods. External Load Balancers like NGINX and HAProxy integrate with Kubernetes networking. Ingress controllers manage traffic routing with rules for URLs and hostnames (Pessolani, Taborda and Perino, 2021).

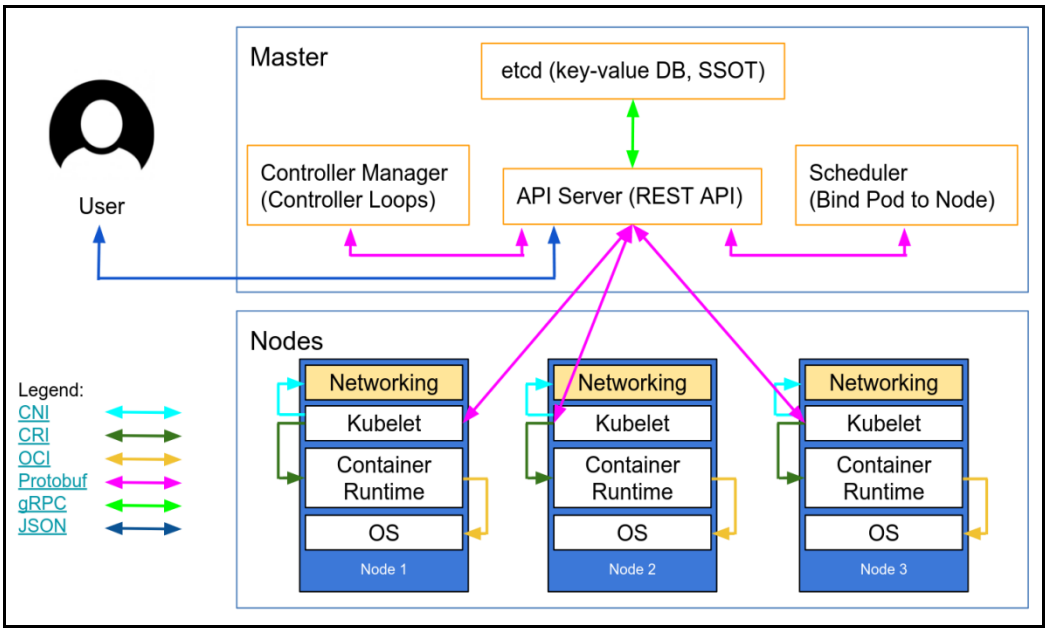

Figure 3: A quick look at different components of a Kubernetes cluster

Source: (MetricFire Blogger, 2020)

Kubernetes supports rolling updates for deploying new versions without downtime. Canary deployments and blue-green strategies reduce risks during major upgrades. Service discovery enables pods to find each other within the cluster. kube-proxy handles network rules that allow communication between distributed pods. Monitoring scalability requires tools like Prometheus for metrics and Grafana for dashboards (Leppänen, 2021). Kubernetes also integrates with Istio service mesh for traffic control and observability. Istio supports advanced routing, retry logic, and circuit breaking for resilience (Wang and Ma, 2019). Together, orchestration and automated balancing provide elastic scalability and stability. This enables microservices to handle unpredictable workloads efficiently in production environments. Businesses gain cost efficiency and reliable performance with Kubernetes-driven scaling solutions.

Development Agility Enabled by Python Flexibility and Java Enterprise Stability

Development agility in microservices is supported by Python flexibility and Java stability (khoirom, 2020). Python provides lightweight frameworks like Flask and FastAPI for rapid prototyping. These frameworks support RESTful APIs, async calls, and JSON serialisation easily. Python's dynamic typing reduces code complexity and accelerates feature development cycles. Dependency managers like pip and virtualenv simplify package handling and environment isolation.

Python integrates seamlessly with Docker for container-based service deployments. It also supports machine learning libraries useful in data-driven microservices (Ueckermann *et al.*, 2020). Java, in contrast, offers enterprise-grade frameworks like Spring Boot and Spring Cloud. These frameworks provide service discovery, configuration management, and distributed tracing features. Java's strong typing improves reliability in large-scale systems with complex logic. The JVM ensures high performance and portability across different operating systems (Lefort *et al.*, 2021). Java microservices commonly use tools like Maven and Gradle for builds. Integration with Kubernetes and Istio enhances Java's orchestration and service mesh capabilities.

Python suits microservices needing quick iteration, experimentation, and AI integration. Java suits core enterprise services requiring strict reliability, transactions, and scaling. Together, both languages can coexist in polyglot microservices environments effectively. API gateways allow Python and Java services to communicate securely

via REST (Kornienko *et al.*, 2021). Security frameworks like Spring Security in Java and OAuthLib in Python are vital.

Feature/Aspect	Python Flexibility	Java Enterprise Stability	Combined Benefit
Rapid Prototyping	Simple syntax, dynamic typing	Strong type system, structured framework	Fast prototyping with reliable architecture
Scalability	Lightweight frameworks (Flask, FastAPI)	Enterprise-grade frameworks (Spring)	Scalable systems with agile deployment
Integration	Easy integration with APIs and libraries	Robust enterprise integrations	Flexible yet stable system connectivity
Maintenance & Reliability	Quick iteration and testing	Strong backward compatibility	Efficient updates with dependable runtime

Table 2: Development Agility Enabled by Python Flexibility and Java Enterprise Stability

CI/CD pipelines like Jenkins and GitLab automate builds, testing, and deployments (Saad, 2021). Monitoring tools like Prometheus and the ELK stack support debugging across services. Combining Python flexibility and Java stability creates agility with robustness. This dual approach supports fast innovation without compromising enterprise-grade dependability and compliance.

DISCUSSION

The findings show how microservices gain strength through isolation, security, scalability, and development balance. Service isolation with containers improves performance by reducing conflicts and allowing independent deployment. Docker and Kubernetes make it possible to manage resources and recover quickly. Security becomes stronger with OAuth2, JWT, and encrypted communication between APIs (Gbenle., 2022). These methods protect user data, secure service requests, and block attackers effectively.

Aspect	Java Microservices	Python Microservices	Security Frameworks (Java)	Security Frameworks (Python)	Adaptability Features	Use Cases
Frameworks	Spring Boot, Micronaut	FastAPI, Flask, Django	Spring Security, Keycloak	OAuthLib, PyJWT, Flask-Security	Auto-scaling, modular design	Banking apps, ERP
Performance	High performance, strong JVM optimization	Lightweight, faster prototyping	Role-based access control	Token-based auth (JWT, OAuth2)	Elastic scaling with Kubernetes	E-commerce platforms
Language Strengths	Strong typing, enterprise ecosystem	Dynamic typing, ease of development	Secure session management	API-level encryption	Multi-language service communication	Healthcare data systems
Security Integration	Deep integration with enterprise security	Flexible, faster integration with libraries	LDAP, SAML integration	OAuth2.0, MFA	Zero-trust architecture	Government services
Deployment & Adaptability	Works well with Docker, Kubernetes	Works well with Docker, Kubernetes	API gateway with rate limiting	Reverse proxy with NGINX	Fault tolerance, CI/CD pipelines	IoT, cloud-native applications

Table 3: Adaptive Microservices Development in Java and Python with Integrated Security Frameworks

Scalability is handled by Kubernetes orchestration, where pods and nodes adjust automatically. Load balancers and ingress controllers spread traffic, keeping services stable under heavy loads (Ejaz *et al*., 2019). Monitoring tools like Prometheus and Istio ensure visibility and faster problem detection. Development agility is achieved by mixing Python's speed with Java's reliability. Python frameworks like FastAPI give quick API building and integration with AI. Java frameworks like Spring Boot give strong enterprise features like transaction handling. Both languages fit into the same ecosystem with CI/CD automation. API gateways help them communicate without security issues or performance loss (Zhao, Jing and Jiang, 2018). Overall, the findings show that adaptive microservices combine flexible coding, strong security, and reliable scaling. This combination gives businesses faster delivery, higher stability, and lower downtime risks. The results confirm that adaptive microservices provide a concrete pathway for modern systems.

CONCLUSION

This study concludes that adaptive microservices in Java and Python offer significant benefits. Service isolation and containerization improve performance and reliability. Integrated security using OAuth2, JWT, and encrypted APIs ensures safe communication. Kubernetes orchestration and automated load balancing provide dynamic scalability under varying workloads. Combining Python's flexibility with Java's enterprise stability enhances development speed and robustness. The findings show that proper tooling, DevOps integration, and monitoring are critical for success. Overall, adaptive microservices deliver faster delivery, higher stability, and secure operations. Businesses can adopt this approach to build modern, resilient, and efficient cloud-based applications with confidence.

BIBLIOGRAPHY

Benavente, V., Yantas, L., Moscol, I., Rodriguez, C., Inquilla, R., & Pomachagua, Y. (2022, December). Comparative analysis of microservices and monolithic architecture. In *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)* (pp. 177-184). IEEE. Available at: https://www.researchgate.net/profile/Isabel-Moscol/publication/367145513_Comparative_Analysis_of_Microservices_and_Monolithic_Architecture/links/6846b3b5df0e3f544f5db9e6/Comparative-Analysis-of-Microservices-and-Monolithic-Architecture.pdf

De Bayser, M., Segura, V., Azevedo, L. G., Tizzei, L. P., Thiago, R., Soares, E., & Cerqueira, R. (2022, April). DevOps and microservices in scientific system development: Experience on a multi-year industry research project. In *Proceedings of the 37th ACM/SIGAPP symposium on applied computing* (pp. 1452-1455). Available at: https://arxiv.org/pdf/2112.12049

Fraser, S., & Ziadé, T. (2021). *Python Microservices Development*. Packt Publishing. Available at: https://sciendo.com/2/v2/download/chapter/9781801079372/10.0000/9781801079372-001.pdf?Token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VycyI6W3sic3ViIjoyNTY3ODUxNywicHVicmVmIjoiNzY0NDg4IiwibmFtZSI6Ikdvb2dsZSBHb29nbGVib3QgLSBXZWIgQ3Jhd2xlciBTRU8iLCJ0eXBlIjoiaW5zdGl0dXRpb24iLCJsb2dvdXRfbGluayI6Imh0dHBzOi8vY29ubmVjdC5saWJseW54LmNvbS9sb2dvdXQvNjdmNTgwNDBmYTQ5NDljOTM5MGNkYTAxMGM3MmZkMDciLCJhdXRoX21ldGhvZCI6ImlwIiwiaXAiOiI2Ni4yNDkuNzkuMjM3In1dLCJpYXQiOjE3NDQxNDU0MDIsImV4cCI6MTc0NTM1NTAwMn0.FPKHYj9JcNXnERGzlad6C5ULRG3zgNJFEq13ksp2V_c

Larsson, M. (2019). *Hands-on Microservices with spring boot and spring cloud: build and deploy Java microservices using spring cloud, Istio, and Kubernetes*. Packt Publishing Ltd. Available at: https://f.letmeprint.ru/259895732-72f61bb0/fragment_10991705.pdf

Patkar, U., Singh, P., Panse, H., Bhavsar, S., & Pandey, C. (2022). Python for web development. *International Journal of Computer Science and Mobile Computing*, *11*(4), 36. Available at: https://www.academia.edu/download/87115279/V11I4202208.pdf

Åkesson, T. and Horntvedt, R. (2019). *Java, Python and Javascript, a comparison*. [online] DIVA. Available at: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1355073

Bajaj, D., Bharti, U., Goel, A. and Gupta, S.C. (2021). A Prescriptive Model for Migration to Microservices Based on SDLC Artifacts. *Journal of Web Engineering*. Available at: https://doi.org/10.13052/jwe1540-9589.20312

Chatterjee, A., Gerdes, M., Khatiwada, P. and Prinz, A. (2022). Applying Spring Security Framework with TSD-based OAuth2 to Protect Microservice Architecture APIs: A Case Study. *IEEE Access*, pp.1–1. Available at: https://doi.org/10.1109/access.2022.3165548

Ejaz, S., Iqbal, Z., Azmat Shah, P., Bukhari, B.H., Ali, A. and Aadil, F. (2019). Traffic Load Balancing Using Software Defined Networking (SDN) Controller as Virtualized Network Function. *IEEE Access*, 7, pp.46646–46658. Available at: https://doi.org/10.1109/access.2019.2909356

Gbenle, T.P., Abayomi, A.A., Uzoka, A.C., Ogeawuchi, J.C., Adanigbo, O.S. and Odofin, O.T. (2022). Applying OAuth2 and JWT Protocols in Securing Distributed API Gateways: Best Practices and Case Review. *International Journal of Multidisciplinary Research and Growth Evaluation*, 3(5), pp.628–634. Available at: https://doi.org/10.54660/.ijmrge.2022.3.5.628-634

Haryanti, T., Rakhmawati, N.A., Subriadi, A.P. and Tjahyanto, A. (2022). *The Design Science Research Methodology (DSRM) for Self-Assessing Digital Transformation Maturity Index in Indonesia*. [online] IEEE Xplore. Available at: https://doi.org/10.1109/ICITDA55840.2022.9971171

khoirom, selina (2020). *Comparative Analysis of Python and Java for Beginners*. [online] Academia.edu. Available at: https://www.academia.edu/download/94738677/IRJET-V7I8755.pdf

Kornienko, D.V., Mishina, S.V., Shcherbatykh, S.V. and Melnikov, M.O. (2021). Principles of securing RESTful API web services developed with python frameworks. *Journal of Physics: Conference Series*, 2094(3), p.032016. Available at: https://doi.org/10.1088/1742-6596/2094/3/032016

Lefort, A., Pipereau, Y., Amponsem, K., Sutra, P. and Thomas, G. (2021). J-NVM. *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp.408–423. Available at: https://doi.org/10.1145/3477132.3483579

Leppänen, T. (2021). *Data visualization and monitoring with Grafana and Prometheus*. [online] www.theseus.fi. Available at: https://www.theseus.fi/handle/10024/512860

Martínez-Arellano, G., McNally, M.J., Chaplin, J.C., Ling, Z., McFarlane, D. and Svetan Ratchev (2022). Visualisation on a Shoestring: A Low-Cost Approach for Building Visualisation Components of Industrial Digital Solutions. *Studies in computational intelligence*, pp.277–289. Available at: https://doi.org/10.1007/978-3-030-99108-1_20

Niemi, A., Bogdan, A. and Ekberg, J.-E. (2021). Trusted Sockets Layer: A TLS 1.3 Based Trusted Channel Protocol. *Lecture notes in computer science*, pp.175–191. Available at: https://doi.org/10.1007/978-3-030-91625-1_10

Pessolani, P., Taborda, M. and Perino, F. (2021). Service Proxy with Load Balancing and Autoscaling for a Distributed Virtualization System. *Unlp.edu.ar*. [online] Available at: http://sedici.unlp.edu.ar/handle/10915/130438

Saad, T.J. (2021). Creating Pipeline and Automated Testing on GitLab. *Theseus.fi*. [online] Available at: http://www.theseus.fi/handle/10024/490105

Ueckermann, M.P., Bieszczad, J., Entekhabi, D., Shapiro, M.L., Callendar, D.R., Sullivan, D. and Milloy, J. (2020). PODPAC: open-source Python software for enabling harmonized, plug-and-play processing of disparate earth observation data sets and seamless transition onto the serverless cloud by earth scientists. *Earth science informatics*, 13(4), pp.1507–1521. Available at: https://doi.org/10.1007/s12145-020-00506-0

Wang, Y. and Ma, D. (2019). *Developing a Process in Architecting Microservice Infrastructure with Docker, Kubernetes, and Istio*. [online] arXiv.org. Available at: https://arxiv.org/abs/1911.02275

Yoon, S., Park, H., Cho, K. and Bahn, H. (2022). Supporting Swap in Real-Time Task Scheduling for Unified Power-Saving in CPU and Memory. *IEEE Access*, [online] 10, pp.3559–3570. Available at: https://doi.org/10.1109/ACCESS.2021.3140166

Zampetti, F., Geremia, S., Bavota, G. and Di Penta, M. (2021). *CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study*. [online] IEEE Xplore. Available at: https://doi.org/10.1109/ICSME52107.2021.00048

Zhao, J.T., Jing, S.Y. and Jiang, L.Z. (2018). Management of API Gateway Based on Micro-service Architecture. *Journal of Physics: Conference Series*, 1087, p.032032. Available at: https://doi.org/10.1088/1742-6596/1087/3/032032

References of Figure

Hamza Ed-douibi (2018). *APIComposer: Data-driven Composition of REST APIs*. [online] Modeling Languages. Available at: https://modeling-languages.com/rest-api-composer/ [Accessed 24 Sep. 2025].

Hiren Dhaduk (2022). *Scalable CI/CD Pipeline Examples: Improve the Development Churn*. [online] Simform - Product Engineering Company. Available at: https://www.simform.com/blog/scalable-ci-cd-pipeline-examples/ [Accessed 24 Sep. 2025].

MetricFire Blogger (2020). *What Is Kubernetes: A Container Orchestration Platform*. [online] Metricfire.com. Available at: https://www.metricfire.com/blog/what-is-kubernetes-a-container-orchestration-platform/ [Accessed 24 Sep. 2025].