Creating a Student-Friendly PaaS Platform: Experiences with Tsuru in Software Engineering Education

Robert Chatley
rbc@imperial.ac.uk
Imperial College London
United Kingdom

Jason Bailey
jason.bailey@imperial.ac.uk
Imperial College London
United Kingdom

Ivan Procaccini
ivan.procaccini14@imperial.ac.uk
Imperial College London
United Kingdom

Zaki Amin
zaki.amin20@imperial.ac.uk
Imperial College London
United Kingdom

Estibaliz Fraca
e.fraca@imperial.ac.uk
Imperial College London
United Kingdom

Abstract

Teaching modern software engineering requires balancing authentic industry practices with accessibility for students. Cloud platforms like AWS and Azure are powerful but can be overly complex for educational contexts. To address this, we developed an internal Platform-as-a-Service (PaaS) using the open-source Tsuru platform. This solution simplifies deployment, enabling students to focus on building and iterating on applications without the need for direct interaction with complex cloud infrastructure, and facilitates administration and access for instructors to support the learning experience.

This paper presents our experiences using this platform in software engineering courses. We discuss challenges, lessons learned, and practical advice for educators aiming to create similar solutions to support student learning.

CCS Concepts

• **Social and professional topics** → **Software engineering education**; • **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → **Software as a service orchestration system**.

Keywords

Cloud, Platform, PaaS, Software Engineering Education

ACM Reference Format:
Robert Chatley, Jason Bailey, Ivan Procaccini, Zaki Amin, and Estibaliz Fraca. 2025. Creating a Student-Friendly PaaS Platform: Experiences with Tsuru in Software Engineering Education. In *33rd ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE Companion '25), June 23–28, 2025, Trondheim, Norway.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3696630.3727239

1 Introduction

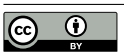
The technical skills needed by a modern software engineer extend far beyond programming. Developing modern systems often involves integrating many different tools and technologies, and building on top of ever more sophisticated products and services provided by vendors. As educators, we want to create an authentic learning experience for our students, introducing them to modern technology that is representative of what they might use in industry. At the same time, we want students to avoid getting distracted by the detail of complex tools in a way that prevents them from seeing the bigger picture and learning overarching principles. This problem presents itself particularly when teaching students to build systems that are deployed online.

Cloud providers such as Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure offer a vast range of resources and services that support engineers in building internet applications. At the time of writing, AWS offers over 200 different cloud services. While these Infrastructure-as-a-Service (IaaS) environments provide engineers with immense power, configuring and managing them can be highly complex and come with a steep learning curve.

A current trend in industry is the adoption of *developer platforms* [6] and *platform engineering* teams [5] that build abstractions in front of cloud providers' offerings, helping product engineers to navigate their complexities. Developer platforms usually provide a templated path to deploy an application to production, removing much of the complexity of configuring cloud resources. This allows developers to concentrate on building new features and iteratively refining their products, rather than the technical complexities of deploying and operating their applications in the cloud.

Some vendors offer these streamlined developer platforms as commercial services, known as Platform-as-a-Service (PaaS). However, we have not found a current commercial PaaS that fits easily with our educational scenario - notably, most require students to enter credit card details when registering, something we cannot ask our students to do.

At our university, we teach software engineering with an emphasis on project-based learning. We teach technical skills for engineering high-quality systems together with iterative methods for product design and delivery [1]. We thread concepts like DevOps [7] throughout the curriculum, so that the associated tools and techniques become something that students use across projects in many different courses, not because they are told to, but because they know they will help them produce better quality results [2].



ESEC/FSE Companion '25, June 23–28, 2025, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3727239

To support this style of teaching, we wanted a deployment solution that balances simplicity of use for students with professional relevance. Our aim was to create an internal PaaS platform which enables students to deploy applications reliably and securely without needing to interact directly with IaaS platforms, or to sign up for accounts with third-party vendors.

In this paper, we report on our experience adapting Tsuru[1], an open-source PaaS, to work with our university's existing Microsoft Azure infrastructure, and using the platform in a variety of courses. We will discuss challenges faced and lessons learned, aiming to give actionable advice and recommendations to fellow educators who might wish to set up a similar system in their own institution.

2 Platform-as-a-Service

Until recently, many commercial cloud providers offered open free usage tiers, allowing users a limited amount of cloud resources without having to subscribe to a payment plan. For teaching software engineering courses that focus on product development, rather than cloud architecture, platforms like Heroku[2] or fly.io[3] provided students with ways to conveniently deploy their applications to the public internet reliably and securely, without having to configure and manage a lot of infrastructure. This meant that students used these platforms in much the same way as engineers at a startup or small enterprise might do, outsourcing configuration of infrastructure and allowing them to concentrate on application development, continuous delivery and customer value [11]. Our experience in teaching agile development with a focus on getting feedback from real users shows that being able to deploy something live to the public internet is important to enable beta users to access it easily. We want to keep the students' focus on developing products incrementally and gathering feedback, rather than complex deployment infrastructure [1].

2.1 Prior Experiences of PaaS for Education

In previous years, we found the developer experience provided by Heroku to be particularly effective for teaching. To deploy an application, a student would need only to a) code it locally in a supported language, b) write a very simple configuration file, c) create a Heroku account, d) run three commands on the command line to authenticate, create the application, and deploy. With this, the application would be deployed live to the public internet.

Free access to tools like Heroku was very useful for teaching and learning, allowing students the experience of using commercial tools while not overwhelming them with infrastructural complexity. Educators at other institutions have reported similar success using freely available public cloud provisions in their courses [8].

However, in recent times, the explosion in cryptocurrency mining led to many enterprising miners creating large numbers of free-tier accounts to use compute resources without paying for them [3]. In light of this, most commercial cloud providers removed or reduced their free offerings and further required each user to register a credit card, thus mandating account verification and automatic billing for any significant use of their resources. While we wanted our students to be able to use these sorts of tools during their studies, we could not ask them to sign up to an external service using a credit card in order to complete their assignments.

2.2 Free PaaS options

There are still some cloud offerings that do provide a free usage tier, but these tend to be much more limited in what they offer technically. For example, Vercel[4] offers serverless function execution within their free tier and Supabase[5] offers an increasing range of functionality centered around a Postgres database, but neither of these give the same functionality as Heroku where a full web application can be deployed. Even where third-party services are available for free, if students register their own accounts independently on the platforms, this can limit the educators' view of what they are doing. Course leaders have no ability to access and administer the accounts on behalf of their students.

Before creating an internal PaaS, we considered other related technologies. The most popular open-source container-based PaaS solutions seemed to be Dokku, CapRover and Coolify[6]. Each of these was trialled in turn, but it was quickly determined that all of these solutions are aimed at hobbyist developers or small teams.

2.3 Requirements for the Educational Context

We have identified specific needs for the educational context for a PaaS solution to complement teaching and management of software deployed by students. For faculty members teaching a course, it is desirable to have holistic control over accounts, teams and configuration options for students. The ability to view application logs and deployment errors for any student's application facilitates troubleshooting by teaching staff, especially when responding to queries asynchronously. If students register their accounts and applications independently on third-party platforms, an educator-focused management paradigm is not feasible.

An essential feature is granular role-based access control (RBAC), allowing students to deploy and manage their own applications in isolation while preventing them from interfering with others. Notably, the alternative PaaS options we considered lacked functionality to arrange users within teams and restrict permissions to specific contexts; educators demand higher privileges than students.

Another requirement is to enable account registration and application deployment without payment details. We could resolve this issue by taking out a subscription to a commercial PaaS on behalf of the university and enrolling all students as users. However, none of the commercial PaaS providers we have looked at have an explicit educational offering, and purchasing a license as an enterprise customer is typically very expensive. A contract with Heroku that would allow all of our students to use the service would cost tens of thousands of dollars per month. This is not a cost that we can bear to support students completing class assignments and projects.

[1] https://tsuru.io
[2] https://herkou.com
[3] https://fly.io
[4] https://vercel.com
[5] https://supabase.com
[6] https://dokku.com, https://caprover.com and https://coolify.io respectively

3 An Internal PaaS Using Tsuru

Beyond the problems and general requirements previously stated, we aimed to create an internal PaaS system with an implementation that supports a simple command-line interface (ideally with a developer experience comparable to Heroku) and integrates with the Microsoft Single Sign-On protocol used by our institution.

After some preliminary work exploring potential options (some of which we discuss in Section 2.2), we settled on forking and customising an existing open-source PaaS, Tsuru. Tsuru offers a lot of the functionality that we were looking for, but it did require some extension and configuration to make it work effectively in our context. Notably, Tsuru's primary target platform is GCP, but we wanted to run it in Azure.

When adopting an open-source tool, there is always concern about its maturity and continued maintenance. Tsuru is used commercially by one of Brazil's largest telecommunications companies, *Globo* [10], which gave us confidence in its reliability and ability to support production workloads.

3.1 Configuration and Implementation

Tsuru (Japanese for *crane*, in reference to the bird's elegance and simplicity) is designed to run on most of the popular cloud platforms using a single account or subscription, and comes with a Google Cloud Platform configuration by default. It supports the standard PaaS notions of `app` (a uniquely identified entity to deploy code against), `service` (e.g. a database or file storage) and `token` (for performing actions in a secure and user-independent way, especially as part of CD practices), and offers a team-based organization of users and apps. The software is designed to run on a Kubernetes cluster and leverages Kubernetes' ingress management, routing and load-balancing capabilities to control the provisioning of computational resources appropriate to each app. Information about users, permissions, applications, etc. are stored in a MongoDB[7] database and the Tsuru command-line interface (CLI) allows a user to manage their applications, platforms, teams, tokens and more.

The typical deployment of an application in Tsuru is as follows: a Tsuru app is created, in association with an individual user or a team and targeting a supported `platform` (e.g. Python, Ruby, Node.js, etc). Deployment of Dockerised applications is also supported. Upon issuing the `deploy` command to 'push' code to the app, Tsuru builds and runs the code, and provisions an ingress router with an application-specific SSL certificate. On a successful deployment, the user can access the live app at the provided URL and further inspect the app's configuration with dedicated commands.

Given our university's existing subscription with Azure, we deployed Tsuru on an Azure Kubernetes cluster. Tsuru's abstractions hide the underlying Kubernetes cluster but at times, the cluster may require direct interaction. A small group of administrators has direct access to Kubernetes, and can adjust settings such as the number of nodes in the cluster. Tsuru makes use of Helm Charts to configure the cluster. Students do not need to interact with Kubernetes directly, but it is beneficial for administrators to understand Kubernetes and associated tools.

[7] https://www.mongodb.com/

Authentication is achieved using OAuth via an *app registration* in Azure, allowing users to authenticate with their university login details via Microsoft Entra. This means that students and faculty can log in to Tsuru using their regular university credentials, with no need to create new accounts. To allow for dynamic provision of database resources, we implemented a MySQL database service provider, and later a PostgreSQL provider, following a standard Tsuru pattern. This allows users to create databases in Azure and link them to applications deployed within Tsuru.

We have made our Tsuru extensions and configuration available to other educators via GitHub: https://github.com/impaas

3.2 Exploring the Developer Experience

Before designing teaching materials around Tsuru, we trialled the developer experience by designing and deploying some new applications supporting administrative tasks within our department. These ranged from student project allocations to reviewing university applications from prospective students. The approach we took was exactly the one a group of students might follow: we created a Tsuru team, assigned ourselves as members of that team, created an application in a particular language, and provisioned database services to store relevant data.

Building several small applications allowed us to go through the complete application development cycle multiple times, and gain familiarity with working with Tsuru from a developer's point of view. We thoroughly explored the space of features that students might use in a class project, for example, manipulating environment variables in production, inspecting app logs, and managing application state. We tested the flexibility of the platform with respect to supporting different technology stacks and architectural choices e.g. deploying a Python API with a separate React frontend, a full-stack application written using Next.JS, and a Java servlet. We also tested triggering deployments to Tsuru from GitLab CI/CD and GitHub Actions workflows, to ensure that it would fit with the DevOps patterns that we teach our students.

By working through these different use cases, we were able to refine our understanding to the point that we could design guided tutorials for students that allowed them to use the platform to complete in-class exercises. Overall, we were pleased to find that the developer experience offered by Tsuru was very close to that offered by Heroku – three simple steps: log in, create app, deploy.

4 Teaching with Tsuru

Our first forays into teaching with the Tsuru platform were in a masters course called *Software Systems Engineering* (SSE) with approximately 60 students who had little previous experience in computing or software engineering. The group of staff running the course were closely involved with setting up Tsuru, and had overall administrator privileges. At this scale, the team could handle any problems through 1:1 interactions during in-person laboratory classes, or asynchronously via online forums.

4.1 Course Design

The SSE course focuses on skills adjacent to programming that are needed to build a modern software system collaboratively. It takes a thin slice through a range of topics: the web, networking, databases,

version control, build pipelines, continuous delivery, consuming data from APIs, and cloud platforms. The course design covers each of the above topics in a practical way, allowing students to build something at each stage through guided laboratory exercises, then integrating the different topics into a more significant project at the end of the course. The PaaS paradigm fits well here as it allows students to build a simple application and deploy it so that they (and their friends) can access it from their laptops and phones. As a lot of the complexity of the cloud is hidden by the simple developer experience of our internal PaaS, we are able to do this early on, and we had students deploying their first applications in the second week of the course.

At this stage, the applications are, understandably, not complex. Typically they are just "hello world" applications, perhaps with the submission of an HTML form and the processing of submitted parameters to yield a results page. We have used Python and Flask because (in our opinion) these give the most accessible starting point for new web developers.

While it may be tempting to have students go deeper and develop larger applications, we believe it is more valuable instead to pause coding at this early point and have them deploy their walking skeleton to production. This approach offers the opportunity to teach more software engineering principles without the need for students to be advanced programmers. Just being able to define a few functions is enough to construct a basic application, and then most of the learning that follows can be around the mechanisms and techniques we use to make changes in a controlled way. We then discuss notions of quality, whereby deployment should happen only if the code is in a good state. We introduce some simple automated checks and tests, demonstrate how these can be orchestrated in a build pipeline using GitHub Actions, and add a final step deploying to Tsuru using our custom GitHub Action.

The students could then take this pipeline forward to use in their end-of-course projects, where they worked over a two-week period in teams of four building an application that showcased the technical concepts from the course.

4.2 Student Experience

Our aim was to provide students with a smooth path to deployment, without convoluted setup, as we had previously experienced with Heroku. Tsuru did provide this with its 3-line deployment experience and all of our students managed to deploy an application successfully in the course of a single laboratory session.

Something that could be considered either a strength or a weakness of the chosen approach is that the students did not acquire an in-depth understanding of the infrastructure behind their application (web servers, containers, networking, Kubernetes, Azure configuration, etc). The advantage of Tsuru's abstraction here is that students do not need to study these concepts in depth or understand them before completing a full cycle of development, and can concentrate on understanding what the different development tools *do*, rather than *how they work*. The downside is that the underlying technology is concealed and students only have a surface-level introduction to infrastructure topics; they cannot gain the mechanical sympathy that might allow them to troubleshoot problems later on when they start to develop more sophisticated systems.

An easy deployment experience enabled students to make small changes to their application and to see the effect of the build pipeline on catching code that did not pass the quality checks, preventing deployment from going ahead, while automating the path to production whenever a commit was good. Having the students set up a complete deployment pipeline early in the course put them in a position where it was easy, and even natural, to deploy a new version of their software after every change that they made through the following weeks of the course. This is a working pattern that we would like to encourage, as it aligns well with the practices of effective professional development teams [4].

We observed that when our students were able to deploy their application easily and reliably to the public internet, this gave them a sense of achievement and excitement about what they could build.

4.3 Instructor Experience

The choice to adopt a self-hosted professional PaaS brought about the great and multi-faceted benefit of full administrative control over permissions and access privileges. After our preliminary investigation into Tsuru's role-based access control model and CLI (see Section 3.2), we were able to implement a simple suite of scripts to create teams and team-bound apps in bulk as the course lab schedule demanded. We assigned team-level privileges not only to the actual team members, but also to course instructors who could then inspect deployments via the Tsuru CLI any time they wanted. This proved especially convenient to assist students with queries on our online forums outside of the allocated laboratory hours: instead of posting partial screenshots of their logs and environment variables, often insufficient if not altogether unhelpful to staff trying to troubleshoot, students could simply reference their application name, which instructors could use to access the live production environment and view application error logs directly. This instructor access aligned well with our educational mission and was a crucial improvement in this sense over the industry standard offered by Heroku and similar proprietary PaaS systems, which generally offer team features and comparable fine-grain permission management only in their paid tiers, and often with costs further growing alongside the number of teams.

As we set up resources, we realised that Tsuru's user model mirrors a generic company structure where users either work independently or in teams. That same model does not serve the coarser organisational structure typical of higher education. In such a setting, a notion of *course*, *cohort*, or even *department* to further group teams together may be desirable for a cleaner separation of concerns and a more precise, convenient and secure management of roles and permissions. This was not a blocker to our immediate needs, and we managed to successfully work around this limitation by using consistent naming conventions for `apps`, `teams` and `tokens`, but given our plans for a wider adoption of Tsuru (see Section 5.3) we believe this could be an area for future improvement.

4.4 Notable Technical Challenges

Tsuru has a powerful, but complex, permissions model. Careful thought must be given to configuring permissions for students. It took us some time to find the right set of permissions for students in

a class that gave them enough power to complete their assignments without being able to affect resources owned by other users.

Another aspect of Tsuru to consider from an administrator point of view is its internal use of Nginx [9] for ingress routing. Nginx has its own configuration, and we soon realised that this adds an extra layer of complexity that is entirely hidden from the CLI. For example, the default Nginx configuration shipped with Tsuru silently ignores large HTTP headers, causing unexpected issues with applications that rely on these. Such issues proved difficult to identify as they were not reflected in application error logs, and fixing them required editing Nginx configuration templates.

5 Discussion

Reflecting on our experience with Tsuru, the largely positive outcomes also came with challenges and costs, which we detail below.

5.1 Operating Costs

While the Tsuru software itself is free, running it on Azure does incur costs. With our setup, the baseline cost for running the platform with no applications deployed is approximately 12 USD per day. Expanding the cluster to host approximately 50 applications concurrently took the cost to around 25 USD per day. Running at this capacity permanently would yield a monthly bill of under 1000 USD – significantly lower than the estimates we made for commercial PaaS agreements. In fact, our overall cost was considerably lower than this as we were able to scale back the size of the cluster once assignments were complete.

The Kubernetes cluster can be configured to automatically scale to match load. This is generally a helpful feature, but we found it helpful to put a limit on the number of nodes, which put a predictable ceiling on the potential cost. However, at busy times and if the cluster is over capacity, some students might not be able to deploy successfully; in an academic environment, this seems preferable to uncontrolled costs.

5.2 Lessons Learned

Tsuru effectively abstracts the complexities of the cloud. It allows students to deploy their software to the public internet without needing a deep knowledge of the cloud, containers or Kubernetes. Running a PaaS platform ourselves removes the need for students to sign up to third party vendors with their own credit cards. While the cost for hosting our platform is not zero, we were able to manage resources effectively and the overall cost seems reasonable for the size of cohorts we were teaching.

Using a PaaS enables students to experience a quick end-to-end path to production. Being able to deploy a real application live in a short timeframe proved very rewarding for our students. An internal platform enables instructors to support students better. A key example of this is that instructors can easily access logs for a student's application to help them debug problems, which is difficult when students use third party services to deploy.

5.3 Extending to Wider Contexts

The next stages are to use the platform in courses run by other staff in the same department, with larger cohorts, and then more widely by staff and students outside the computer science department. At the time of writing, we are beginning a DevOps course for a cohort of 250 undergraduate students that runs over 2-3 weeks and involves deploying a web application through an automated build pipeline. Having ironed out some initial difficulties during the masters course, we are confident that Tsuru should work well for this course, although we expect to need to scale up the capacity available in the Kubernetes cluster for the duration of the course.

In order to fit with our undergraduate computer science curriculum, the DevOps laboratory exercise uses a template application written in Java. Our installation of Tsuru was not initially set up to support Java applications, but Tsuru offers the ability to add support for different languages (e.g. Java, Go, PHP, and more) by adding *platforms*. For supported languages, this is a matter of an administrator running a single command. It is worth noting however that a Java application consumes more resources than a similar application written in Python or Go, so it will be important to monitor resource usage as more applications are deployed.

At the end of the year, we plan to use Tsuru for our *Designing for Real People* (DRP) project-based course [1] with a large cohort of approximately 60 groups. The student groups are asked to develop digital tools, typically web or mobile applications, to solve a real-world problem. We do not give the students any specific constraints in terms of technology choices, so at this stage students may ask for more platforms to be added to Tsuru. It remains to be seen whether there are any limits on this technically, but there is perhaps a concern that if a language is supported by the platform, instructors should also be able to assist students using it. Given the rate of change of web development technology, this may well not be possible, so it may be the case that we can provide better education if we restrict the range of supported languages.

The final stage of rollout is to allow use of the platform beyond the Department of Computing. We have conducted an initial trial with the Department of Bioengineering which offers a software development course for its own students. This course teaches Java programming, and covers web programming with servlets. Initial trials showed that they were able to deploy easily to our Tsuru instance once Java was configured. We added the students and course leader as users on the platform, but nothing further was needed, given that authentication was being handled centrally using university login credentials.

Our experience so far shows that our internal PaaS built with Tsuru has improved the experience of both educators and students. As software engineering cuts across many different subject areas in the modern world, we hope to expand use of the platform across the university in future, supporting both specialist computer scientists and developers from other disciplines.

Acknowledgments

This project was completed in collaboration with the Technology Office within Imperial's central ICT organization, particularly Richard Howells and Nelson Cerqueira, and supported by funding from the university's *Digital Innovation Fund*. A lot of the initial prototyping work on setting up Tsuru and adapting it to work in our environment was done by a group of our undergraduate students: Aaryan Dharmadikari, Aaryan Purohit, Sachin Wadhwani, Ajay Mittal, Thom Hughes and Rushil Ambati.

References

[1] Robert Chatley, Tony Field, Mark Wheelhouse, Carolyn Runcie, Clive Grinyer, and Nick de Leon. 2023. Designing for Real People: Teaching Agility through User-Centric Service Design. In *Proceedings of the 45th International Conference on Software Engineering: Software Engineering Education and Training* (Melbourne, Australia) *(ICSE-SEET '23)*. IEEE Press, 11–22. https://doi.org/10.1109/ICSE-SEET58685.2023.00007

[2] Robert Chatley and Ivan Procaccini. 2020. Threading DevOps Practices through a University Software Engineering Programme. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 1–5.

[3] Catalin Cimpanu. 2021. Crypto-mining gangs are running amok on free cloud computing platforms. https://therecord.media/crypto-mining-gangs-are-running-amok-on-free-cloud-computing-platforms. Accessed: Jan 2025.

[4] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations* (1st ed.). IT Revolution Press.

[5] Camille Fournier and Ian Nowland. 2024. *Platform Engineering*. O'Reilly Media.

[6] Gregor Hohpe. 2024. *Platform Strategy*. LeanPub.

[7] Gene Kim, Patrick Debois, John Willis, and Jez Humble. 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press.

[8] Zheng Li. 2020. Using public and free platform-as-a-service (PaaS) based lightweight projects for software architecture education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* (Seoul, South Korea) *(ICSE-SEET '20)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3377814.3381704

[9] Will Reese. 2008. Nginx: the high-performance web server and reverse proxy. *Linux J.* 2008, 173, Article 2 (Sept. 2008).

[10] Jonathan Reimer. 2025. Open Source Alternatives to Heroku. https://www.opensourcealternative.to/project/tsuru. Accessed: Apr 2025.

[11] Rehmana Younis, Mansoor Iqbal, Khalid Munir, Muhammad Aaqib Javed, Muhammad Haris, and Saad Alahmari. 2024. A Comprehensive Analysis of Cloud Service Models: IaaS, PaaS, and SaaS in the Context of Emerging Technologies and Trend. In *2024 International Conference on Electrical, Communication and Computer Engineering (ICECCE)*. 1–6. https://doi.org/10.1109/ICECCE63537.2024.10823401