Research Project Exhibition 2025

M.Sc. in Software Solutions Architecture, and the M.Sc. in DevOps,

TU DUBLIN
OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

Technology Ireland ICT Skillnet

FOREWORD

Today's Research Project Exhibition is a day presenting great achievement by our taught Master's students in our M.Sc. in Software Solutions Architecture, and the M.Sc. in DevOps, Programmes. The students now presenting their research projects started their Masters programme journeys in January 2023. While it seems like only yesterday, yet so much has happened since then.

Those graduating today, started as students of the Department of Computing which was then part of the School of Science and Computing. You will graduate later this year as some of the first graduates to complete a programme in the new School of Enterprise Computing and Digital Transformation. This new school is part of a new Faculty of Computing, Digital and Data and represents a major investment by TU Dublin which sees Computing, Data Science and Digital Transformation as major activity areas for now and the future

As a School of Enterprise Computing and Digital Transformation we are using this opportunity to say that we are very proud to have you as graduates. Your dissertation work, which is at the cutting edge of Technology and Computing as it is practice in industry, represents some of the best work produced by students in this School. As a new School we are, engaged on an extensive consultation process around our mission, vision and activities and we welcome any comments, feedback and input on how our activities as they are anow and as they should or could be.

These programmes are being taught here in TU Dublin but they are collaborative programmes where we have joined with industry in defining and scoping these programmes. Industry needs in these programmes were refined and proposed by Technology Ireland Skillnet and their ICT employer partners identified the market need for these programmes, TU Dublin, Computing responded, co-designing an industry list of requirements into an academic Masters programme.

For the Software Architecture programme we are proud to partner with the International Association of Software Architects (IASA) and their Irish partners the Irish Computer Society (ICS) in offering this programme. It should be noted that this programme is recognised by the IASA for membership of the association.

Today we have a new cohort starting their journey as well as a cohort moving to the final part of their journey. They have plenty to learn but they too will soon reach this point. Creating and passing on knowledge is the duty of a University and doing this in fast moving Technological Fields relevant to industry is the particular mission of a Technological University.

We hope you enjoy the project symposium.

Finbarr Feeney PhD / Head of School
School of Enterprise Computing and Digital Transhformation
OT Bhaile Átha Cliath / TU Dublin - Tallaght Campus
D24 FKT9
Ireland

M.Sc. in Software Solutions Architecture, and the M.Sc. in DevOps

RESEARCH PROJECT SYMPOSIUM AGENDA

24th January 2025

14:00 - 14:15 Opening Addresses

Gary Clynch, School of Enterprise Computing and Digital Transformation, TU Dublin

Jimmy Doody, Head of Discipline, School of Enterprise Computing and Digital Transformation, TU Dublin

Sean McHugh, Head of Discipline, School of Enterprise Computing and Digital Transformation, TU Dublin

Dr. Barry Feeney, Head of School of Enterprise Computing and Digital Transformation, TU Dublin

14:15 - 15:00 Speaker - Dave Egan, Infineon Technologies

The origin of SaaS

15:00 - 15:45 Speaker - Sooraj Shajahan, Multi-Cloud Architect, Cloudera

Exploring Kubernetes Multi-Tenancy & SaaS with GenAI

15:45 - 17:30 Poster Presentations

Technology Ireland ICT Skillnet

TU DUBLIN
OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

Online MSc in DevOps

With most technology organisations moving their delivery platforms to a DevOps approach the shortage of people with cross sectional skills in DevOps is now acute. Developed by industry as a direct response to this need this first-ever Master's degree in DevOps aims to fill these important talent gaps and give credit, recognition and credibility to technologists working in this field.

The advantages of Development Teams and Operations Teams collaborating to improve the delivery of technology solutions has meant a rapid adoption of DevOps approaches to the Software Development Lifecycle. Closely associated with Lean and Agile concepts in enhancing the delivery of technology solutions, the DevOps approach has impacted very rapidly on the Technology industry.

Most existing DevOps 'specialists' grow or develop into their role with no formal standards or certification, and a modicum of training in the actual practice of cross functional DevOps practices. They may already be experienced, highly skilled, competent and high performers in their own field of Software Development, Computing, IT Management, or Quality Assurance but they can lack the knowledge and understanding of the other cross functional disciplines they now find themselves working with daily. Understanding not only the technical, but also the business and human factors at play during the high pressure demands of modern software delivery processes, is essential in the modern discipline of DevOps.

Award Level

There are two phases to the award. Candidates are registered for the full Masters of Science in DevOps Level 9 degree (90 credits) however candidates may opt to exit the programme on successful completion of the first three semesters with 60 credits and receive a Level 9 Postgraduate Diploma in DevOps (60 credits). Please note exit awards are at the discretion of the college and no refund of fees will be due.
The award structure will place greater emphasis on continuous assessment, practical and project work rather than on formal examinations. In fact there are only 2 modules that carry an actual exam.
The aim is that participants will gain a deep understanding of the topics and content covered, and be able to demonstrate this acquired knowledge as proven competence in tests and exercises drawn from practical "real life" DevOps scenarios.

Programme Delivery

The programme will start with a 3 day workshop which will involve all participants being physically present. This is seen as important to facilitate networking, experience sharing and group learning.

It is expected that lectures will be delivered one evening per week in term time and every 3-4 weeks there may be a requirement to hold lectures twice in that week. There will also be a requirement to attend one on-campus day at the end of each semester.

Lectures will be streamed live from TU Dublin (Tallaght Campus) i and will be available for download and offline viewing.

Semester 1: Introduction to DevOps

Human and Organisational Issues	Software Development Methodologies
• Lean and Agile movements and methods • Assess and evaluate organisational design and culture to facilitate DevOps style development, deployment and support • Develop and manage global multi-disciplinary teams including an understanding of the cultural and practical issues which arise • Be able to form, lead and develop teams • Assess competence, accountability, responsibility, norms and operational management • Collaboration, negotiation and partnering • Managing the Future - Creating a readiness for organisational change, organisational development and change management	• Technical implications of DevOps – the philosophy, the history, the SDLC, Lean, Agile Manifesto, continuous feedback and learning • Change, Source, Defect Control Systems, Examination of major industry implementations (e.g. Atlassian, VSTS) • Code Promotion • Code Synchronization • System Debugging • Software QA • Automated Testing • Software Security Vulnerability Management • Software Telemetry and Monitoring • Feedback and Learning

Semester 2: DevOps Fundamentals

Business Technology Strategy	IT Infrastructure Fundamentals for DevOps
• The Business Case for Agility and DevOps • Lean/Agile management/methods/frameworks (SAFE) • Product road maps, pipelines, backlogs, valuing new features and technical debt • Business case development and risk assessment • Creation/management of multi-annual business plans • Financial Management of Product and Technology life-cycles • Project Management and Methodologies • The end of the monolithic project • Designing for agility and value • Challenges for DevOps • Regulated Software • Impact for Customers of DevOps approach	• Automation of Infrastructure • Task and Process automation languages • Advanced System Administration • Software Security • System Hardening • Policies and implementation • Virtualisation • Containerisation • IT Network and Infrastructure Protocols • IT Network Monitoring • Continuous Deployment • Cloud Computing Concepts • Infrastructure as Code

> *Understanding not only the technical, but also the business and human factors at play during the high pressure demands of modern software delivery processes, is essential in the modern discipline of DevOps.*

Semester 3: Advanced DevOps

Advanced IT Infrastructure for DevOps	DevOps in Practice
• Architectural Design to support DevOps • The DevOps supply-chain and PLM relationship • DevOps in the Public Cloud • Comparative Analysis of Cloud Offerings • Cloud Scalability and Elasticity3 • Load Balancing • Virtualisation Automation • Provisioning and Orchestration • Software Configuration Management • Software Provisioning Management • Security in the Public Cloud • Degrading systems gracefully • Chaos Monkey • Server-less Compute in the Cloud	• The DevOps paradigm/pipeline in practice requirements • Develop Continuous Integration/Test/Deployment Release management • Monitor and Learn • Feedback and Iteration • Detailed DevOps Case Study of the technical and human experiences of typical practitioners, e.g. - Google SRE (Site Reliability Engineering) - Intercom (Customer Messaging Platform)

Semester 4: DevOps Research

Research Methods	Research Project
• Academic Writing • Qualitative and Quantitative research • Surveys • Statistics	• Applied piece of Research in DevOps area • Encompasses a Proof of Concept/Prototype • Supplements DevOps Theory knowledge *This is an opportunity for students to carry out a piece of work which is at the cutting edge of the field and explores in depth a feature or element of that field. It is perfectly feasible, and there are many examples of this, for students to carry out their research project on a piece of work of direct relevance to their company or organisation. The academic team in TU Dublin (Tallaght Campus) have deep industry experience and have supervised and developed MSc. projects which explore business values, infrastructure automation and DevOps projects with real industry relevance.*

Technology Ireland ICT Skillnet

M. Sc. Applied IT Architecture (online)

In conjunction with Irish Computer Society and accredited by International Association of Software Architects. A Technology Ireland Skillnet funded programme. This programme is 80% online with two days per semester attendance required.

M. Sc. Computing with DevOps (online)

This programme was designed in conjunction with leading ICT companies such as Microsoft, Fidelity, IBM, Ericsson who form the Technology Ireland Skillnet. This programme is 80% online with two days per semester attendance required.

Non-Standard Applicants

Note for interested applicants: Next intakes for these Skillnet programmes set for January 2019. Standard admissions requirements include a relevant bachelor's degree at honours level. It is recognised that there are experienced and skilled potential participants for the programme who may not fit the standard entry profile. A non-standard admission process is available here which can be based on prior experiential learning and/or qualifier modules. These qualifier modules can be taken from September 2019 for admission in January 2019. Contact bfeeney@it-tallaght.ie or mhendrick@it-tallaght.ie for more information.

Accreditation of Master of Science in Applied IT Architecture by IASA

Technology Ireland ICT Skillnet

The TUDublin (Tallaght Campus) M.Sc. in Applied IT Architecture is the first of its kind in the world to be developed based on the IASA Five Pillars.

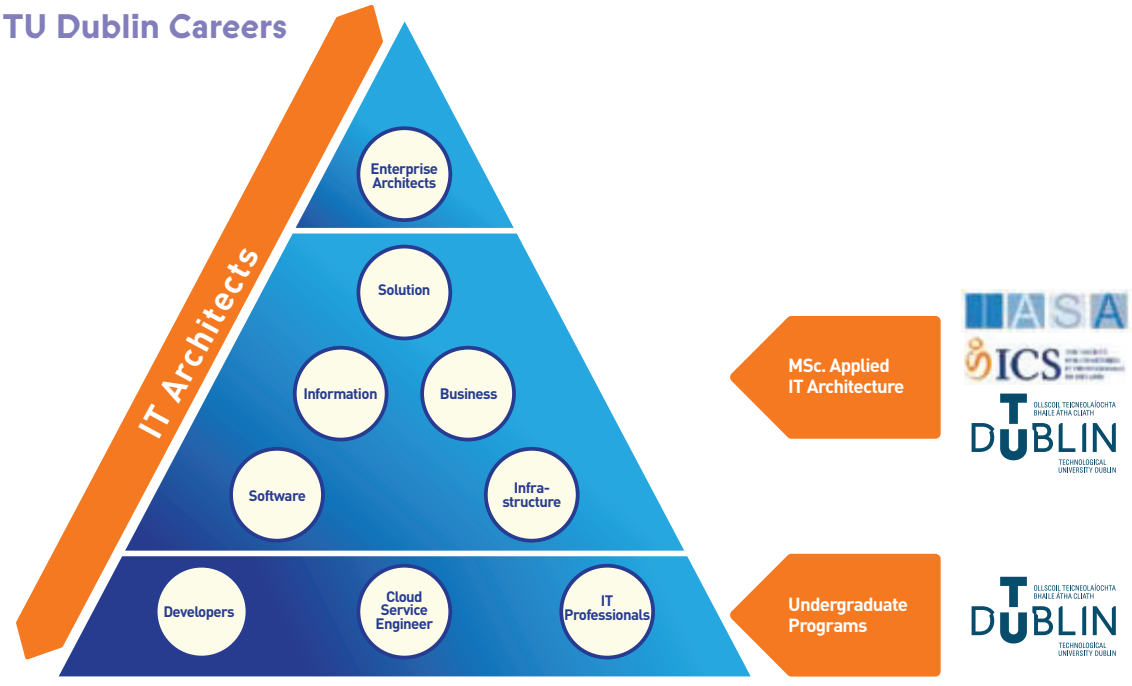
For the first time, candidates can gain a full Masters of Science degree in this specialist area through a mixed learning process with an emphasis on practical application in the workplace.

What is IT Architecture? (From IASA Global)

Architecture at IASA is the practice of business, organization or client gain through the application of technology strategy. It is the art and science of designing and delivering valuable technology strategy. At its core, the ITABoK describes how to create a professional person or group of professionals who can consistently find new applications of technology to generate positive outcomes for their client or employer. IT Architects:

- Retain depth in technical skill as well as business skill
- Able to successfully work with both business and technical staff
- Develop their own or others business cases based on technology driven innovation
- Retain the ability to deliver projects on those business cases
- Deliver business projects more successfully based on outcomes than others

TU Dublin Careers



PROJECTS

PROJECTS contd.

Comparison of Asynchronous Architectural Patterns on AWS & GCP Using Terraform

Alan McGee
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
x00205745@mytudublin.ie

Introduction

Asynchronous architecture patterns have become a fundamental part of the way modern software systems communicate. Distributed systems require a fault tolerant and loosely coupled way to transmit messages. The thesis investigates the managed asynchronous message queues offered by AWS, "Simple Queue Service" (SQS), and Google, "Pub/Sub", and focuses on the implementation of the four patterns: Fan In, Fan Out, 1:1 and M:N. Terraform is an infrastructure as code (IaC) tool which is used to create deployment scripts and is used in the research to assess the resource requirements and configuration, complexity, and analyse vendor specific features in implementing the four architecture patterns.

Message Queues

A message queue is a data structure that stores messages until they are consumed. Message queues can be used when different systems, or different parts of the same system, need to be able to communicate with each other. The producer (sender) creates a message (data) that must be passed to the consumer (receiver). The message can be text or an organised array of data such as JSON or XML. The consumer receives and processes the messages from the queue. The queue system is managed by a broker, which is responsible for ensuring that the messages are delivered from the producers to the consumers and managing processes for message ordering and delivery failures.

Architecture Patterns

1. **1:1 Pattern:** This is the simplest type of queue with one message producer and one consumer.
2. **M:1 Pattern:** The M:1 pattern, or "fan in" message pattern, consists of many message producers and one consumer.
3. **1:M Pattern:** The 1:M pattern, which is also known as a "fan out" pattern, requires broadcasting messages from one producer to many message consumers.
4. **M:N pattern:** The M:N pattern, which is also known as a fan in and fan out pattern, requires broadcasting messages from many message producers to multiple message consumers.

Method

Test environments were created in AWS and GCP consisting of a micro VM running Ubuntu. A bash script for each asynchronous pattern was created and used to automate the configuration of the VM, then download and install Terraform with the appropriate main.tf file for each pattern.
Bash scripts were then created to simulate the process of system components sending and receiving messages to and from the queues according to the required architectural message pattern.

Infrastructure As Code

Infrastructure as Code (IaC) is where the configuration of computer infrastructure such as servers, networks and storage, are defined, automated and managed to create environments. Previously this would have been done manually, or in combination with scripts, by a system administrator. Terraform is an IaC tool provided by HashiCorp. It allows users to provision and manage infrastructure across many cloud environments.

Results

Both cloud providers could provide messages queues corresponding to the message patterns. The "fan in" patterns were similar on both platforms. However, while GCP Pub/Sub could manage the "fan out" patterns natively, AWS SQS required an additional service SNS, which increased the complexity and the total number of Terraform resources.

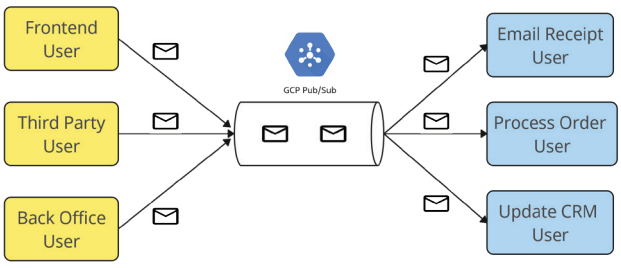
	AWS SQS	SQS + SNS	GCP Pub/Sub
1:1	Yes	-	Yes
M:1	Yes	-	Yes
1:M	No	Yes	Yes
M:N	No	Yes	Yes

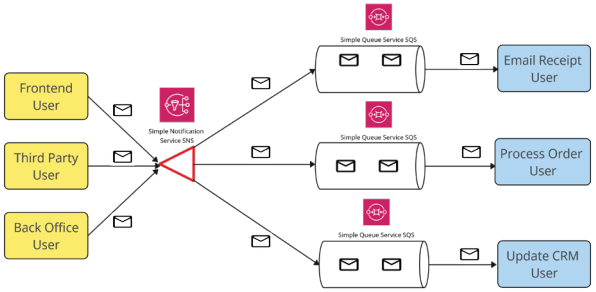
	AWS	GCP
1 : 1	8	7
M : 1	8	9
1 : M	25	11
M : N	25	13

Left table shows the patterns matrix, right table shows the number of Terraform resources required.

Discussion

Pattern Analysis: Both platforms provide managed services that fulfil the basic message queue requirements for all four architectural message patterns.
Terraform Analysis: For the fan out patterns, AWS required more than double the amount of Terraform resources than GCP
Complexity Analysis: The main differences in complexity are user policies/permissions and the configuration of fan out queues and subscriptions.
Features Analysis: SQS has a rich set of features, while Pub/Sub has improved with features such as message ordering and exactly once delivery.





The M:N pattern for GCP is shown on the left with a comparatively simpler configuration. On the right the M:N pattern in AWS is more complex requiring SNS to broadcast to three SQS queues, one for each consumer.

Conclusions and Future Work

The core strengths of AWS SQS are on point-to-point messaging with FIFO and deduplication. While fan out patterns can be supported, they require the addition of the AWS SNS service, which requires additional configuration.
GCP Pub/Sub offers easier and simpler configuration, and fan out patterns and subscriptions are supported natively. Originally, some of the queue features were not as robust as those offered by SQS, but over time new features others have now matched, and in some cases overtaken SQS. The use case, and the specific features that are required from the queues, determines which system is preferable over the other.

QR Code for Recording

The Irish Food Manufacturing Industry's Preparedness for NIS2 Cybersecurity

Niall McGowan
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205713@myTUDublin.ie

Introduction

The EU Network and Information Security's updated cybersecurity directive, NIS2, introduces compliance requirements demanding that organisations assess their cyber posture, and risks associated with their systems, and apply an appropriate level of cybersecurity measures to mitigate against any credible risks. NIS2 will be an enforced legal requirement for critical infrastructure (CI), including food manufacturing, with heavy penalties if not complied with. Cybersecurity risk preventative measures, supply chain risk management, incident response planning and reporting, are mandated to be in place for CI organisations across the EU. A survey of Irish food manufacturers shows that they are not prepared for NIS2. Using a case study it was determined that cybersecurity frameworks can aid the journey to NIS2 compliance and increase business resilience.

NIS2

Article 20: Management have ultimate governance for cyber risks and can be held liable for cybersecurity infringements or negligence.
Article 21: Appropriate and proportionate cybersecurity must be in place. Cyber policies, incident handling, supply chain risk evaluation and basic cyber hygiene are some of the listed requirements.
Article 23: Reporting obligations. Any significant incident that has an impact on the provision of services or causes financial loss must be reported.
Article 33: Important entities, such as Food Manufacturers, will be supervised in an "ex post" or reactive manner.

QR Code for Recording

Case Study

Center for Internet Security (CIS): CIS Controls self-assessment tool (CSAT) can help create a cybersecurity best practice gap analysis and determine a road map to strengthen cybersecurity across multiple security controls, each working towards system resilience and NIS2 compliance. It also aligns with the US National Institute of Standards and Technology (NIST) cyber principles.

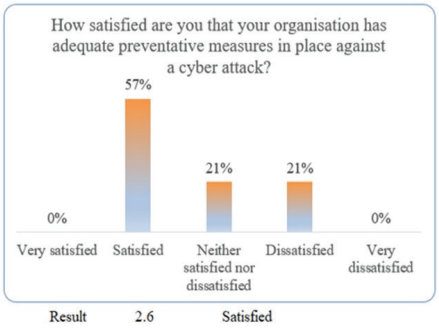
IT and Operational Technology (OT) Gap Analysis: Rating IT and OT separately enables clarity and gives focus for business investment in the areas with most risk. The needs of OT, where the digital meets the physical world, are different to those of IT, as the focus is on safety and operational availability rather than data.

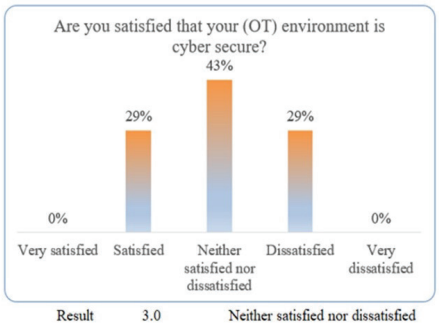
Centre for Internet Securty GAP Analysis

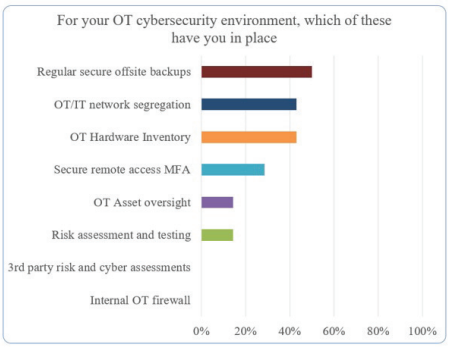
Control #	CIS Control	IT	OT
1	Inventory and Control of Enterprise Assets	20	0
2	Inventory and Control of Software Assets	25	0
3	Data Protection	35	18
4	Secure Configuration of Enterprise Assets	71	23
5	Account Management	67	12
6	Access Control Management	47	25
7	Continuous Vulnerability Management	32	7
8	Audit Log Management	6	4
9	Email and Web Browser protections	68	80
10	Malware Defences	68	0
11	Data Recovery	90	35
12	Network Infrastructure Management	71	50
13	Network Monitoring Defence	45	23
14	Security Awareness and Skills Training	50	0
15	Service Provider Management	14	4
16	Application Software Security	50	16
17	Incident Response Management	12	11
18	Penetration Testing	0	0
		43	17

81 - 100
61 - 80
41 - 60
21 - 40
0 - 20

Food Manufacturing Survey Results







57% of respondents are satisfied that they have preventative measures against cyber-attacks.
29% of respondents are satisfied that their OT environment is cyber secure.

OT asset oversight 14%. OT backups 50%.
0% assessment of OT 3rd party risk.

Conclusions

The Irish food manufacturing industry has many short fallings in both awareness and preparedness for NIS2 and cybersecurity risks in general. IT and Operation Technology (OT) convergence and cooperation is paramount to creating a cybersecure environment for manufacturing organisations. The OT domain has fallen behind and needs to work in collaboration with IT to build a cohesive business continuity plan and enable resilience for all systems within the manufacturing environment's "system of systems". To be compliant with NIS2, you must know what to protect. OT Asset Inventory was an issue for all those surveyed and 3rd party service provider compliance assessment was not in place for any survey respondent. The case study demonstrated how you can greatly increase your cybersecurity awareness, and preparedness, by using a cybersecurity framework and a tool such as CIS CSAT. By identifying your systems and digital assets, assessing the risks for each system, and working as a collaborative team of IT, OT and 3rd party stakeholders, you can move the needle forward toward better business continuity through cyber resilience, and increase NIS2 compliance.

Future Work

The operational and engineering part of cyber awareness have been neglected until now. NIS2 enforces industries to seize the opportunity to use every tool to improve resistance to cyber-threats. Engineering tools are a part of that opportunity.
Cyber Informed Engineering is the umbrella for a system of systems - safety engineering, automation engineering, network engineering, OT, IT and cybersecurity. If there is a viable threat, then assess and mitigate against that threat with defence-in-depth.

Analysis of impact of configuration choice upon Azure Service Bus performance.

Brian Skehan
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
x00205786@mytudublin.ie

Introduction

This thesis aimed to understand how different choices in configuring a message broker or a queue within it affect the message brokers performance. For example, enabling duplicate detection may be beneficial for reliability or consistency but may have an adverse impact upon the performance of the system.

With the evolution of Event Driven Architecture and cloud infrastructure, system architects build layers of isolation and loose coupling between components. These abstractions increase the difficulty in accurately determining the performance of a software application. Architects require an understanding of the functionality options available, how they operate and their impact upon the software system being delivered. This enables them to design high performing, scalable, and reliable applications. Choosing the correct configuration is dependent on the specific needs of the system. Features that are enabled or disabled could improve performance in one context yet be detrimental in another.

Research Question 1

How do selected configuration options impact performance of Azure Service Bus?

This question explores how various configuration settings, duplicate detection, sessions, partitions influence the overall performance of Azure Service Bus. Understanding these factors provides insights into optimizing the system for high throughput.

Research Question 2

How does event or message payload size impact upon performance of Azure Service Bus?

This question examines how the size of events or messages affects processing speed and throughput. Larger payloads may introduce performance bottlenecks, while smaller payloads could optimize efficiency. This analysis will help determine the trade-offs between payload size and system performance.

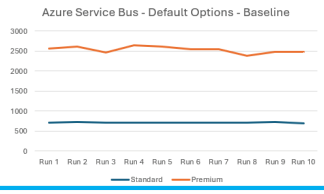
Research Question 3

Does the number of subscriptions to an Event topic impact upon performance of Azure Service Bus?

This question investigates how the number of subscriptions to a single event topic affects the system's ability to process messages. It evaluates the potential performance degradation as subscription counts increase, offering guidance on scalability and architectural design for systems requiring high fan-out capabilities.

Research Method

The null hypothesis for each question is that configurations options set in Azure Service Bus or message size have no impact upon the performance of Azure Service Bus. To test the hypothesis a high number of messages were enqueued at a high rate onto Azure Service Bus and the throughput rate was measured. A baseline throughput rate was determined and used as the standard against which the throughput rate associated with the various configuration options were analysed. If a statistically significant difference can be determined for the throughput rate, then the null hypothesis can be rejected.



Results Research Question 1

These null hypotheses were rejected:

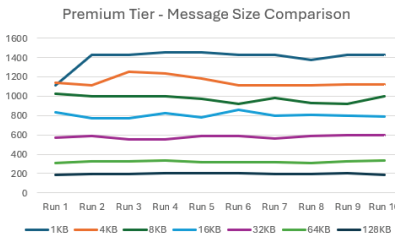
- The choice of tier for Azure Service Bus does not impact message throughput.
- Duplicate detection on a queue within an Azure Service Bus instance does not impact message throughput.
- Enabling Sessions on a queue within an Azure Service Bus instance does not impact message throughput.

The null hypothsis for "Enabling partitioning on a queue within an Azure Service Bus instance does not impact message throughput." was not rejected as a statistically significant result was not determined.

Results Research Question 2

Null Hypothesis – throughput rate performance is not impacted by the size of the message being enqueued.

- This null hypothesis was rejected as a statistically significant result was determined.



Results Research Question 3

Does the number of subscriptions to an Event topic impact upon performance of Azure Service Bus?

It was observed that increasing the number of subscribers on the premium tier has a statistically significant negative impact upon the message throughput rate. Increasing the number of subscriptions from 1 to 5 decreasing performance by an approximate 20%. 50 subscriptions drops performance by almost 80% as compared to the baseline throughput rate.

Premium Tier	Mean	t Statistic	Critical value	P (T <= t) two-tail	Significance Level
Default	2535.56				
1 Subscription	2004.63	13.985	2.306	< 0.001	0.05
5 Subscriptions	405.46	59.208	2.306	<0.001	0.05
50 Subscriptions	557.137	44.522	2.306	<0.001	0.05

Conclusions and Future Work

Final conclusions The thesis provides empirical evidence of how specific configuration options in Azure Service Bus impact performance. By quantifying these effects, the study addresses a gap in the literature, offering actionable insights for system architects.

Future Work Future research could address these gaps by examining other message brokers, alternative cloud providers, combining configuration options, exploring cross-region deployments, SaaS and non-SaaS Message Brokers and integrating advanced monitoring tools to capture real-time system behaviour under dynamic conditions.

QR Code for Recording

AI-Based Predictive Analytics for Network Operations

Timur Nikisin
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205784@mytudublin.ie

Introduction

The application of Artificial Intelligence (AI) tools and methods to real-world problems has been advancing at an unprecedented pace. From the integration of generative AI tools into everyday activities to the application of Machine Learning techniques for solving complex challenges in Information Technology (IT) systems, AI continues to attract significant attention from individuals, businesses, and academia alike. This research paper focuses on the application of AI in IT operations, specifically exploring how AI in general, and AI-based predictive analytics in particular, can enhance network operations processes. The study investigates whether AI represents a transformative technology capable of making IT networks more reliable and fault-tolerant, or if it is merely the latest iteration of the ever-evolving IT hype cycle. By examining the potential of AI to predict and proactively address network failures, this research aims to shed light on the practical value of AI-driven solutions in improving network reliability and operational efficiency.

Research Methodology

This study investigates the application of AI in network operations through a combined approach of primary research and literature review. The primary research involved creating a virtualized network lab using open-source tools and libraries to simulate real-world scenarios, such as high congestion, latency, and resource unavailability. Data collected from these simulations was analyzed by an AI model designed to predict behavioral anomalies and demonstrate the benefits of AI-driven analytics in enhancing network performance and reliability.

To complement these findings, an extensive literature review explored the application of AI in proprietary systems offered by leading vendors. This review provided valuable insights into commercial networking solutions and expanded the research scope, enabling a comparison between proprietary and open-source approaches to AI in network operations.

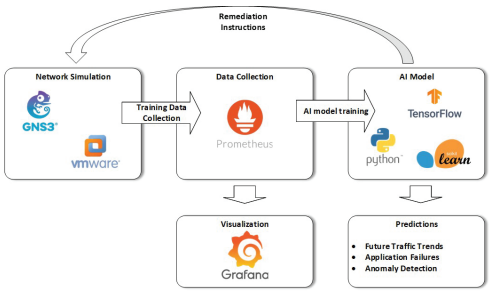
By integrating experimental results with insights from proprietary systems, the study aims to guide IT professionals and organizations in adopting AI tools effectively. The findings emphasize aligning technological advancements with operational goals, driving smarter, more efficient, and reliable network systems for future innovations.

Experiment

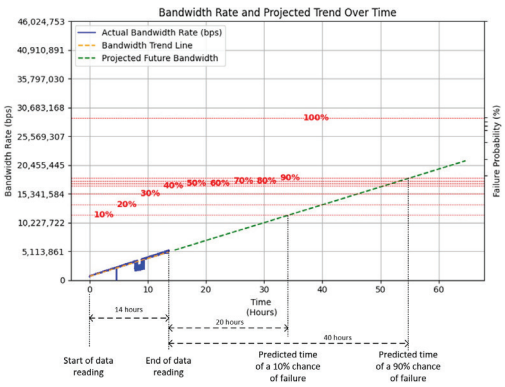
The experiment focuses on using a neural network to predict web application failures based on simulated network conditions. A network testbed was created to collect data under various scenarios, including network congestion and anomalies. The neural network was trained on this data to identify patterns and predict potential failures. The model's predictions were further utilized to implement automated remediation strategies, demonstrating the potential of AI to enhance network reliability and performance.

Tools and Tecnologies

The experiment utilized GNS3 to simulate network environments, enabling the creation of realistic scenarios like congestion and anomalies for data collection. Prometheus was used to gather key metrics such as bandwidth utilization and application health, while Grafana provided interactive dashboards for visualizing trends and identifying anomalies. TensorFlow powered the neural network, which analyzed patterns in the collected data to predict web application failures. These predictions informed automated remediation strategies, such as traffic rerouting, to enhance network reliability and minimize downtime. The diagram represents the tools and technologies that were used as part of the experiment, along with their connections.



Summary of the Experiment and Results



The experiment demonstrated the use of a neural network to predict web application failures caused by network bandwidth issues. A balanced dataset of 99,327 samples, including both failure and non-failure instances, was used for training. The model achieved an F1-score of 0.8983 for failure predictions.

The F1-score, a critical metric combining precision (how many predicted failures were correct) and recall (how many actual failures were identified), reflects the model's strong reliability in predicting failures while minimizing false positives and false negatives.

Analysis of the results showed that failure likelihood began at 10% when bandwidth exceeded 11 Mbps, increasing to 90% at 18 Mbps. These predictions were used to implement proactive remediation strategies, such as rerouting traffic to alleviate congestion, effectively preventing service disruptions.

The findings validated the neural network's ability to analyze real-time and historical data, providing actionable insights into potential network issues. This highlights the potential of AI-driven predictive analytics to enhance network reliability, optimize performance, and minimize downtime through early detection and automated remediation. The results showcase a promising approach for integrating machine learning into network operations to address real-world challenges efficiently.

Conclusions and Future Work

This research demonstrated the potential of AI-driven predictive analytics to enhance network operations by accurately predicting web application failures using neural networks. The findings highlight the advantages of machine learning in providing proactive and efficient network management compared to traditional methods.

Future work should focus on incorporating additional features, exploring more complex scenarios, and advancing automation in network management. Continued research will drive innovation toward smarter, more reliable, and autonomous network systems.

QR Code for Recording

Comparative Cost and Capacity Analysis of Managed Service and Non-Managed Service API Gateway Architectures on Leading Cloud Platforms

George Brown
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205802@myTUDublin.ie

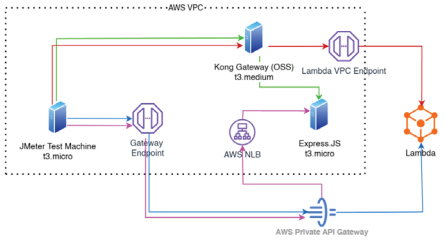
Introduction

Three of the benefits often claimed for choosing API Gateway managed services are: scalability, ease of use and potential cost savings. Implying managed service API Gateways are more capable and cost effective for handling very large workloads. Many architects may make the assumption that non-managed service API Gateways will not suffice for handling a large quantity of requests or high throughput, or that for handling such, architectures using such API Gateways would need to be costly and complex, requiring expensive compute resources and complicated scaling solutions.

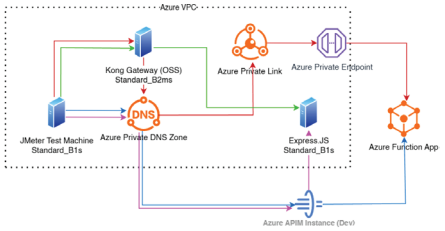
Objective

This research is intended to explore the validity of assumptions regarding the capabilities and cost effectiveness of non-managed service API Gateways when compared to managed service API Gateways, through experimentation. Conclusions are intended to inform decisions regarding API Gateways in IT/Software Architectures.

Method



Experiment involved two analogous systems, one on AWS, the other on Azure. Both had x3 VMs hosting JMeter, Kong Gateway OSS, and Express.JS server. Both systems also used the Cloud Service Providers respective managed service API Gateways, AWS API Gateway and Azure APIM as well as their Serverless Compute offering, AWS Lambda and Azure FunctionApp. Several variations of API Gateway architectures were evaluated to understand request behavior and costs at different tiers of usage.



Results

Throughput and Monthly Visitor Capacity

API Gateway	Endpoint	Cloud Host	RPS	MVC
Kong Gateway OSS	Express.JS	AWS	1,268	1,500,000
Kong Gateway OSS	Express.JS	Azure	1,048	1,200,000
Azure APIM Gateway	Express.JS	Azure	895	1,000,000
AWS API Gateway	Express.JS	AWS	443	537,000
Azure APIM Gateway	FunctionApp	Azure	404	491,000
Kong Gateway OSS	FunctionApp	Azure	221	269,000
Kong Gateway OSS	Lambda	AWS	100	121,000
AWS API Gateway	Lambda	AWS	91	110,000

Kong Gateway OSS with Express.JS hosted on AWS handled the highest throughput and therefore monthly site visitor capacity. Azure APIM handled almost double the throughput of AWS API Gateway. Endpoint corresponds strongly with costs, but API Gateway choice has substantial impact.

API Gateway Costs at Different Usage Tiers

Commerce Site Visitors Per Month	Highest Cost API Gateway Architecture ($)	Lowest Cost API Gateway Architecture ($)
1K	147.20 - **Azure APIM** + FunctionApp	12.76 - **AWS API Gateway** + Express.JS
15K	147.20 - **Azure APIM** + FunctionApp	22.19 - **Kong Gateway** + Express.JS (AWS)
50K	147.20 - **Azure APIM** + FunctionApp	42.72 - **Kong Gateway** + Lambda (AWS)
250K	180.44 - **AWS API Gateway** + Express.JS	43.32 - **Kong Gateway** + Express.JS (AWS)
1M	147.17 - **Azure APIM** + Express.JS	44.19 - **Kong Gateway** + Express.JS (AWS)
1.5M	N/A	44.77 - **Kong Gateway** + Express.JS (AWS)

Cheapest choice can change dramatically at different tiers of usage. AWS API Gateway costs the least at lower tiers, when paired with a server based endpoint. However it can become the most expensive at higher tiers of usage. At higher tiers of usage, some API Gateway Architectures did not reach the throughput needed, and cannot be considered here.

Cost Effectiveness (Cost per Throughput

Commerce Site Visitors Per Month	Least Cost Effective API Gateway Architectures ($ per RPS)	Most Cost Effective API Gateway Architectures ($ per RPS)
1K	0.41 - **Kong Gateway** + Lambda (AWS)	0.02 - **AWS API Gateway** + Express.JS
15K	0.42 - **Kong Gateway** + Lambda (AWS)	0.03 - **Kong Gateway** + Express.JS (AWS)
50K	0.43 - **Kong Gateway** + Lambda (AWS)	0.03 - **Kong Gateway** + Express.JS (AWS)
250K	0.41 - **AWS API Gateway** + Express.JS	0.03 - **Kong Gateway** + Express.JS (AWS)
1M	0.16- **Azure APIM** + Express.JS	0.03 - **Kong Gateway** + Express.JS (AWS)
1.5M	N/A	0.04 - **Kong Gateway** + Express.JS (AWS)

AWS API Gateway when paired with a server based endpoint is the most cost effective API Gateway Architecture at lower tiers of usage - giving the most throughput per dollar spent. Kong Gateway when paired with a server based endpoint is the most cost effective at all usage tiers beyond 1K visitors a month. At higher tiers of usage, some API Gateway Architectures did not reach the throughput needed, and cannot be considered here.

Conclusions

Server based, non-managed API Gateways when paired with server-based compute endpoints, without complex optimizations such as gateway instance scaling, can handle larger throughputs and corresponding monthly visitor capacities when compared to serverless managed API Gateways. There are potential cost savings in choosing a managed serverless gateway at smaller tiers of usage, but the opposite is true at larger tiers. Server-based non-managed API Gateways may achieve larger scales of usage, at a lower cost, and with little need to implement scaling gateway instances. Median latencies and response times are affected by API Gateway architectures. This difference could be impactful for certain solutions that are intended to be highly optimized for low latency. The choice of API Gateway correlates with the maximum throughput achievable, but the choice of endpoint has a stronger correlation. AWS API Gateway, Azure APIM Gateway, and Kong Gateway OSS, all allow for an SLA of 99.95 percent or more requests to have responses times and latencies below 500ms. Server-based API Gateways can offer a higher throughput for each dollar cost than serverless managed API Gateways at a variety of tiers of usage. An exception to this is AWS API Gateway when paired with Express.JS at the lower tier of usage.

Future Work

Further research could be done with a greater variety of configurations, both in terms of software and hardware. This could mean introducing always provisioned managed API Gateway instances to avoid cold starts, evaluating instance scaling on self-managed server-based API Gateways, configuring different retry rates using software settings for Lambda, using a variety of virtual machines with different capabilities for hosting both the API Gateway and the endpoint. Increasing the scope of gateway varieties evaluated. The impact of regional and interregional communications on API Gateway throughput capacity, costs and monthly visitor capacity could also be researched.

Cost Optimization in Open Telemetry

Niksa Jadric
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205794@myTUDublin.ie

Introduction

OpenTelemetry (OTEL), an open-source observability framework and a Cloud Native Computing Foundation project, has emerged as an industry standard for collecting and exporting telemetry data (traces, metrics and logs). Its features are suitable for monitoring complex cloud-native systems as it offers standardized instrumentation and sampling options to help organizations track system health. However, there is a trade-off between these benefits and resource consumption.
This thesis examines cost optimization strategies in OpenTelemetry implementations by addressing two research questions: (1) identifying the key factors influencing cost management in OTEL while ensuring effective observability and (2) exploring configuration strategies within the OpenTelemetry Demo application to optimize telemetry collection costs.

OTEL components

OpenTelemetry in its essence is three base components: protocols (specs and standards), instrumentation (SDK ecosystem) and OpenTelemetry collector, which we can look at as an agent.
Collector is a central component that is used to make pipelines, to define the path the signals flow, and its configuration is one of the main points of interest in this research.

The Collector

The OpenTelemetry Collector is a flexible and powerful component, designed to be the central part of the ecosystem. It is responsible for gathering, processing, and routing telemetry data (metrics, traces, and logs) to various destinations. The Collector is using three core elements: receivers, processors, and exporters.

Cost optimization

OpenTelemetry (OTEL) is a powerful tool for observability, but it introduces significant costs related to storage, computation, and engineering. Managing these costs is crucial and requires collaboration across DevOps teams.
Leveraging configurational flexibility and automation tools such as OpAMP, organizations can minimize OpenTelemetry costs through the strategic optimizations such as Efficient Sampling, Dynamic Adjustments, Custom Distributions, Integration Choices and Serverless Deployments.

Researching sampling configurations

In the second part of this research, experiments are conducted against OTEL Demo Application. Prometheus is used as a backend for collected metric, and Grafana for displaying and exporting data, while the researcher was configuring OTEL Collector for various configurations such as head based sampling (probabilistic), tail based sampling, attribute inserts and more. In addition, a tests were performed where several error simulations ran while applying different sampling techniques.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
9207f92b62f7	load-generator	670.84%	5.477GiB / 6GiB	91.28%	3.85GB / 208MB	17.1MB / 86.5GB	3023
5d81722daa35	otel-col	16.73%	170.3MiB / 500MiB	34.07%	364MB / 83.1MB	0B / 0B	16
d11992da1783	jaeger	2.09%	207.5MiB / 600MiB	47.92%	84.8MB / 5.27MB	0B / 0B	14
2eca2e602b27	frontend-proxy	16.92%	66.63MiB / 100MiB	66.63%	4.92TB / 6.88TB	8.19kB / 8.19kB	26
1e3abb9b58a2	frontend	43.97%	115.8MiB / 300MiB	38.61%	89.3GB / 795GB	5.92MB / 4.1kB	22
fe8f3c6dcc6c	checkout-service	0.03%	20.68MiB / 20MiB	68.27%	10.5MB / 91.5MB	8.19kB / 0B	11
221895df399e	recommendation-service	1.72%	77.68MiB / 600MiB	12.91%	26GB / 9.99GB	1.47MB / 819kB	29
f6f7e5bf7221	cart-service	2.04%	88.62MiB / 150MiB	96.60%	3.48GB / 5.10GB	119kB / 0B	56
a1e226abbf8a	payment-service	0.00%	70.7MiB / 150MiB	47.13%	284MB / 193MB	3.86MB / 4.1kB	22
2214813f3a89	quote-service	0.03%	10.83MiB / 30MiB	36.06%	252MB / 605MB	163kB / 0B	6
19957c3fa283	product-catalog-service	1.21%	15.33MiB / 30MiB	51.11%	6.78GB / 56GB	3.61MB / 0B	15
e7eeb4ff7c69	currency-service	1.03%	7.367MiB / 20MiB	11.72%	81.6MB / 190MB	8.19kB / 0B	17
81c2e7e97a2e	imageprovider	1.66%	7.395MiB / 150MiB	4.93%	18.8GB / 3.21TB	8.19kB / 8.19kB	7
87bda988f149	ad-service	0.60%	238.5MiB / 350MiB	65.84%	261MB / 670MB	381kB / 1.6GB	30
11bdfa56f092	shipping-service	0.01%	26.65MiB / 30MiB	88.18%	396MB / 399MB	0B / 0B	10
d971cf66f726	accounting-service	0.09%	96.54MiB / 150MiB	64.36%	3.41GB / 71.04GB	185kB / 189MB	25
ef6e8f0a0aea	email-service	0.01%	84.63MiB / 150MiB	65.09%	5.53MB / 6.66MB	77.8kB / 0B	8
d83c10f16cb5	flagdui	0.00%	48.37MiB / 150MiB	25.46%	116MB / 584kB	2.04MB / 4.1kB	22
6e55d92d2dba	fraudetection-service	0.31%	288.6MiB / 350MiB	82.46%	3.9GB / 6.49GB	1MB / 1.81GB	35
a93e3c851a4a	flagd	1.63%	72.66MiB / 80MiB	90.58%	324MB / 2.48GB	29.2MB / 0B	14
c567e837f91d	grafana	0.12%	101.9MiB / 500MiB	20.36%	137MB / 41.3MB	13.5MB / 131MB	15
8163d323dad1	kafka	1.18%	597.2MiB / 800MiB	74.65%	170MB / 268MB	6.24MB / 2.41GB	101
6e7c2cc958ed	valkey-cart	0.96%	3.515MiB / 20MiB	11.77%	1.7GB / 1.23GB	4.1kB / 811MB	6
0a7acf7769d1	opensearch	1.66%	949.1MiB / 1GiB	92.69%	25.1GB / 884MB	59.6GB / 95.7GB	118
a3ef581f5d0f	prometheus	0.97%	288.5MiB / 600MiB	48.08%	87.9GB / 1.5GB	607MB / 13.9GB	15
21025f388e30	cadvisor	5.66%	94.09MiB / 300MiB	31.36%	1.52GB / 98.6GB	6.59MB / 607GB	21

Final experiment results for OTEL Demo application

Choosing correct sampling for a given scenario can help manage telemetry data volume and can enable systems to drop selected traces without losing critical information.
In the experiments, all gathered results are composed into a table on the right that shows heat map for each column pointing out differences in values for better readability.
The highest CPU utilization was recorded for experiment 8 for no specific sampling, but only attribute inserts. Closely following are tail-based techniques, but little difference between head-based sampling methods. This makes sense as regardless of number of samples coming at receiver component, sampling happens before processor component and avoids CPU usage. The least CPU utilization is seen in default setup with 0 and 10 users respectively, which is expected.

ID	Experiment	CPU utilization average	Memory utilization average (MB)	Network receive average (MB per 15 seconds interval)	Network transmit average (MB per 15 seconds interval)	Network receive total (MB in 10 minutes)	Network transmit total (MB in 10 minutes)	Storage (MB)
1	0_users_M0-2_default	0.0171	144.2871	0.0076	0.0159	0.3115	0.6516	21.8668
2	10_users_M0-2_default	0.0684	143.9346	0.2381	0.0529	9.7634	2.1692	2068.5577
3	50_users_M0-2_default	0.1183	155.2122	0.4871	0.1035	19.9710	4.2431	4395.9460
4	50_users_M3_probabilistic_20_traces_100_logs	0.1126	132.6453	0.4941	0.0680	20.2579	2.7882	896.3104
5	50_users_M4_probabilistic_50_traces_100_logs	0.1179	131.4444	0.4946	0.0851	20.2797	3.4877	2226.0148
6	50_users_M5_probabilistic_100_traces_20_logs	0.1052	136.4534	0.5193	0.0924	21.2923	3.7867	4702.6631
7	50_users_M6_probabilistic_100_traces_50_logs	0.1104	135.8158	0.5243	0.0998	21.4964	4.0936	4751.5639
8	50_users_M7_probabilistic_50_traces_50_logs	0.1104	138.7389	0.4962	0.0755	20.3441	3.0944	2277.0503
9	50_users_M8_50_attribute_inserts	0.1454	143.4123	0.5264	0.1156	21.5804	4.7400	2529.5310
10	50_users_M9_tailsampling_error_code	0.1224	204.5350	0.5286	0.0596	21.6731	2.4436	11.1994
11	50_users_M10_tailsampling_filter_attributes	0.127	226.0021	0.5240	0.0687	21.4826	2.8158	523.1233
12	50_users_M9_tailsampling_error_code feature_flag: adServiceFailure	0.1206	224.1066	0.5141	0.0593	21.0781	2.4323	15.6920
13	50_users_M10_tailsampling_filter_attributes feature_flag: adServiceFailure	0.1267	226.5826	0.5039	0.0674	20.6593	2.7652	509.5162
14	50_users_M9_tailsampling_error_code feature_flag: adServiceFailure, cartServiceFailure, paymentServiceUnreachable	0.1189	229.3475	0.4888	0.0600	20.0424	2.4586	79.6168
15	50_users_M10_tailsampling_filter_attributes feature_flag: adServiceFailure, cartServiceFailure, paymentServiceUnreachable	0.1172	226.8944	0.4630	0.0651	18.9812	2.6691	535.3655
16	50_users_M9_tailsampling_error_code feature_flag: loadgeneratorFloodHomepage, kafkaQueueProblems	0.1251	233.5631	0.5576	0.0842	22.8616	3.4529	11.7861
17	50_users_M10_tailsampling_filter_attributes feature_flag: loadgeneratorFloodHomepage, kafkaQueueProblems	0.1304	243.0650	0.5534	0.0940	22.6875	3.8541	649.7385

Conclusions and Future Work

With the literature research and our own experiments, it can be concluded that cost can be highly managed. Correctly choosing the properties of each component can make a difference between valuable insight at acceptable cost and costly, heavy observability tooling. Another conclusion is that there is no one-fits-all scenario. Every environment needs to have carefully thought-out implementation of OpenTelemetry. However, in this research it has been concluded that there are steady guidelines that could be used as a starting point to ever-adjusting observability system. Some of the proposed future research is automatic and adaptive sampling, and dynamic data aggregation. Additionally, it can be concluded that current and future implementation of AI would be beneficial in cost reduction.

QR Code for Recording

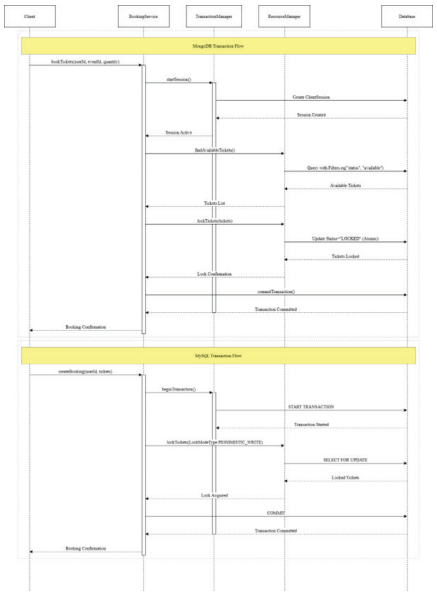
Comparative Analysis of MySQL and MongoDB in a High-Concurrency System

Gabriel Solares
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X002057168@myTUDublin.ie

Introduction

Selecting an appropriate database system is vital for applications requiring high concurrency, as it involves balancing transactional integrity with schema flexibility. This research presents a comparative analysis of MySQL and MongoDB by simulating concurrent ticket booking scenarios and evaluating their performance under user loads ranging from 1 to 5,000 simultaneous requests. A Java-based ticketing system was implemented—utilizing Hibernate for MySQL and Morphia for MongoDB—to assess how each database handles transactional behaviour, schema modifications, and data consistency.
The findings reveal that MySQL offered superior transactional integrity and consistent performance scaling, maintaining stable query response times even under substantial load. In contrast, MongoDB demonstrated greater schema flexibility and better initial performance with moderate parallel loads but exhibited increasing variability at higher concurrency levels. Schema modifications in MySQL required meticulous planning and timing, while MongoDB facilitated seamless schema evolution with minimal impact on performance. These insights enhance the understanding of database selection criteria for high-concurrency systems, indicating that MySQL is preferable for applications prioritizing strict data consistency, whereas MongoDB is advantageous for scenarios necessitating rapid schema evolution.

Transactional Process



Research Questions

1. **How does MySQL ensure transactional integrity in a high-concurrency ticketing system, and what challenges arise from its rigid schema when dealing with complex data relationships?**

   MySQL adheres to ACID principles, ensuring reliable transactions and preventing data inconsistencies during simultaneous bookings. Its rigid schema enforces strict data validation and referential integrity but introduces challenges like table-level locking during schema modifications, potentially causing latency under high-load conditions.

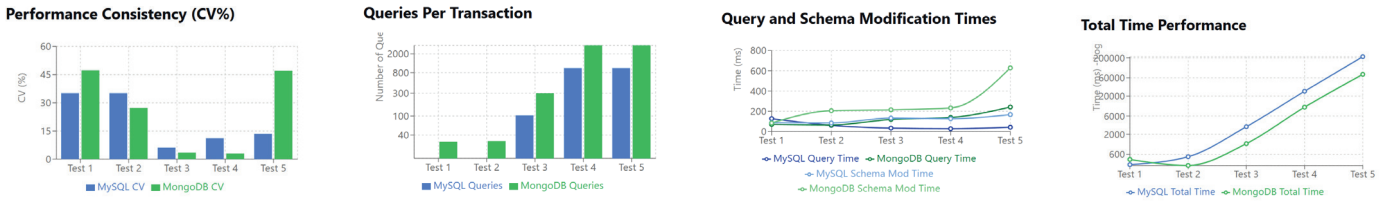
2. **How does MongoDB handle transactions in a concurrent purchase scenario, and how does its flexible schema influence the modelling of intricate data structures?**

   MongoDB leverages a flexible, schema-less architecture, allowing rapid adaptations to evolving data structures without complex joins, and enhancing initial query performance. However, under high concurrency, performance becomes more variable, with schema modifications sometimes leading to inconsistencies and longer modification times, necessitating robust application-level validations.

3. **What are the key differences in implementing transactional operations and data modelling between MySQL and MongoDB in the context of the ticketing system?**

   MySQL utilizes a rigid schema requiring comprehensive upfront planning, ensuring consistent performance and robust data integrity but limiting development flexibility. In contrast, MongoDB offers a schema-less architecture that allows dynamic data structure modifications and scalability but demands sophisticated application-level mechanisms to maintain transactional consistency and data integrity.

Results



Conclusions and Future Work

The analysis demonstrated that MySQL provided superior transactional integrity and maintained stable performance scaling under high concurrency levels. In contrast, MongoDB offered greater schema flexibility and exhibited better initial performance with moderate loads but showed increased performance variability as concurrency intensified. Additionally, while MySQL's schema modifications became more efficient over time, MongoDB faced challenges in maintaining efficient schema changes under high transactional loads.
Future research will focus on exploring optimization strategies for MongoDB to enhance its performance stability under high concurrency. Moreover, the study aims to investigate hybrid database solutions that combine the strengths of both MySQL and MongoDB. Expanding the comparative analysis to include additional NoSQL databases is also planned to provide a more comprehensive evaluation of their performance in high-concurrency environments.

QR Code for Recording

Securing IaC:Comparing Checkov, Terrascan, and Tfsec on AWS and Azure

Maliha Binte Ruhul Amin
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205799@myTUDublin.ie

Introduction

This research focuses on evaluating and comparing the effectiveness of three popular Infrastructure as Code (IaC) security tools—Checkov, Terrascan, and Tfsec—across two widely used cloud platforms, AWS and Azure. With the increasing adoption of cloud technologies and the automation of infrastructure through IaC, ensuring the security of these configurations has become a critical concern for organizations. Each of these tools aims to help identify security misconfigurations in Terraform scripts but their performance and accuracy can vary depending on the cloud environment and specific use cases. By comparing these tools within the AWS and Azure environments, this research aims to provide insights into their strengths and weaknesses, helping organizations make informed decisions on which tool to use for their specific cloud security needs. The findings will contribute to enhancing the security of cloud infrastructures through more effective use of IaC security tools.

Background and Setup

In this research, a literature review was conducted on the state-of-the-art concepts of Infrastructure as Code (IaC), Terraform, HCL, cloud providers AWS and Azure, and the importance of IaC scanning. A detailed discussion focused on IaC security tools, specifically Checkov, Terrascan, and Tfsec. The test scenarios were derived from an open-source GitHub repository, which includes 200 test cases licensed under MIT, with a detailed setup for testing.

Implementation

For this research, twenty test cases were selected, evenly distributed between AWS and Azure, covering five key categories: best practices, encryption, IAM, logging, and networking. Two test cases were chosen for each category, sourced from the GitHub repository . These test cases address common security and compliance issues in cloud infrastructure. To execute the tests, a bash script was executed to pull Docker images for the tools and run the selected test cases across all three tools. The results were then analyzed to evaluate the implications of each misconfiguration.

Result Analysis

1. Catch Rate Analysis:

Based on the scan reports generated from running the bash script, Checkov demonstrated the highest performance, with a 100% catch rate for AWS and 90% for Azure, resulting in an overall catch rate of 95%. Terrascan and Tfsec performed better in AWS, with 90% catch rates, but showed a decline in Azure, with rates of 65% and 70%, respectively. These results highlight Checkov as the most reliable tool, especially for AWS, while Terrascan and Tfsec are more effective in AWS than in Azure. This trend suggests that Checkov's higher catch rates across both platforms make it the preferred choice for comprehensive scans, while Terrascan and Tfsec may need further optimizations to improve their Azure performance.

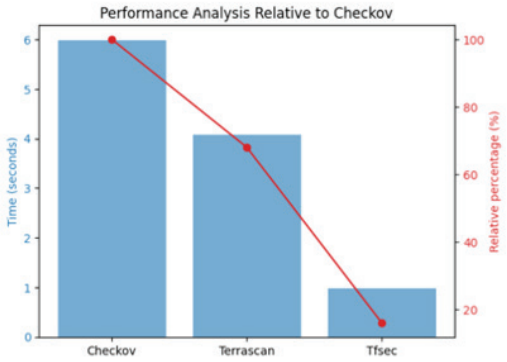
2. Usability Analysis:

This analysis compared three security scanning tools—Checkov, Terrascan, and Tfsec—on readability, UI, ease of use, and features. Checkov offers detailed output with remediation links but can be overwhelming for large codebases. Terrascan provides simpler output but lacks detailed guidance and can be harder to troubleshoot. Tfsec stands out with clear, actionable output and performance metrics, making it easy to use and optimize.

3. Performance Analysis:

The performance analysis compared the execution times of Checkov, Terrascan, and Tfsec. Checkov takes the longest at 5.99 seconds, serving as the baseline (100%) with comprehensive checks for detailed assessments. Terrascan completes scans in 4.08 seconds (68% of Checkov's time), balancing speed and thoroughness. Tfsec is the fastest, completing scans in just 0.98 seconds (16% of Checkov's time), ideal for quick feedback. The choice depends on the need for detailed analysis (Checkov), speed and thoroughness (Terrascan), or rapid scans (Tfsec).

Comparative Analysis



Criteria	Checkov	Terrascan	Tfsec
Support for AWS and Azure IaC	Strong in AWS, good for both AWS and Azure	Effective in AWS, less in Azure	Effective in AWS, less in Azure
Accuracy (Catch rate)	100% AWS, 90% Azure	90% AWS, 65% Azure	90% AWS, 70% Azure
Performance (Execution time)	Longest execution time	Balanced execution time	Fastest execution time
Ease of use	Detailed information	Simpler, less detailed	Focuses on performance metrics
Strengths	Detailed policies, remediation links	Easy-to-understand scan reports	Quick scans, performance insights
Weaknesses	Lacks performance insights, slower	Requires more troubleshooting, less info	Less effective at identifying issues
Best use case	In-depth security analysis	Balanced speed and thoroughness	Quick, detailed assessments when time is critical

Conclusions and Future Work

In conclusion, Checkov, Terrascan, and Tfsec were evaluated for securing IaC on AWS and Azure. Checkov proved most accurate and detailed, ideal for deep analysis but slower. Terrascan offered a balance of speed and accuracy, while Tfsec excelled in fast scans but was less effective in Azure. Each tool has unique strengths, with Checkov best for thorough security checks and Tfsec for quick assessments. Future work could involve testing more tools, diverse scenarios, and integrating CI/CD pipelines, machine learning, and other platforms like GCP.

QR Code for Recording

Performance Evaluation of Zabbix and Azure Monitor in Hybrid IT Infrastructure

Ivan Godoy
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X0025792@myTUDublin.ie

Introduction

As organizations adopt hybrid IT infrastructures that combine on-premises and cloud-based resources, effective monitoring becomes essential to maintain performance and reliability. This study examines and compares two monitoring tools, Zabbix and Azure Monitor, specifically for hybrid environments. The evaluation focuses on key factors such as resource efficiency, alert accuracy, integration capabilities, and cost-effectiveness.
The results guide organizations in selecting monitoring tools that align with their hybrid IT environment goals.

High Stress and Alerts Notification

Azure Machine:

- **CPU:** Zabbix peaked at 99.64%, Azure Monitor at 100%.
- **Memory:** Task Manager 6.3 GiB; Azure Monitor 5.1 GiB; Zabbix 6.09 GiB.
- **Alerts:** Azure Monitor alerted at 191s, Zabbix at 202s.

On-Premises Machine:

- **CPU:** Zabbix 99.64%, Azure Monitor 100%.
- **Memory:** Task Manager 11.6 BiB; Zabbix 11.6 GiB; Azure Monitor 12.4 GiB.
- **Alerts:** Zabbix alerted 102s faster than Azure Monitor.

Cost Analysis

Azure Monitor:

- Data ingestion: €2,853/GB in West Europe (Pay-As-You-Go).
- Notification Costs: €0,67 VM/monthly.

Zabbix:

- Open-source with free templates and notifications.
- Monthly server maintenance: €150 for on-premises hosting.

Performance Analysis - Baseline Monitoring

Azure Machine:

- **CPU Utilization:** Zabbix recorded an average CPU utilization of 13.22% with peaks up to 39.32%, while Azure Monitor captured similar trends with peaks at 44.67%.
- **Memory Utilization:** Zabbix observed an average of 1.62 GiB memory usage, peaking at 2.47 GiB. Azure Monitor reported comparable results, averaging 1.6 GiB and peaking at 2.4 GiB.
- **Network Traffic:** Inbound traffic ranged from 5.66 MB to 15.26 MB, and outbound traffic from 9.7 MB to 37.09 MB using Zabbix. Azure Monitor showed slightly higher peak outbound traffic at 52.3 MB.

Metric	Azure Monitor			Zabbix		
	Min	Avg	Max	Min	Avg	Max
CPU Utilization (%)	6.37	13.4544	44.67	7.285	13.2238	39.32
Memory Utilization(GiB)	0.890	1.6	2.4	0.969	1.62	2.47
Network Traffic In (MB)	5.3	8	15.3	5.66	8	15.26
Network Traffic Out (MB)	9.4	19.1	52.3	9.7	19.05	37.09

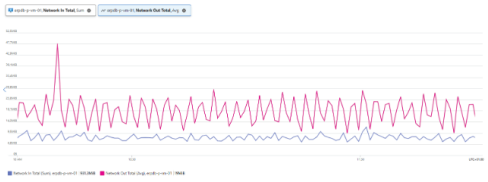
On-Premises Machine:

- **CPU Utilization:** Azure Monitor recorded average utilization at 10.95%, peaking at 74.48%. Zabbix showed higher averages of 14.04% but slightly lower peaks at 63.63%.
- **Memory Utilization:** Both tools captured consistent trends, with Zabbix reporting averages around 10.01 GiB and Azure Monitor 10 GiB. Peaks aligned at approximately 10.6 GiB.
- **Network Traffic:** Zabbix effectively tracked inbound traffic between 1.2 MB and 2.31 MB, and outbound traffic from 0.74 MB to 1.69 MB. Azure Monitor lacked support for network traffic metrics on Arc-enabled machines.

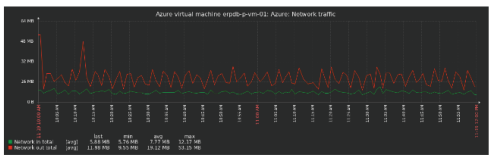
Metric	Azure Monitor			Zabbix		
	Min	Avg	Max	Min	Avg	Max
CPU Utilization (%)	0.84	10.95	74.48	4.02	14.0416	63.625
Available Memory (GiB)	9.7	10	10.6	9.68	10.01	10.36
Network Traffic In (MB)	n/a	n/a	n/a	1.2	1.69	2.31
Network Traffic Out (MB)	n/a	n/a	n/a	0.7362	1.0057	1.69

Topic Overview

(a) Azure Monitoring - Network Traffic

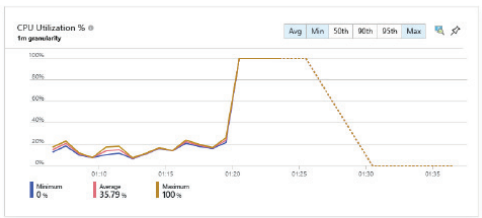


(b) Zabbix Monitoring - Network Traffic



Baseline Monitoring: Azure VM

(a) Azure Monitor - CPU Utilization



(b) Zabbix Monitor - CPU Utilization

High Stress: Azure VM

Conclusions and Future Work

Conclusions: Zabbix offers flexibility and cost efficiency, making it ideal for diverse hybrid infrastructures, while Azure Monitor excels in seamless Azure integration and ease of use for cloud-centric environments. The choice depends on organizational needs, budget, and infrastructure priorities.

Future Work: Explore monitoring specialized workloads (e.g., Lab Equipment), and expand research to include other Cloud services provider such us, AWS, GCP, and long-term performance studies across industries.

QR Code for Recording

An evaluation of Zero Trust Principles in modern software development

Cezar Vararu
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205798@myTUDublin.ie

Introduction

Zero Trust has become a strategic aproach to continuously strengthen the organization's security posture by implementing relevant security practices in modern architectures. This research explores the implementation of Zero Trust Architecture principles from a modern software development perspective. It highlights the concept of Zero Trust by design, meaning that the principles of Zero Trust Architecture extend beyond network security to include the entire software development lifecycle, emphasising the need of integrating these principles from the beginning of the software design and development process.

The research identifies key principles of Zero Trust like *never trust, allways verify*, *continous monitoring* and *continuous evaluation*, and demonstrates their application through the development of security mechanisms in a sample web app. In particular, it implements functionalities such as OAuth 2.0 authorization code flow with PKCE, exchanges the access token via *on-behalf-of* flow when an API needs to call a downstream API, it demonstrates how to onboard users via user federation with an external Identity Provider, as well as implementing an example of Continuous Access Evaluation (CAE) in the frontend application. The demonstration concludes with a cost/performance evaluation assessing the impact of implementing the security mechanisms on the sample application API endpoints based on Zero Trust principles. It examines the additional resource requirements by analysing metrics such as CPU utilization, network throughput, and request duration, to measure the effects of these security enhancements.

Research Questions

1. What cloud services and technologies are available on Microsoft Azure to implement zero trust principles for a web application?
2. What are the cost/performance implications for adopting zero trust principles in software development?

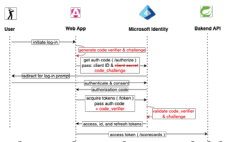
Methodology

To answer the RQ1, the sample application demonstrates the following key implementations: User authentication for SPA, handling the token generation for various backend APIs, maintaining the signed-in user's context during cascading API calls, SSO with user federation based on an external IdP, continuous Access Evaluation (CAE) in the frontend application.

To address RQ2, a series of load tests have been conducted based two representative endpoints: scenario 1 - unprotected endpoint; scenario 2 - endpoint protected for ZT. The performance metrics such as request duration, network throughput, and CPU usage were compared across these test scenarios.

Sample App Implementations

1. Authorisation code flow with PKCE

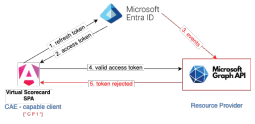


The authorization flow with PKCE is similar to the standard authorization code flow, but it replaces the client secret with a one-time code challenge. This eliminates the need for the client app to store the client secret, making it more secure for public clients like SPAs. The figure on the right describes the authorization code flow paired with PKCE elements marked in red. Mandatory in the upcoming OAuth 2.1 but also available as an extention on OAuth 2.0.

2. Exchange access token on API with OBO flow

The OAuth 2.0 On-Behalf-Of flow is intended for the scenario where there's a need to pass the user's identity through the request chain to a downstream API. When exchanging the access token with OBO flow, the new token will still keep a reference to the user logged-in in the frontend app; however, the token will have a new audience claim, which is the downstream API app registered in Entra ID.

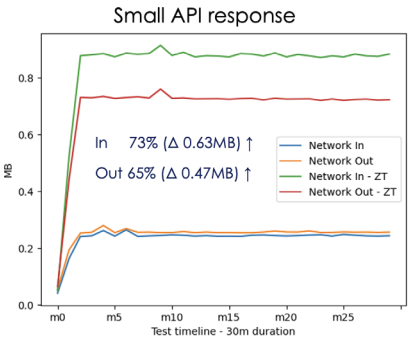
3. Continuous Access Evaluation

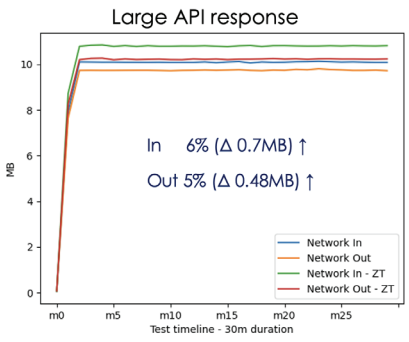


The access tokens have a static lifetime; once issued it could be used until it reaches the expiration date stated in the *exp* claim. What CAE is introducing is a mechanism that will allow an event to be sent to the Resource Provider, flagging that something has changed in the user condition and the access needs to be re-evaluated.

Performance assessments

scenario	response size	metric	avg	increase
1	small	http_req_duration	23ms	89%
2	small	http_req_duration	215ms	
1	large	http_req_duration	50ms	78%
2	large	http_req_duration	232ms	





Conclusions and Future Work

This research paper explored the Zero Trust Architecture, focusing on how its fundamental principles and key concepts could be applied in software development. It provided insights from an application development perspective, and some considerations that software developers must be aware of when building web applications and services to align with Zero Trust principles.

Like most research projects, the efforts presented here can be extended as future initiatives. One opportunity is to enhance the Continuous Access Evaluation, initiating the claim challenge process when there's a change in the user context. Another workstream could involve exploring how zero trust principles can be applied to areas like deployment pipelines, source code repositories, and more broadly in the supply chain to prevent tampered software or libraries, compromising a service or application.

The Business-Day Cloud: A Hybrid Kubernetes and Serverless solution for Sustainable Scaling with Predictable Load Patterns

Brendan Burnside
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205791@myTUDublin.ie

Introduction

Cloud computing has transformed industries with benefits like scalability, flexibility, and high availability. However, its rapid growth has led to significant carbon emissions, as data centers consume increasing amounts of energy. The cloud industry's carbon footprint now exceeds that of aviation, highlighting the need for sustainable practices by both providers and consumers.

Research shows that architectural patterns greatly impact cloud energy efficiency. Solutions like containerization with Kubernetes improve resource usage, while serverless functions offer scalable, event-driven execution. Despite their efficiency, Kubernetes requires minimum resources to remain available, and serverless functions face concurrency limits and cost challenges.

A hybrid approach combining Kubernetes and serverless architectures may offer the best balance of scalability and efficiency. By leveraging the strengths of both in a hybrid solution, cloud practitioners can optimize resource usage, reduce costs, and lower the carbon footprint of their applications.

Carbon Footprint Calculation Tool

Cloud Carbon Footprint (CCF) is an open-source, multi-cloud solution for calculating the carbon footprint of cloud infrastructure. CCF's transparency allows customization to suit this use cases without dependency on proprietary constraints, ideal for granular carbon analysis.

CCF estimates energy use using cloud usage metrics like CPU, networking and storage. This is converted into energy use coefficients, applying regional Grid Emissions Factors and Power Usage Effectiveness to calculate emissions.

Custom Footprint Calculation

While CCF offers transparency, flexibility, and access to its codebase for calculating the energy use and carbon emissions, its limitations, including a lack of hourly data granularity and consolidated reporting of carbon factors, make it unsuitable for short-duration tests requiring detailed analysis. These limitations stem from the tool's design for long-term tracking rather than real-time assessments.

To address this, a manual calculation was adopted using billing data to reverse-engineer CCF. This allowed for precise hourly granularity, verifies the accuracy of the calculation, and identifies hidden factors that influence carbon estimates, ensuring a more customized and comprehensive analysis.

Methodology

1. Environment setup:

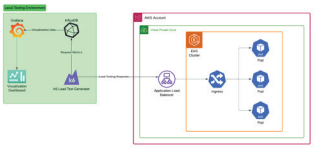
Three different architectures were created to implement the same application, written in Python, that generated a Fibonacci calculation. The hybrid architecture represents a combination of both EKS and Lambda Architectures. The load balancer is configured to route to EKS or Lambda based on a header in the request.

Each cloud environment accepts requests via HTTPS from a local testbed environment, through the K6 load testing tool. The output of these requests are output to an InfluxDB time series database. A Grafana server then reads this data to analyze the data and create visualizations for request performance.

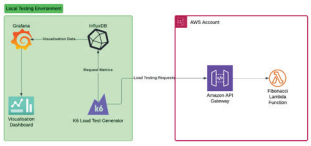
2. Experiment Design:

The test involves simulating the demand pattern of an application that has a predictable period of high demand, with low demand outside of these periods. The test is run over a 10 hour period. The timeframe is separated into five separate stages, an initial intermittent demand period of 90 minutes, followed by a ramp up period of 90 minutes which leads up to a peak demand period which will be sustained for 4 hours. A ramp down period of 90 minutes follows, followed by an additional 90 minute period of intermittent demand. This soak test is performed using the K6 testing framework.

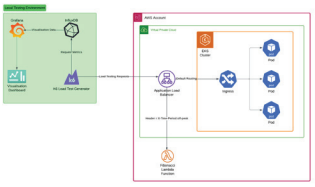
After 24 hours, billing data is then retrieved from AWS via Athena for use in carbon footprint calculation



EKS-Architecture

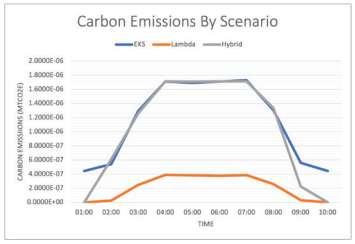


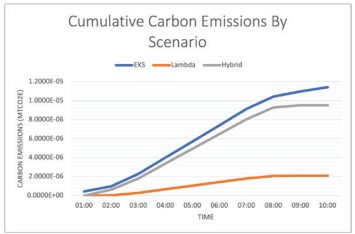
Lambda-Architecture

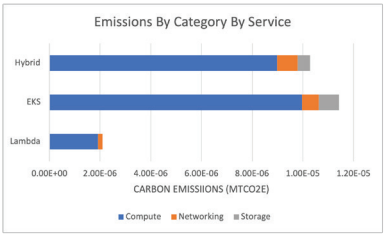


Hybrid-Architecture

Results







Conclusions and Future Work

The tests showed Lambda had a carbon footprint just 18.3% of EKS and better response times, even under peak loads. Lambda was also more cost-effective, though API Gateway added significant expense. A hybrid approach using Lambda during off-peak hours reduced emissions by 9% and costs by 3.25%, if this was extrapolated to the off-peak period for a typical business day, these savings would have represented a figure closer to 20% 7% respectively

However, carbon estimates lacked full accuracy due to limited data on managed services such as the EKS control plane and Lambda infrastructure. Future work could refine these methods, explore longer tests with complex applications, and assess serverless Kubernetes options like AWS Fargate.

QR Code for Recording

A Comparison of Terraform and BICEP Quality Attributes

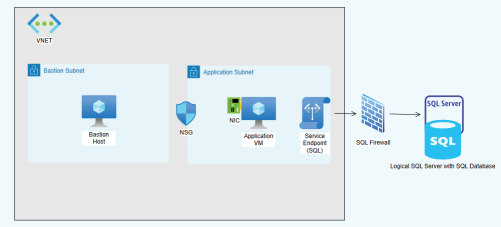
Colm O'Hara
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205717@myTUDublin.ie

Introduction

Infrastructure as Code is often stated to be a key enabler of multi-cloud strategies, however organisations in the earlier stages of cloud adoption may select a single strategic cloud provider for infrastructure and platform services while their cloud management capability matures. In the context of infrastructure resource deployment within a single cloud environment, specifically Microsoft Azure, this study examined whether existing, readily available tools that claim to be "provider-neutral", such as Terraform, offer a significant advantage, in terms of quality attributes such as usability, scalability, maintainability and reversibility, over domain specific tools tailored to deploying and managing infrastructure resources inside a single provider, such as Azure BICEP.

Research Methodology

For all RQs, two separate BICEP and Terraform Codebases were developed based on a sample architecture shown to the right. A set of several experiments aimed to capture how easy it was to develop the codebases using popular tooling and documentation, comparative resource consumption and execution time when deploying the architecture using each codebase, and how effectively BICEP and Terraform dealt with configuration drift post deployment.
The author's own findings regarding Usability, Maintainability and Reversibility were supplemented via a survey circulated to roughly 20 IT professionals with varying degrees of Infrastructure as Code experience.



Usability

Between experience of developing the two codebases and supporting ratings/comments from the survey, Terraform was generally regarded as superior in terms of availability and usefulness of online help, however a larger sample size would be required to determine the if gap between the tools is statistically significant.
No other blockers/drivers to adoption of either tool were widely identified in the survey.

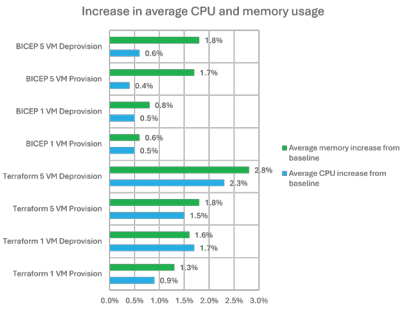
Maintainability and Reversibility

Terraform was better at reporting changes to "in-state" resources, and BICEP had a marginally easier time reconciling changes with code and deleting obsolete resources, but both tools were quite limited in recognising and reverting changes that added new resources.
BICEP could also only delete resources at a resource group level either when reverting changes or tearing the infrastructure down entirely, which could prove problematic when trying to manage enterprise scale deployments.

Performance and Scalability

Overview

The performance tests aimed to deploy and then destroy the sample architecture using both tools' respective codebases, before repeating the tests after adding additional VMs to the architecture to see how the resource usage and execution time was affected as the codebase scaled.

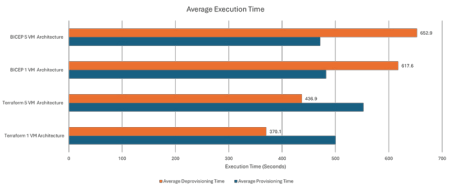


Resource Consumption

Compared to an averaged system performance baseline across ten test series, BICEP tended to exhibit a lower CPU and memory consumption than Terraform, though potentially experienced a faster upward trend in memory consumption when provisioning resources as the architecture grew.

Execution Time

Within the confines of this study, Terraform outperformed BICEP in deprovisioning time, though it showed a larger upward trend as the architecture scaled compared to BICEP. BICEP also showed a possible downward trend in deprovisioning time when one of the more anomalous results was taken out of the sample.



Conclusions and Future Work

Usability

Terraform appeared to be the most popular tool, though ease-of-use and popularity were the only clearly identified selection criteria for IaC tools. Future studies could focus on real-life IaC practices and decision areas regarding choice of tools (e.g. reusability of code, testability, security etc.), asking multiple testers or teams to develop codebases for a sample architecture, as well as aiming to understand/resolve blockers to adoption of more abstracted modelling/Infrastructure from Code tools such as DOML.

Performance and Scalability

BICEP exhibits better performance in most scenarios and seems to gain increased efficiencies as architectures scale. The differences observed weren't substantial at the scale of the architecture deployed, though if larger, more complex architectures are deployed in future studies we might see more pronounced patterns in both resource consumption and execution time. Automation of tests would really help to generate a larger sample size and mitigate the effect of anomalous results.

Maintainability and Reversibility

Both tools have considerable limitations when recognising and handling (or not) configuration drift, and BICEP is limited when it comes to destroying resources at a larger scale. Given the problems identified with a hybrid approach to infrastructure deployment, Strong IaC practices must be enforced to prevent drift.

Investigation Into FinOps Techniques To Optimise Cost in AWS Cloud Deployments.

Denis Parker
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205718@myTUDublin.ie

Introduction

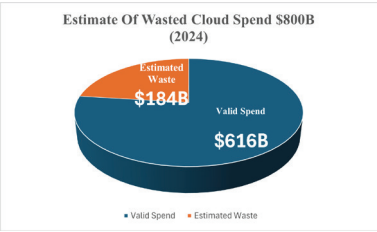


Figure 1: Estimated waste 2025

Cloud infrastructure spending is predicted to approach 50 per cent of organizational IT budgets by 2025. Industry sources expects the Cloud Market to be worth **800M** (**Statista**) in 2024 70 percent of organizations report more than half of their infrastructure exists in the cloud. And 49 percent say they're actively moving more of their data to the cloud. (**Pluralsight**, 2023)
However Flexera estimate that up to 23 per cent of costs incurred by customer does not deliver the intended value to their customers (Flexera, 2024). Sources (AWS) also estimate that up to **25 per cent of cloud spend is wasted.** resulting in wastage of the hundreds of billions of dollars.
There would seem to be an extraordinary amount of money mis-spent on cloud services.
There would also appear to be a lack of academic interest in the cost impacts of cloud architectural decisions. Even though the term FinOps is recent, even standard searches for Cost or Expense on Scholar yield few results.
The objective of this investigation is to present an overview of the FinOps process and demonstrate how it can be used as a measurement characteristic to manage and monitor cloud deployments from a business point of view, and how it should be used as an essential building block of any cloud architecture.
The FinOps Framework is a methodology that attempts to bring the areas of finance and engineering together – using a way of working like DevOps. It contains a set of principles; deployment phases and identifies a set of personas for the people most likely to use and benefit from the use of this framework.
FinOps is a crucial component of sustainability efforts and the move to reducing the environmental impact of cloud computing.

RQ 1

What is the current level of knowledge and understanding of the FinOps framework in Industry professionals?

RQ 2

How could a standard deployment be instrumented to provide FinOps metrics?

RQ3

Given a standard commercial deployment how would an organisation benefit from the implementation of FinOps metrics and practices?

Method

RQ1 FinOps Awareness Survey

The Survey was conducted through direct contact and social media. It was first circulated through direct connections in Industry and Academia followed by a survey posted to LinkedIn.

RQ2 Webserver Deployed with Finops metrics

A small environment architecture was deployed in AWS through the use of a Cloud Formation script. The items in the environment were then adapted to include a set of related tags and a cloud watch agent was configured to report on resource management.

RQ3 Deployment plan for FinOps Program

The purpose of this research question is to investigate the actual implementation of a FinOps framework and methodology in a company. The questions analysed include
What would the tagging policy look like?
What does a sample bill look like?
What metrics are available?
What would a deployment program look like?
What would be the roles and responsibilities for this program?

Results

RQ1 From the analysis of the responses the following themes emerged. Lack of knowledge of Cloud Optimisation strategies, Lack of in-house expertise, Lack of cross department collaboration
RQ2 The deployment of tags to resources for a simple case is extremely easy however scaling this approach to a larger environment such as the Three trier environment which as more than 25 resources would be difficult and require careful planning and preferably Automation
RQ3 FinOps provides a technology and business framework to identify manage and control costs and encourages the inclusion of costs management in the design stages of cloud deployment. Compliance is essential . Strategy decisions should be moved to a central authority such as a **Cloud Centre of Excellence** but local decision making should be kept as close to the developer as possible.

Conclusions and Future Work

There appears to be a lack of academic interest in the cost impacts of cloud architectural decisions. The objective of this investigation is to present an overview of the FinOps process and demonstrate how it can be used as a measurement characteristic to manage and monitor cloud deployments from a business point of view, and how it should be used as an essential building block of any cloud architecture.
Future Suggestions for investigation include AI for FinOps, FinOps for AI, FinOps for SAAS and On-Premises. FinOps for Sustainability and GreenOps

QR Code for Recording

QR Code Goes here

Improving IaC Script Quality: Evaluating Static Analysis Tools and Establishing Best Practices

Isha Rai
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205824@myTUDublin.ie

Introduction

Infrastructure as Code (IaC) revolutionizes cloud management by automating infrastructure provisioning, but poorly written scripts can lead to security vulnerabilities and inefficiencies. This thesis evaluates static analysis tools (Terrascan, Checkov, KICS) for their effectiveness in improving IaC quality and establishes best practices to mitigate risks and enhance security, compliance, and operational efficiency.

Research Objective

RQ1: How effective are static analysis tools in detecting issues in IaC scripts?

RQ2: What are the best practices for high-quality IaC scripts?

Methodology

Script Selection: 20 Terraform scripts representing diverse use cases from public repositories.

Tools Evaluated: Terrascan, Checkov, and KICS.

Evaluation Process: Tools scanned scripts for vulnerabilities, compliance issues, and best practice violations.

Validation: Issues cross-checked for false positives/negatives, contextual relevance, and practical impact.

Key Findings

1. Tool Evaluation:

- Terrascan: Strength in high-severity vulnerabilities (e.g., public access, encryption).
- Checkov: Balanced security and operational checks.
- KICS: Broad detection but higher false positives.

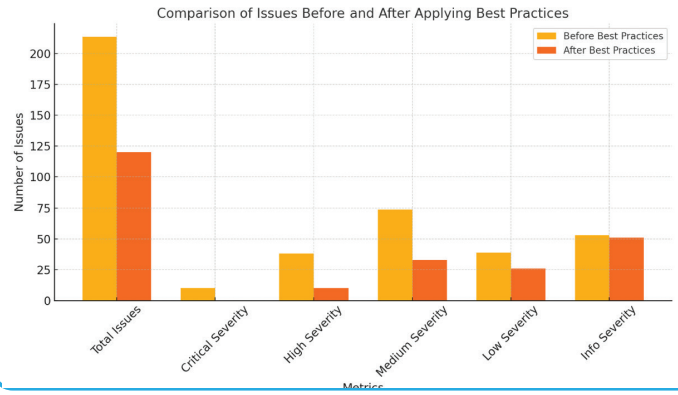
2. Improvements After Best Practices:

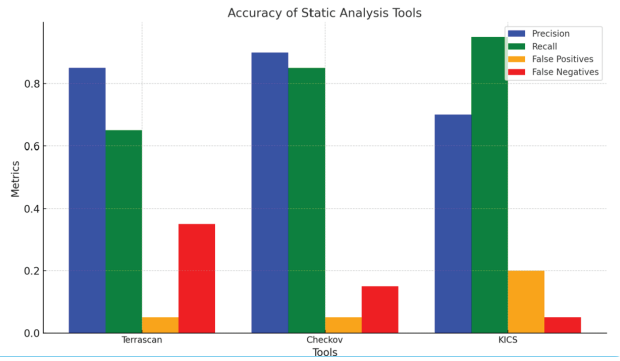
- Critical issues reduced by 100% in most cases.
- High-severity vulnerabilities addressed across all scripts.
- Average improvement: 55-60% reduction in total issues.

3. Comparative Insights:

- Terrascan: Best for compliance checks.
- Checkov: Versatile with robust usability.
- KICS: Comprehensive but noisy; useful for complex scripts.

Quantitative Evaluation of Results





Conclusions and Future Work

Combining Terrascan, Checkov, and KICS ensures comprehensive IaC script analysis. Implementing best practices significantly improves security, operational efficiency, and compliance.

Future work in this area involves evaluating additional static analysis tools and their effectiveness across diverse IaC frameworks, such as AWS CloudFormation and Azure Resource Manager templates. Integrating contextual factors into static analysis tools could help reduce false positives and negatives, improving their practical utility. There is also significant potential in exploring AI/ML techniques to enhance detection capabilities and deliver more accurate results. User-focused qualitative surveys could provide valuable insights into tool usability, integration challenges, and the applicability of best practices in real-world workflows. Additionally, extending research to hybrid, multi-cloud, and edge computing environments will enable the assessment of IaC practices and tools in increasingly complex and scalable infrastructure setups.

QR Code for Recording

Nomad vs. Kubernetes

Alexandru Constantin Cardas
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
x00205805@mytudublin.ie

Introduction

Containers have become the de facto standard for handling application deployments at scale. Their popularity has given rise to the microservice architecture which solves the problem of scalability and isolation of concerns. However, microservices are not without their disadvantages. They can become difficult to manage once the production level is achieved and can get complicated once they reach a certain size. This is why this architectural pattern needed another way to be managed. The solution that emerged from this pitfall is the container orchestrator. These let teams and businesses manage vast amounts of containers and services in an abstracted fashion without having to interact with any of them manually. They have become crucial to the DevOps world and it is worth discussing what options are available in this domain so that teams can choose the correct tool for their use case. This paper compares two container orchestration platforms. Kubernetes as a managed service in AWS (EKS) and a self-deployment of Nomad cluster in AWS.

Performance

From the testing, it is clear that Nomad outperforms Kubernetes in all scenarios when it comes to application performance and even in most for resources allocated. However, performance is not always the driving factor in adopting a certain technology. Other factors such as time investment to setup is the major distinguishing one that was apparent while setting up the clusters for testing. Nomad requires manual setup and management, since it does not have a managed service provided by AWS.

Cost

The total cost of the Nomad servers is 20.15 USD per month while the price of the EKS cluster is 73.00 USD. The price of the EKS cluster is roughly **3.62** times higher than that of using a set of three servers in Nomad. These are also fixed prices for the test scenarios that are presented in this paper. For larger environments that require more than seven clients each, the size of the Nomad servers would have to go past the micro size which, in turn, would make it more expensive, while the price of the EKS cluster would remain the same.

Extras

1. Load Generation:

For generating a load on the system, K6 is used. K6 is an open source load testing tool that is used to generate the load on systems. It is used in conjunction with the Prometheus metrics originating from K6 to get more information on the number of requests and the latency of each. The executor used for K6 is ramping-vu so that Virtual Users (VUs) can scale up based on the target. Each VU has its own number of requests that it can produce.

2. Metric Collection:

The approach used to collect metrics is via Grafana and Alloy. Grafana is the full stack observability platform that allows for the collection and storage of metrics through its components. For this scenario only the metrics are being used, and these are as part of Prometheus. Alloy is the vendor-neutral OpenTelemetry (OTel) collector. Alloy is deployed as a system job on the Nomad and as a DaemonSet on the Kubernetes clusters.

3. Consistency:

Ensuring consistency in terms of capabilities between Nomad and Kubernetes is a challenge. This is due to the way Kubernetes manages the infrastructure as part of its managed state within AWS. Kubernetes under EKS automatically installs the required APIs and add-ons that Kubernetes needs to operate. Consul from HashiCorp is used because of its capability to provide an automatic auto-joining of Nomad servers. It is also the basis for a service-mesh, although not used in this paper, it must be deployed on both Nomad and Kubernetes to ensure consistency across the deployed applications. Consul setup is slightly different from Nomad and Kubernetes. Since the control-plane is managed by EKS, it is not possible to add any extra services to those nodes.

Topic Overview

Conclusions and Future Work

Based on the results shown in this document, Nomad performs better in smaller environments, especially because it is not limited by a safety limit of IP addresses allocated per node. It also benefits from the fact that it does not have any hops in between requests to reach a service via its IP and port, like Kubernetes needs to do. It is able to surpass it in every scenario regardless of the cluster size.
Kubernetes could be deployed using a mixture of Network Load Balancer and Application Load Balancer to use IP based target groups rather than instances. This is a best practice that should provide a faster DNS resolution. Furthermore, it simplifies the configuration, and it will reduce the latency because the traffic no longer needs to pass through the Kube-proxy and instead it will flow directly through to the Traefik application proxy. Another avenue, although very difficult to do, would be to self-host Kubernetes as well with tools such as kOps where a fully production ready Kubernetes cluster would be created.

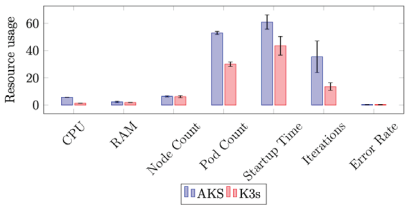
A Comparison of AKS and K3s on VMSS

Craig Dillon
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
x00205790@mytudublin.ie

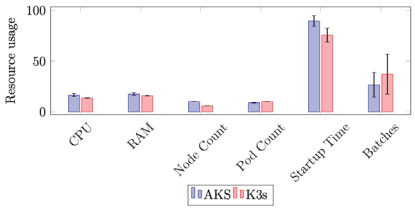
Introduction

This study presents a comprehensive comparison of AKS and K3s running on VMSS, focusing on performance, scalability, cost-effectiveness and operational complexity. Test environments were deployed using Terraform, tested under identical workloads and metrics were collected by Prometheus and Grafana.
Test applications used were Google Online Boutique and TensorFlow, tests were performed at the same time for 30 minutes each and repeated 5 times. Each application was tested independently and simultaneously.

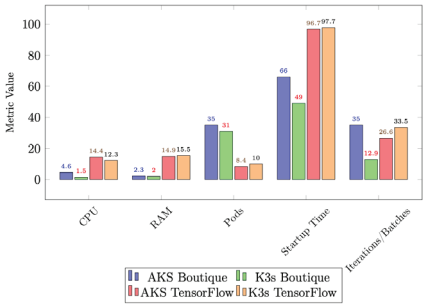
Results



Online Boutique results



TensorFlow results



Simultaneous test results

Research Questions

RQ1 - Performance: How does each platform perform under different workloads, both individually and simultaneously?

K3s and AKS showcase distinct strengths in efficiency and performance. K3s excels in resource efficiency, with lower CPU and RAM consumption, faster pod scaling, and consistent performance, performing strongly in batch-processing workloads. In contrast, AKS offers superior node scalability and handles multi-component applications more effectively, albeit with higher resource usage and variability.

RQ2 - Scalability: How does each platform respond to increases in demand, how effectively does it scale resources and what impact does this have on performance?

AKS demonstrates faster node scaling, making it well-suited for workloads requiring rapid infrastructure adjustments. However, K3s outperforms AKS in pod startup time, enabling quicker workload readiness and more efficient resource utilization at the application level.

RQ3 - Cost Effectiveness: What are the financial implications of deploying these platforms, how does the cost relate to the relative performance?

K3s running on VMSS costs 29.3% less than AKS under similar low load scenarios and 15.3% less under intense testing loads. When running simultaneous workloads there is an overall performance impact on each individual workload, where K3s had a vastly lower cost per 1% in performance drop, €1.66 compared to AKS at €9.95 per 1% drop in performance.

RQ4 - Operational Complexity: What are the differences in deployment and management complexity between each platform?

The deployment of AKS was streamlined and efficient, with Terraform offering options to manage almost every possible configuration in deploying a cluster. Deploying K3s on VMSS required a much more involved, sophisticated approach. The deployment was created using a combination of Terraform, cloud-init and custom Bash scripts totalling over 400 lines of code compared to 33 for AKS.

Conclusions and Future Work

The findings of this study demonstrate that both AKS and K3s have distinct advantages and limitations when deployed on similar Azure based environments. AKS features a streamlined setup process with automated management of many deployment components and showed greater stability under intense load. However, this came at a financial cost and the reliance on Azure native integrations may make it less appealing for cost-sensitive deployments or scenarios that require a high level of customisation.

K3s, in comparison, operated with a smaller footprint overall and performed strongly in TensorFlow tasks at a significantly lower cost than AKS. The increased engineering effort and complexity of deployment make this a less suitable option for organisations who have a preference for rapid scalability without deep technical knowledge.

Future work could include comparing static node performance, optimizing configurations for greater efficiency, evaluating additional distributions like Minikube and OpenShift, and testing a broader range of applications to gain deeper insights into platform performance across diverse workloads.

Exploring Rust's Performance in a Serverless Environment

Saoirse Mullen
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205707@myTUDublin.ie

Introduction

This research examines Rust's potential in serverless computing, a transformative cloud development paradigm that allows scalable application deployment without infrastructure management. Serverless architectures offer benefits such as increased agility, reduced complexity, and optimized resource utilization. The study investigates how programming languages, particularly Rust, impact these advantages in terms of performance and cost.
Using Google Cloud Run, a managed serverless platform for containerized deployments, the research benchmarks Rust's performance against other programming languages. Key metrics include execution speed, cold start performance, and cost efficiency. Google Cloud Run's scalability and simplicity make it an ideal environment for analyzing Rust's strengths without infrastructure management overhead.
The study evaluates Rust's capabilities in serverless environments, focusing on cold and warm start times, resource utilization, and computational efficiency. By highlighting Rust's potential to address challenges in serverless computing, this research contributes to understanding programming languages' roles in shaping the future of cloud development.

Rust

Rust is known for its performance, memory safety, and developer-friendly features. Its ownership model and compile-time checks prevent vulnerabilities like buffer overflows, ensuring secure code. Combining C/C++-level control with safety and concurrency, Rust is ideal for performance-critical applications like operating systems and web services. Its growing popularity is supported by strong community backing, detailed documentation, and tools like Cargo, which simplify development. Rust enables developers to build reliable, efficient, and scalable software.

Serverless

Serverless computing lets developers build and deploy applications without managing infrastructure. Cloud providers handle provisioning, scaling, and maintenance, allowing developers to focus on code. Event-driven and pay-per-use, serverless architectures reduce complexity, enhance agility, and optimize costs. Platforms like AWS Lambda and Google Cloud Run enable scalable and efficient applications with minimal overhead.

Experiment

1. Experimental Design and Workloads:

To evaluate Rust's performance in serverless environments, three workloads were selected: factorial for lightweight computation, a prime number finder for moderate tasks, and fannkuch-redux for CPU-intensive processing.These functions, implemented in Rust, Go, and C++, were chosen for their simplicity and compatibility across languages to ensure consistency in evaluation.

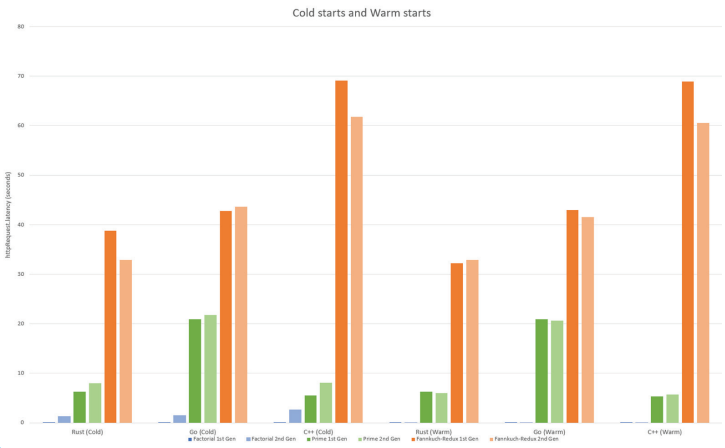
2. Comparing Rust with C++ and Go:

C++ offers raw performance but lacks built-in memory safety, leading to vulnerabilities like buffer overflows. Growing concerns over security have prompted advocacy for safer languages, such as Rust. Rust combines C++-level performance with strict memory safety guarantees, modern tooling, and reduced debugging efforts. In contrast, Go prioritizes simplicity and productivity but relies on garbage collection, which may introduce latency in performance-critical serverless applications.

3. Deployment and Performance Metrics:

The workloads were deployed on Google Cloud Run using both 1st and 2nd Generation Execution Environments. Metrics such as cold and warm start latencies and costs were analyzed to assess performance differences. This comprehensive data provided insights into container startup latency, HTTP response times, and resource utilization, effectively benchmarking Rust's strengths against Go and C++.

Overview - Key Findings



- 1st Gen Environment: Cold starts were consistently faster across all tested languages, with minimal latency differences between cold and warm starts, making it ideal for short-lived, on-demand workloads.
- 2nd Gen Environment: Cold starts were slower with higher latency variability, but once warmed up, the environment provided faster response times, indicating improved steady-state performance.
- Rust vs Go: Rust outperformed Go in cold start latency, demonstrating better efficiency during the initial spin-up phase in both environments.
- Rust vs C++: C++ was slightly faster than Rust in some cold start scenarios, but once warmed up, Rust's performance matched C++ in warm scenarios.
- Serverless Efficiency: Rust proved to be highly efficient in serverless computing, especially in workloads requiring performance consistency, competing closely with C++ and surpassing Go.

Conclusions and Future Work

Rust demonstrates significant potential in serverless computing, combining high performance with strong safety guarantees. It competes effectively with established languages like Go and C++ despite differences in cold start behaviors. Rust's efficiency and competitiveness make it a solid choice for serverless environments, with potential for future work involving a broader range of workloads and platforms. As serverless technologies evolve, Rust's ability to handle demanding tasks efficiently positions it as a strong candidate for such workloads.

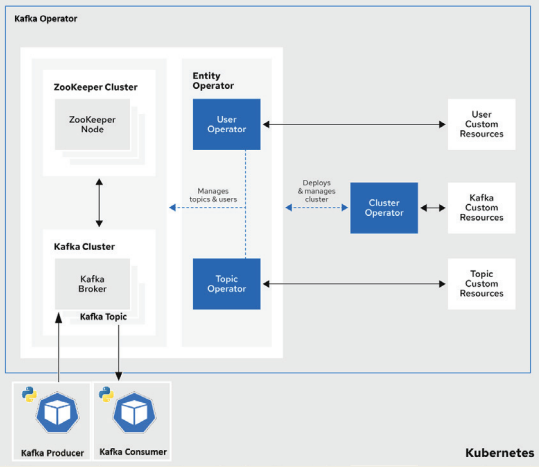
QR Code for Recording

Comparison of Kafka Operators: Strimzi vs Koperator vs Confluent: A Comprehensive Industry Analysis

Ankit Antony
Department of Computing, TU Dublin, Tallaght, Ireland
x00193216@myTUDublin.ie

Introduction

*Apache Kafka is a high-throughput, low-latency event streaming platform widely used in industries like finance and telecommunications for real-time data processing. Kubernetes, the industry standard for container orchestration, offers automated deployment, scaling, and fault tolerance, simplifying the management of containerized applications. Deploying Kafka on Kubernetes combines Kafka's real-time streaming capabilities with Kubernetes' orchestration features but introduces challenges such as resource allocation and scaling due to Kafka's stateful nature. To address these complexities, Kafka operators have been developed. These Kubernetes-native tools automate tasks like provisioning, scaling, and monitoring Kafka clusters, enabling efficient management in containerized environments. This thesis examines the challenges of managing Kafka clusters in dynamic environments and the role of Kubernetes Operators in simplifying deployment, scaling, and maintenance. It compares three Kafka operators—**Strimzi, Confluent Operator, and kOperator**—evaluating their performance, resource efficiency, operational costs, and user experience in real-world scenarios. By analysing metrics like throughput, latency, resource usage, and ease of management, the study provides insights into each operator's strengths and limitations. The findings aim to guide organisations in optimising Kafka cluster management to meet their unique technical and business needs.*



Environment

- **Host machine:** Ubuntu 22.04 LTS with quad-core CPU, 16 GB RAM, 256 GB SSD
- **Minikube cluster:** 4 CPU cores, 8 GB RAM, 50 GB disk space
- **Virtualization:** Linux KVM

Research Questions

- ***RQ1:*** *How does the resource consumption and scalability of different operators affect the performance of Kafka clusters under varying workloads?*
- ***RQ2:*** *What is the total cost of ownership (TCO) trade-off between open-source and enterprise-grade Kafka operators when factoring in licensing, infrastructure, and operational costs?*

Methodology

This study employs a quantitative, experimental approach to evaluate Kafka operators, focusing on key performance metrics and operational efficiency. The testing environment includes a Kafka cluster deployed on Minikube, featuring brokers, Zookeeper, and monitoring tools like Prometheus, Grafana, and cAdvisor. A custom Python-based Kafka client application generates realistic workloads by producing and consuming order data to assess the operators' performance. The testing procedure involves simulating order data with the Faker library and Kafka-python package. Kafka topics are monitored using Kafka-UI, with external services exposed via Minikube. Performance metrics, such as resource usage and throughput, are analyzed under various workloads to ensure a detailed comparison of the operators' capabilities and efficiency. To facilitate localized testing, Docker images of the client applications are built and deployed as Kubernetes pods on Minikube. Logs from the producer and consumer applications confirm successful data generation, publishing, and consumption. This comprehensive methodology enables a robust performance evaluation of Kafka operators in containerized environments.

Feature Comparison Criteria and Results

The table represents the comparison between different operators undertaken based on Research Methodology. The comparison focuses on key criteria that are crucial for organizations when selecting a Kafka operator for their Kubernetes environment. These criteria include:

- **Resource Consumption:** Evaluating CPU usage, memory usage, and disk I/O performance under load conditions
- **Total Cost of Ownership (TCO):** Breaking down the costs into licensing, infrastructure, and operational expenses
- **Installation and Deployment:** Assessing the average setup time and configuration complexity
- **Authentication:** Comparing the support for TLS and SASL protocols

Criteria	Confluent	Koperator (Banzai Cloud)	Strimzi
Resource Consumption			
CPU Usage (avg under load)	40%	-	35%
Memory Usage (avg)	3.5 GB per broker	-	3.2 GB per broker
Disk I/O Performance	750 MB/s	-	800 MB/s
Total Cost of Ownership			
Licensing Costs	€5,000/year	€0	€0
Infrastructure Costs	€10,000/year	€9,500/year	€8,500/year
Operational Costs	€30,000/year	€27,000/year	€25,000/year
Installation/Deployment			
Setup Time (avg)	2 hours	1.75 hours	1.5 hours
Config Complexity (scale 1-10)	5/10	4/10	3/10
Authentication			
TLS Support	Yes	Yes	Yes
SASL Support	Yes	Yes	Yes

Conclusion and Future Work

The thesis evaluates three Kafka operators—Strimzi, Confluent, and Koperator (Banzai Cloud)—for managing Kafka clusters in Kubernetes environments. Strimzi stands out for cost-effectiveness and simplicity, suited for small to medium businesses, while Confluent offers enterprise-grade tools at a higher cost, ideal for reliability-focused organizations. Koperator excels in Kubernetes-native features but faces challenges with limited community support and documentation. The study highlights that the choice of operator should align with organizational needs rather than a one-size-fits-all approach, aiding in optimizing data streaming architectures. Future research could explore performance testing under varied workloads to identify limits and scalability, multi-cloud and hybrid deployments for consistency and latency management, and advanced security features for compliance with regulations like GDPR. Integration with emerging technologies such as AI/ML, edge computing, and IoT could also provide insights into real-time data processing. Additionally, user experience studies can bridge gaps between theoretical capabilities and practical application, further enhancing the usability and adaptability of Kafka operators.

Video

Analysis of functional programming languages for use in serverless lambda functions on AWS platform

William Spain
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205706@myTUDublin.ie

Introduction

Serverless computing has emerged as one of the most popular and rapidly evolving segments of cloud computing, moving far beyond its once limited scope. Modern providers are offering increasingly flexible features such as custom runtimes, allowing developers to run specialized or even deprecated languages when a particular use case demands it. In this study, we investigate the viability of custom runtimes by examining three functional programming languages Haskell, Elixir, and Clojure on AWS Lambda. These languages represent a distinct departure from the mostly object oriented native runtimes traditionally provided. Through a series of experiments, we measure cold start times, package sizes, memory usage, and execution performance to understand how these functional languages perform and whether they can effectively compete with or even outperform the default AWS Lambda offerings.

Research Questions

RQ1. *what is the status of non standard runtimes in the main cloud platforms?*

RQ2. *how dynamic are AWS custom run times?*

RQ3. *What is the best performing functional programming language on AWS lambda out of the ones tested?*

Methodology for analysis

This analysis used seven functions: **Hello World, GCD, Fibonacci, Sieve of Eratosthenes, Binary Search, Matrix Multiplication, Word Frequency.** All except for Hello World had low and high intensity inputs. Each scenario was invoked 12 times (2 warmups, 10 measured). For cold starts, Hello World ran 5 times, each 15 minutes apart. A Python script was used to invoke Lambdas and gathered logs from CloudWatch. The four key metrics were: **Package size, Cold start time, Max memory used, Execution duration**

Research Question 1 and 2

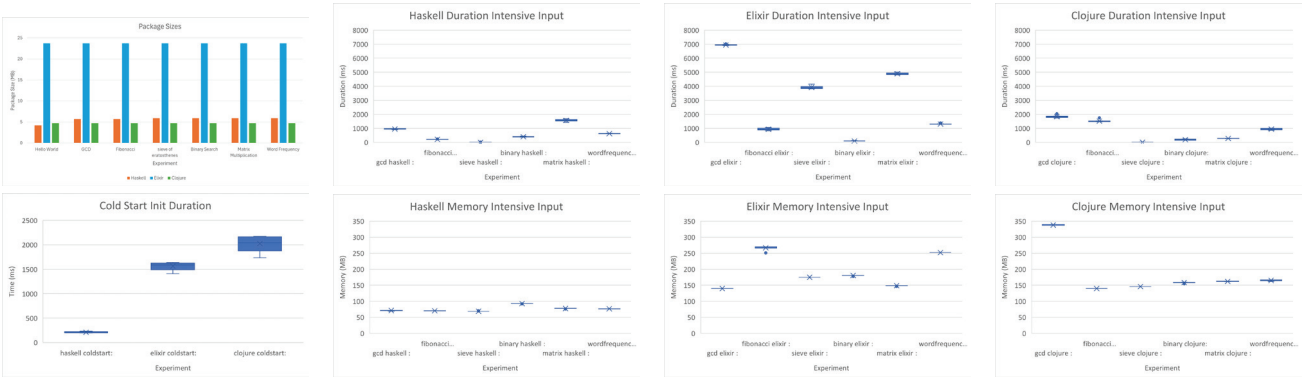
1. Research Question 1:

AWS introduced custom runtimes in 2018, backed by extensive documentation and community tutorials. Azure Functions require that a user should write code that starts a custom web server that will handle the code iteration. GCP generally pushes container based approaches instead of direct runtime customization. AWS is the most straightforward and well supported choice for non-native runtimes.

2. Research Question 2:

AWS imposes no strict language limits; any compiled or scripted runtime works if it can interact with the Runtime API. A 50 MB compressed package limit allows bundling larger environments, and minimal compatibility checks with Amazon Linux 2023 are needed. Custom runtimes also integrate with AWS services and features.

Experiment Results



Conclusions and Future Work

Haskell consistently outperformed Elixir and Clojure in AWS Lambda, thanks to its small package size, faster cold starts, and efficient memory usage. While Elixir benefits from the BEAM runtime and Clojure accesses the Java ecosystem, both exhibit slower startup times and larger package sizes, making them less appealing for cost and speed sensitive scenarios. AWS Lambda itself offers the strongest custom runtime support among the major platforms, with Azure's custom handlers less documented and GCP lacking direct options at the time of this study.

Further testing could explore more complex Lambdas, larger data sets, and additional trigger methods. Investigations into container-based vs. custom runtime approaches, deeper analysis of native runtimes, and systematically testing a broader range of functional or esoteric languages could shed additional light on performance across diverse workloads. Varying test times and multiple day studies might reveal how resource availability affects speed and reliability, especially in mission-critical deployments.

QR Code for Recording

Evaluating Kubernetes Security Mechanisms for DOS Prevention: A Comparative Analysis of Multi-layer Protection Strategies

Sergej Dikun
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205785@myTUDublin.ie

Introduction

Kubernetes is a powerful platform for managing containers, but its popularity makes it a major target for Denial of Service (DoS) attacks that can disrupt services and drain resources. This thesis examines six key security tools—Rate Limiting, Resource Quotas, Network Policies, Role-Based Access Control (RBAC), API Rate Limiting, and Service Meshes —to understand how they protect against these attacks. Using a controlled test setup with Minikube, each tool was tested for its ability to block attacks, its impact on system performance, and how easy it is to use.

Objectives

- Evaluate the effectiveness of Kubernetes policy-based mechanisms in mitigating DoS attacks.
- Analyze the performance impact of these mechanisms, both individually and in multi-layered configurations.
- Develop recommendations for combining policies to achieve effective and resource-efficient DoS protection.

Research Questions

1. How effective are policy-based mechanisms (e.g., Rate Limiting, Resource Quotas, Network Policies) in mitigating DoS attacks at different layers in Kubernetes clusters?
2. What is the performance impact of individual policy-based mechanisms on Kubernetes cluster operations?

Performance Impact and Security Strategies

1. Performance metrics:

The tools tested had varying impacts on performance. Lightweight tools like RBAC and Resource Quotas used minimal resources, with RBAC adding less than 2% CPU and around 20 Mi of memory. Resource Quotas increased CPU usage by 5% and memory by 25 Mi under heavy loads. Advanced tools like Service Mesh required more resources, with up to 25% CPU and 500 Mi of memory during high traffic.

2. Scalability:

Lightweight tools are best for small clusters due to low resource usage. Advanced tools like Service Mesh handle complex traffic well, but require higher resources. Service Mesh maintained stability under high traffic but needed up to 25% CPU and additional memory.

3. Trade-offs and Recommendations:

Lightweight tools, such as RBAC and Resource Quotas, are easy to set up and effective for small clusters. Advanced tools like Service Mesh offer better protection for large clusters but are more resource-intensive. Combining these tools in layers provides both foundational and advanced security.

Minikube: v1.34.0
Kubernetes: v1.31
Memory: 4096 MB
CPU cores: 2

Topic Overview

Mechanism	CPU Usage	Memory	Performance	Effectiveness
Rate Limiting	10-15%	50Mi	15ms	★★★★
Resource Quotas	5%	25Mi	3s	★★★★
Network Policies	3-5%	50Mi	100%	★★★
RBAC	<2%	20Mi	Strict	★★★★
API Rate Limiting	10-20%	100Mi	High	★★★★★
Service Mesh	20-25%	500Mi	Advanced	★★★★★

Solution	CPU Overhead	Memory Overhead	Resource Impact Summary
Rate Limiting	9.85	~50Mi	Moderate increase under traffic
Resource Quotas	0.05	~25Mi	Negligible overhead
Network Policies	2.95	~50Mi	Minimal impact
RBAC	0.98	~20Mi	Almost negligible
API Rate Limiting	9.8	~100Mi	Noticeable overhead under load
Service Mesh	19.75	~500Mi	High resource consumption

Conclusions and Future Work

Conclusions

Study evaluated six Kubernetes security mechanisms, highlighting the strengths and limitations of each. Lightweight tools, such as RBAC and Resource Quotas, provide foundational security with minimal resource use, ideal for smaller clusters. Advanced tools like Service Mesh offer strong protection for large-scale deployments but require significant resources. A layered approach that combines these tools is essential for addressing diverse security challenges while maintaining performance.

Future Work

Future research should focus on:

- **Scalability:** Improving synchronization of security policies across hybrid and multi-cloud environments.
- **Automation:** Integrating AI for threat detection and policy optimization.
- **Efficiency:** Reducing resource usage of advanced tools like Service Mesh.

These areas aim to make Kubernetes more secure, scalable, and efficient for modern deployments.

QR Code for Recording

Examining Terraform and Bicep: A Comparative Analysis of Infrastructure as Code Tools for Provisioning Azure Environments

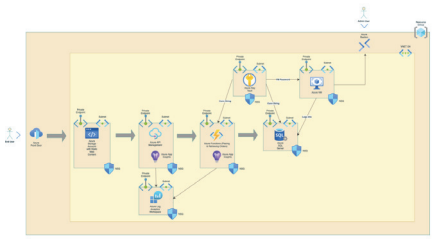
Richard Coffey
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland

Introduction

Infrastructure as Code (IaC) revolutionizes cloud infrastructure management by enabling automation, consistency, and scalability. This research empirically compares Terraform and Bicep, two key IaC tools for Azure, across six criteria: deployment time, plan accuracy, drift detection, state management, community support, and syntax readability. Terraform, with its stateful architecture, excels in planning efficiency, detailed outputs, and a vast ecosystem suitable for multi-cloud use. Bicep, an Azure-native tool, offers concise syntax, seamless ARM integration, and simplicity, though it has longer processing times and a smaller ecosystem. Using a complex Azure serverless architecture and automated CI/CD pipelines, the study highlights Terraform's suitability for hybrid or large-scale projects, while Bicep is ideal for Azure-specific deployments prioritizing readability and speed. This research provides actionable insights for selecting IaC tools based on project needs and complexity. Future studies could explore tools like Pulumi or assess multi-cloud applicability for broader insights.

Criteria Examined

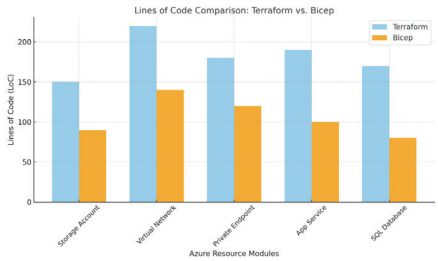
Project's Test Azure Architecture



Research Methodology and Findings

A serverless e-commerce architecture was developed as the testbed, to compare the tools under various criteria. Azure DevOps pipelines were used to deploy the architecture, enabling consistent measurement across both tools. The methodology adopted a mixed approach, combining quantitative timing measurements and qualitative assessments. For **timing execution**, the pipelines were instrumented to capture Plan and Apply durations, distinguishing between processing times and overall times. This revealed how each tool's architecture—stateful for Terraform and stateless for Bicep—impacted deployment efficiency. **Plan accuracy** was assessed by comparing the predicted and actual resource configurations for the architecture. False positives, output verbosity, and clarity were evaluated to gauge the tools' suitability for complex deployments. **Drift detection** tests were conducted by manually altering resources and comparing each tool's ability to identify discrepancies. **State management** analysis involved setting up Terraform's state file in Azure and comparing its functionality to Bicep's reliance on Azure Resource Manager (ARM). The overhead of managing Terraform's state and the implications for collaborative workflows were key considerations. **Community and ecosystem support** were compared using metrics like module availability, provider diversity, and community engagement. Finally, **syntax readability** was evaluated based on code conciseness and clarity, reflecting the tools' usability for developers. This comprehensive methodology ensured a well-rounded comparison of the two tools.

Comparison Graphs for 2 out of the 6 criteria measured as part of this project



Conclusion

This research provides an extensive comparison of Terraform and Bicep for Azure deployments, evaluating six criteria: deployment time, plan accuracy, drift detection, state management, community support, and syntax readability. A rigorous experiment using a complex Azure test environment revealed Terraform's efficiency in planning stages due to its stateful architecture, while Bicep's reliance on ARM caused longer processing times but allowed real-time evaluations. Terraform excelled in granular plan accuracy and drift detection of predefined resources, whereas Bicep detected untracked changes, showcasing complementary strengths.

Terraform's state file enables precise tracking but requires additional management complexity, unlike Bicep's stateless simplicity. Terraform's robust ecosystem supports hybrid and multi-cloud scenarios, while Bicep benefits from Azure-native integration for streamlined deployments. Bicep's concise syntax suits Azure-specific projects, while Terraform's verbosity supports modularity. The study highlights that selecting between the tools depends on project scale, complexity, and infrastructure focus.

Pro's and Con's

Aspect	Terraform	Bicep
Deployment Speed	✔ Efficient for iterative changes	✘ Slower processing due to stateless design
Plan Accuracy	✔ Detailed and precise forecasts	✘ Potential for false positives in complex plans
State Management	✘ Requires setup and maintenance	✔ Simplified stateless approach
Community Support	✔ Large community and extensive modules	✘ Smaller community and fewer reusable modules
Syntax Readability	✘ Verbose for simpler setups	✔ Concise and Azure-specific
Drift Detection	✔ Efficient for predefined changes	✔ Comprehensive real-time detection
Vendor Lock-In	✔ Multi-cloud flexibility	✘ Strong Azure dependency
Ease of Use	✘ Steeper learning curve for Azure-specific use	✔ Intuitive for Azure users

Future Work

This research compared Terraform and Bicep for Azure deployments, highlighting opportunities for future exploration in Infrastructure as Code (IaC). Pulumi, with its support for languages like Python and TypeScript, offers a flexible alternative for multi-cloud use. Future studies could examine stateful versus stateless IaC tools, long-term cost implications, and security in regulated industries. Multi-cloud and hybrid deployments, error handling, and rollback mechanisms also warrant attention. Emerging developments like OpenTofu and Azure portal template exports could impact adoption.

QR Code for Recording

Comparing Kubernetes and Nomad for hosting .NET legacy workloads in Azure

Luke Osborne
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00205719@myTUDublin.ie

Introduction

This research evaluates the suitability of Kubernetes and Nomad for orchestrating legacy .NET Framework workloads within modern IT environments. Driven by increasing regulatory demands and the need for improved security, enterprises face significant challenges in modernising these critical systems. Concerns surrounding potential downtime, limited in-house expertise, and the cost of updating outdated codebases often hinder modernisation efforts. To assess the feasibility of these platforms, a comprehensive performance testing framework was developed.
This framework involved evaluating various workloads across equivalent node configurations on both Kubernetes and Nomad, measuring performance metrics and resource consumption.

Test Setup

Objective: Evaluate the performance of applications deployed on AKS and Nomad.

Tools: Bash script and k6 load testing tool.

Iterations: 3 applications (small, medium, large), 5 tests, each lasting 5 minutes.

Test Workflow *(Repeated for Each Iteration)*:

Ramp-Up: Gradually increases to 10 virtual users over 30 seconds.

Stable Phase: Maintains 10 users for 4 minutes to simulate steady usage.

Ramp-Down: Reduces back to 0 users over 30 seconds.

Hosting Costs

This cost analysis represents the base setup for Nomad and Kubernetes (AKS), calculated using the Azure Pricing Calculator. Nomad shows lower costs due to its lightweight design, while Kubernetes incurs higher costs for its advanced features and resource demands.

Platform	OS	Instance	Cost	Total
Nomad	Linux	Standard_B1s	€8.11	
Nomad	Linux	Standard_B1s	€8.11	
Nomad	Linux	Standard_B1s	€8.11	
Nomad	Windows	Standard_D2s_v3	€143.32	€167.65
AKS	Windows	Standard_D2s_v3	€143.32	
AKS	Linux	Standard_DS2_v2	€91.94	€235.26

Research Questions

1. Compare the performance of migrating a legacy application to Nomad and Kubernetes:

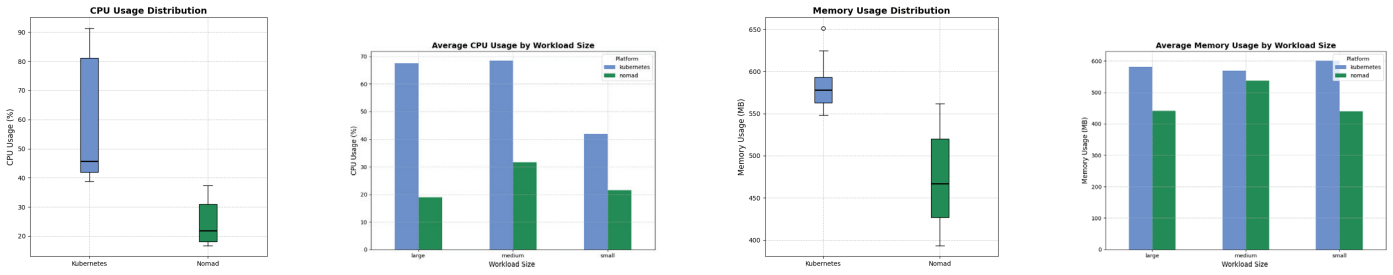
The research will conduct an in-depth comparative analysis of the performance of multiple legacy applications migrating to both Nomad and Kubernetes. This includes identifying the computational and resource overhead introduced by each orchestration platform. The performance metrics evaluated will encompass latency, resource utilisation (CPU, memory), and throughput. The goal is to assess the effect of each platform on the application's operational efficiency and measure the trade-offs involved in choosing one platform over another. This analysis will provide actionable insights into the suitability of Nomad and Kubernetes for different application contexts.

2. Identify the challenges associated with the workload migration:

The second goal focuses on systematically identifying and categorising the challenges when migrating a legacy application to Nomad and Kubernetes. This encompasses technical difficulties, including compatibility issues, configuration complexities, and dependency management, alongside operational obstacles such as deployment automation and monitoring. Additionally, the research will investigate possible mitigations or solutions for these challenges. Ultimately, this objective seeks to thoroughly understand the practical barriers enterprises may encounter while transitioning legacy applications to modern orchestration platforms.

Objective: The research aims to advance the understanding of legacy application orchestration by examining trade-offs and migration complexities. It seeks to provide valuable insights for both practitioners and researchers, offering guidance on addressing these challenges. Additionally, the findings will support enterprises in making informed decisions about similar transitions, ensuring their strategies consider both technical and operational factors.

Testing Results



Conclusions and Future Work

Final Conclusion: The findings show that while both platforms perform similarly, Kubernetes has higher resource utilization but suffers from slow container startup due to large Windows base images. Nomad lacks enterprise-grade features and has unreliable IIS drivers, limiting its suitability for large-scale production.

Future Work: Future work could explore hosting legacy .NET applications on platforms like Azure App Service, evaluate migration tools (e.g., Amazon Q Developer), and develop cost estimation models and migration prerequisites for practical guidance.

QR Code for Recording

NOTES