Title: “**Enhancing TLS Performance in Kubernetes: Evaluating Cipher Suite Selection Strategies**”

This thesis aims to conduct a comparative analysis of Secure Sockets Layer (SSL) cipher suites to evaluate performance and suitability within containerized environments. This thesis will also take into account certificate adoption and security implications on their adoption within these environments. SSL cipher suites play a crucial role in establishing secure communication channels over the internet, and their selection can significantly impact the overall security posture of an application or system. By examining a range of cipher suites, this research seeks to provide valuable insights into their strengths, weaknesses, performance and optimal usage.

Chapter 1: Introduction

1. Background and Motivation
   1. Kubernetes is becoming de facto platform for today’s development especially in microservices and the transition from monolithic systems to API based decomposed solutions. Kubernetes provides many benefits due to scalability, performance, failover etc and supporting agile development.
   2. SSL strategy is quite key to secure implementations. Nevertheless TLS offloading is a service more and more offered by major cloud providers now with their proprietary managed certificates, interesting to compare this approach.
   3. Take advantage of solid architecture & fine grained monitoring tools for reliable results comparison.
   4. Full end to end TLS is the recommended approach with secure DB connection and secure data at rest, will act as the first comparison.
   5. Varying cipher suites and post quantum TLS ciphers, to investigate effect on performance for newer cipher suites
2. Research Objectives
   1. Ensure a solid architecture is in place to have comparative results, with a strong toolset to monitor applications and provide breakdown on performance areas.
   2. Isolate:
      1. app processing
      2. network
      3. SSL handshake
      4. Crypto

to be able to do an effective comparative analysis

1. Research Questions
   1. Post quantum within containerized kubernetes environments is still very new. May be challenging to get this working.
2. Scope and Limitations
   1. Will be based on a simplified custom developed application as this is not the focus of the study area.

TLS methodology and Cipher suites

Due to scope of desired investigation, I will adopt the following 3 TLS implementation methodologies:

* TLS termination. Using this method, TLS traffic is terminated at the load balancer & not re-encrypted before being forward to the backend applications. Then after the load balancer the traffic will be in plain HTTP. This has been de-facto strategy to mitigate TLS performance impacts for many years in technical delivery, interestingly major cloud providers now provide this as standard when using proprietary cloud certificates. As the certificate private key is held by the cloud provider, it cannot be included in the creation of TLS secrets for inter cluster TLS connections. For TLS termination I have used the cloud provider proprietary certificate i.e. google managed certificate.
* TLS passthrough. This is also described as TLS end to end. Load balancer does not decrypt TLS traffic and it is forwarded directly to backend applications with full TLS. As not possible to use cloud provider proprietary certificate in this instance, I have adopted the generation of TLS certificate via acme and let’s encrypt, including the certificate chain and private key into the cluster and mounted to application via Kubernetes tls secret components.
* TLS bridging. This approach is less commonly used, where TLS is terminated at load balancer, decrypted then re-encrypted and forwarded to backend applications. In effect 2 separate TLS sessions are maintained; 1 between client and load balancer and in this case another between ingress controller and our monitored application. I have adopted this strategy as our quantum cipher algorithms are not supported via google managed or lets encrypt certificates to date, thus I am mounting the open quantum certificates into the ingress controller and reverse proxying the request to our OQS container outside of the cluster (on docker?? Or could actually do this within cluster??). Here performance overhead & complexity will be higher due to the second TLS session to be managed.

\*\* Nice to have diagram of TLS methodologies..

Cipher suites:

There are many cipher suites which can be tested for TLS 1.2 & TLS 1.3. **TODO: Add NIST recommendations.** In investigating TLS cipher suites, I started to investigate and rank the TLS cipher suites according to their strengths. However after analyzing the application and certificates, there are restrictions already in place on what cipher suites can be used by google, AWS, letsencrypt certificates as only a subset are supported. I have used both SSL Labs (reference) and nmap (reference) to identify the supported ciphers for cloud provider certificates listed below. Focusing only on (not weak) TLS1.2 ciphers and TLS1.3 ciphers we have the following ciphers to be tested (windows):

* TLS1.2
  + TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
  + TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
* TLS1.3
  + TLS\_AES\_128\_GCM\_SHA256
  + TLS\_AES\_256\_GCM\_SHA384
  + TLS\_CHACHA20\_POLY1305\_SHA256

Note: About AEAD (Authenticated Encryption with Associated Data) cipher suites – as SSL Labs caps the security at B grade if not using a cipher suite supporting forward secrecy. Could be my OS/Browser combination and what is supported from me (client). Check on both MACs to confirm grade.. On windows it is B grade because of this.

Chapter 1b: Technology Selection

1. Choosing the platform - factors came into play - AWS, GCP
   1. **Cost**: For this reason I have used the managed infrastruture of google cloud. I also investigated use of AWS, as I have more professional work experience on AWS. Both providers have similar offerings. For cost reasons, I have chosen to go with google cloud GKE (Google Kubernetes Engine) clusters, and google managed infrastructure services. For creation and maangement of clusters, I have developed cluster and application setup via Terraform, thus allowing me to “spin up” and “teardown” infrastructure as required, ensuring costs are managed optimally & also ensuring consistency in tests results from repeatable architecture setup.
   2. **DNS hosting**: As I chose to use google domains as my DNS hosting provider, GCP has good integration.
   3. **Dynamic load balancing**: GCP has quite dynamic load balancing, which I feel is more intuitive in setup I require with managed certificates and SSL passthrough load balancing.
2. Platform architecture
   1. GCP cluster with TLS Offloading
   2. Full end to end TLS (on traefik and application)
   3. Other investigations
      1. Lets encrypt doesn’t support google domains by DNS lookup - tried with Google cloud DNS
      2. Lets encrypt cert manager deployment
      3. TLS secrets for certificate inclusion
3. Application architecture for TLS
4. Monitoring tools
5. Testing tools
6. Platform services – DB, load balancer
7. DevOps – IaC – cost & stability in ensuring tests are reliable!

Chapter 2: SSL and Cryptographic Fundamentals

2.1 Overview of SSL/TLS

2.4 Cipher Suites

Chapter 3: SSL Cipher Suite Selection Criteria

3.1 Security Considerations

3.2 Performance Factors

3.3 Compatibility and Interoperability

Chapter 4: Comparative Analysis Methodology

1. Research Methodology
2. Test Environment and Tools
3. Monitoring
4. Test suite investigation
5. Evaluation Metrics

Chapter 5: Comparative Analysis of SSL Cipher Suites

5.1 Cipher Suite 1: Description, Strengths, and Weaknesses

5.2 Cipher Suite 2: Description, Strengths, and Weaknesses

5.3 ...

5.n: Comparative Analysis of Additional Cipher Suites

Chapter 6: Security Evaluation

6.1 Cryptographic Strengths

6.2 Vulnerability Analysis

6.3 Resistance to Attacks

Chapter 7: Performance Evaluation

7.1 Computational Overhead

7.2 Latency and Throughput

7.3 Resource Utilization

Chapter 8: Compatibility and Interoperability Analysis

8.1 Browser Support

8.2 Interoperability with Legacy Systems

8.3 Forward Secrecy and Key Exchange Methods

Chapter 9: Discussion and Findings

9.1 Comparative Analysis Summary

9.2 Key Findings and Observations

Chapter 10: Recommendations and Future Work

10.1 Recommended Cipher Suites for Different Use Cases

10.2 Areas for Further Research

10.3 Future Trends in SSL/TLS Security – *OQS support for cloud environments*

Chapter 11: Conclusion

11.1 Summary of Research

11.2 Contributions and Implications

11.3 Final Remarks

Appendices:

A. SSL Cipher Suite Configurations

B. Detailed Test Results

C. Glossary of Terms

D. List of Abbreviations

Ana suggested structure in overleaf:

II. Introduction

II-A. Background

II-B. CLoud environments

SHORT PLEASE

II-C. TLS and company

III. Methodology

III-A. Requirements

ABSTRACT: What do I want to measure/prove/analyze??

III-B. Environment design

III-C. Measurement and monitoring needs

III-D. Technology selection

Which monitoring tool? Libraries? Frameworks?

III-E. Experiment definition

IV. RESULTS

V. DIscussion

VI. Limitations

VII. Conclusions and further work

References:

TLS1.3

<https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>

<https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8447bis-04>

<https://datatracker.ietf.org/doc/rfc8446/>

<https://en.wikipedia.org/wiki/Transport_Layer_Security#Algorithms>

Prometheus setup:

<https://sysdig.com/blog/kubernetes-monitoring-prometheus/>

<https://traefik.io/blog/capture-traefik-metrics-for-apps-on-kubernetes-with-prometheus/>

Reference Terraform:

<https://registry.terraform.io/providers/hashicorp/google/latest/docs>

<https://registry.terraform.io/providers/hashicorp/kubernetes/latest/docs>

<https://registry.terraform.io/providers/hashicorp/helm/latest/docs>

Certificate references:

… Add these in ….

Cipher Suite selection

<https://www.ssllabs.com/ssltest/>

& nmap: nmap --script ssl-enum-ciphers -p 443 xyz.com