UMEÅ UNIVERSITY
Department of Computing Science **May 2025**
Master's Thesis

Spring 2025

Degree Project in Computing Science and Engineering, 30 ECTS

Evaluating Zero Trust Architecture Solutions in Cloud Native Environments: A Comparative Performance Study

Author
Ludvig Larsson

Examiner: Henrik Björklund
Academic Supervisor: Patrik Eklund
Advisors: Emil Palm and Mattias Sehlstedt
External Partner: Omegapoint AB
Master of Science Programme in Computing Science and Engineering, 300 ECTS

Abstract

Zero trust is a paradigm that has emerged from the need for better security in cloud native environments. The paradigm moves away from traditional parameter-based security to a more sophisticated method in which segmentation, user identities, observability, and request studies are utilized.

This thesis studies two enterprise solutions offered by *AWS* and *Microsoft Azure* which are compared by documentation and in practice. Moreover, the practical comparison focuses on the performance aspects of the solutions and how they change as *zero trust* principles are applied. In evaluating the solutions, four systems have been implemented. These consist of two solutions with *zero trust* and two without *zero trust* principles, where they are compared to show the latency overhead. The purpose of this thesis is to provide guidance as to when one of the two solutions can be preferred in terms of performance and overall *zero trust* coverage.

Comparison by documentation and performance showed a static increase in latency for requests made with *zero trust* principles applied. However, the increase in latency is hard to define and compare, as the providers do not offer the ability to specify how the services are to be hosted below region and availability zone level. Thus, making the performance obsolete in the choice of the service provider. Instead, the work showed the importance domain knowledge of a service provider as a major factor in the decision.

Keywords

Zero Trust, Zero Trust Architecture, Performance, Latency, Service Provider, AWS, Microsoft Azure

Acknowledgements

I want to start by expressing my gratitude to examiner Prof. Henrik Björklund and my academic supervisor Prof. Patrik Eklund for their support, feedback and guidance throughout the research of the project.

In addition, I want to thank Omegapoint AB and my advisors Emil Palm and Mattias Sehlstedt. Being able to work at the Omegapoint office has given me insight, motivation, and valuable experience that have contributed to my personal and professional growth. They have also provided me with valuable support, discussions, and insights on the topic that I am grateful for.

Lastly, I want to thank my brother for helping me find an interesting topic for my research. Your help has been valuable to me even before the research started and I am thankful for it.

Ludvig Larsson
Umeå, May 2025

Contents

List of Figures

List of Tables

Listings

1 Introduction

The number of attacks on enterprise systems increases at an alarming rate every year [14]. These attacks are not limited to larger corporations, but also include smaller businesses and individuals. Therefore, it is important to secure confidential resources regardless of the size of a system.

Traditionally, cloud networks were deemed secure by securing the outer perimeter of the network. With the discovery of *Operation Aurora* in 2009 [1, 11], many security experts in cloud environments realized that the older security solutions were no longer viable. Thus, some changes had to be made. Chenquan Gan et al. state that malware is commonly spread in the cloud by infecting images. An image is a set of instructions that are used to run instances of an application. Originally, it was common for virtual machines and nodes in the cloud to share images and image repositories. Therefore, infecting virtual machine image repositories has been an effective way of spreading malware. In addition, in an analysis of *Amazon* images, it was found that 98% of the Windows and 58% of the Linux images included severe vulnerabilities that can be exploited by malware [14]. Multiple proposals have been made on how vulnerabilities such as these can be mitigated, where *zero trust* (*ZT*) is one of the leading solutions.

Trust as security has been a hot topic in the world of cybersecurity in cloud environments for the past decade. The term *Zero Trust Architecture* (*ZTA*) was first discussed by researchers at *NIST* and *NCCOE* in 2020 [26]. However, the concept of *zero trust* has been around in cybersecurity long before the first mention of *ZTA* [21].

Marsh defined the concept of *computational trust* in computer networks as part of his doctoral thesis [21]. Trust is something we humans use every day in our lives, but how can it be defined? And how can it be applied to networks? Those are the questions Marsh discusses, where he also introduced the early concept of *zero trust*. *Zero trust* introduced by Marsh is the concept of two or more agents in a network that have never or just met. At the first meeting, agents define trust by calculating a trust score. This score aims to represent whether an agent can be trusted or not. In first contact, the trust score of an agent is set to 0, where it increases or decreases based on subsequent communication. Marsh recognizes that his work on *zero trust* is shallow and is an avenue for future work.

Today, cloud providers such as *Amazon* and *Microsoft* offer services and guides to adopt *ZTA* in cloud environments. As discussed by Ferretti et al. [13], it is difficult for solutions to be truly in line with the principles of *ZT*. Therefore, having providers who help with the services needed to adopt *ZTA* is important for many businesses to protect their resources. However, *ZTA* comes at the cost of latency [24], where each service must be individually evaluated to adopt the security measures of *zero trust*.

This thesis aims to expand on the discussion of Ferretti [13] and Rodigare [24] about *zero trust* performance in cloud environments. Ferretti exclaims that there is a performance overhead of *ZT* while Rodigare proves it by testing latency between two clusters. This thesis will continue the work by comparing the performance overhead between two commercially available solutions in a single cluster. In addition, this work will try to compare the two solutions in the documentation theory. This comparison will be done by designing *ZTA* approaches to a simple generic system. With the resulting values, it is tried to argue where one of the solutions can be preferable.

1.1 Research Objectives

In this thesis, the topic of *zero trust* will be explored in the world of *Amazon AWS* and *Microsoft Azure*. Furthermore, the solutions of *AWS* and *Microsoft Azure* will be compared and discussed in detail in both theory and practice. The practical part of the comparison will focus mainly on performance, while the theoretical comparison will take a deep dive into the services that are offered. The following are the research objectives that will be focused throughout this work.

RQ1 What are some differences in *zero trust* solutions offered by *Amazon AWS* and *Microsoft Azure*?

RQ2 How does performance differ between solutions and the case without *zero trust*?

RQ3 In what situations can one of the solutions be preferred over the other?

1.2 Related Work

Operation Aurora of 2009 includes a series of malware that had larger Western IT companies under fire. The malware was designed to obtain information and source code in public services. This was done using the fictive trust of networks in systems, where the malware acted as any other service on an infiltrated network. In response to *Operation Aurora*, *Google* started exploring new security architectures to combat the new approach to malware. This work resulted in the creation of *BeyondCorp*. *BeyondCorp* presented a solution to the internal security problem by defining an internal network as untrusted and always compromised. Therefore, every request must be verified by a policy enforcer agent. Thus, removing the possibility of accessing services based on old trusts or tokens of authentication [30].

One of the first concepts commercially available for the problem of internal security was *Software-Defined Perimeter* (*SDP*) proposed by *The Cloud Security Alliance* (*CSA*). *SDP* aims to specify the components, interactions, and protocols needed to secure components within networks. The first version of the specification was published in 2014 and contained a few challenges with regard to overhead and vulnerability. However, *CSA* published a second version of the specification in 2022 to combat these challenges. Furthermore, in the years between the first and second versions of the *SDP* specifications, multiple enterprises have successfully applied the specifications for *zero trust* [16].

In 2020, the *National Institute of Standards and Technology* (*NIST*) published an article on zero trust. The article presented a paradigm for the *zero trust architecture* in which the defining of important topics was the focus. *NIST* introduced the topic by defining critical points. Three of them are [26]:

- **All communication is secured regardless of network location.** – Having access to the network does not imply access to resources in the network. All internal communication is handled as untrusted and is applied to some trust policy. Thus, just because the request comes from inside the network does not mean that it will gain access to a resource.
- **All resource authentication and authorization are dynamic and strictly enforced before access is allowed.** – Static authentication of resources can introduce new problems in which old authentication tokens can be used for access. Therefore, tokens are only usable once for a specific session. Thus, removing the vulnerability of malware using old authentication tokens.
- **No resource is inherently trusted.** – As requests are not trusted, neither are resources. Before accessing a resource, the resource itself is evaluated by a policy enforcement service.

NIST's publication on *ZTA* has had a large role in the topic. The publication can be seen as the standard for *ZT*, defining what is needed in a solution to comply with *ZT* [26].

With the publication of *NIST*, the US government started to enforce the policies of *ZTA*. In 2022, the *Office of Management and Budget* published a directive to move government agencies to *ZT* principles [26, 20]. Moreover, define the importance of *ZTA*.

Some publications examine the individual performance of solutions. Rodigare [24] gave a good overview of the performance cost between two cloud networks. In the survey, both clouds used *ZT* policies to enforce authorization. Thus, Rodigare was able to measure the latency of outer-cluster communications.

Ferretti discussed the performance of *ZTA* in the publication about *Survivable Zero Trust for Cloud Computing Environments* [13]. The main focus of the article is on the survivability of *ZTA* solutions proposed in different articles. In addition, Ferretti proposes the first survivable *ZTA* that satisfies the core principles of *ZT*. Adopting true *ZT* in a cloud environment comes at the cost of increased latency. Therefore, Ferretti chose not to protect request confidentiality due to performance restrictions. However, this is replaced by adopting universally verifiable multiparty computation schemas that allow verification based on a set of secret inputs the user can define. In concluding the report, Ferretti expressed that survivable single-sign-on systems appeared to be the main cause of latencies in the authentication process. Here, the typical response time for authentication is two seconds [13].

In 2023, the *National Security Agency* (*NSA*) devised seven pillars of *ZT* that were to define all aspects of cybersecurity. These pillars were to be seen as guidelines for areas where security should be adopted and refined. The defined pillars were user, device, application, workload, data, network & environment, automation & orchestration, and visibility & analytics. The second of the pillars, the device pillar, is said to be the most important. This is because every aspect of the cloud and users are hosted on some device, whether it is a server or a personal computer. Moreover, securing connected devices is of the most importance, as it could send unauthorized requests if compromised [2].

1.3 Thesis Outline

The thesis will be divided into 8 sections where they are to discuss:

1. **Introduction** – Introduces the topic of *zero trust* where previous works are presented to motivate this project. In addition, it introduces the research questions that it aims to answer.
2. **Background** – Presents background information that is needed in comparisons and discussion throughout the work.
3. **Comparing Solutions** – Compares the solutions of *AWS* and *Microsoft Azure* in theory without implementation.
4. **Case Study** – Presents the system and surrounding information that is used during the practical evaluation.
5. **Results** – Presents the results of the experiments.
6. **Discussion** – Discusses Section 3 and the results presented in Section 5.
7. **Conclusion** – Concludes the report and tries to answer the research questions presented in Section 1.1.
8. **Future Work** – Presents some areas that can be explored in the future on the topic of *zero trust*.

2 Background

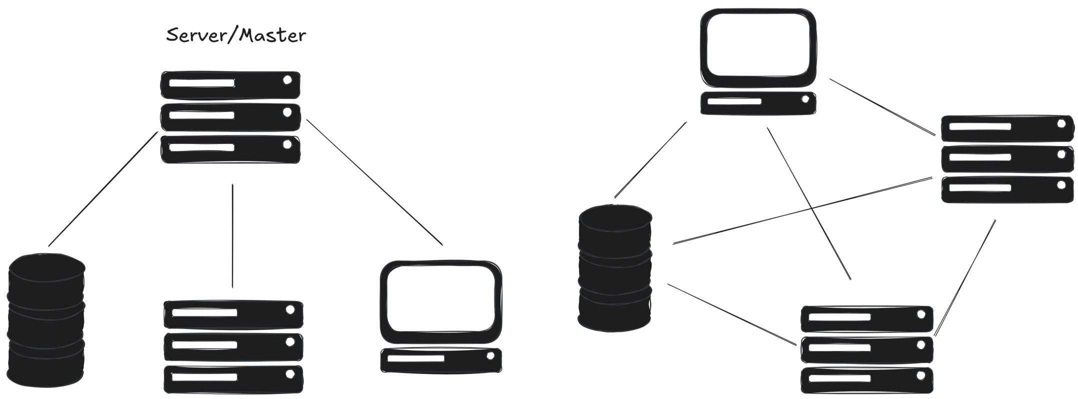
In order to have a better understanding of the topic of *ZT* in cloud environments, this section aims to introduce the needed keynotes. Topics such as computer networks, cloud computing, cloud environments, trust, *zero trust* and cloud providers will be introduced and discussed.

2.1 Computer Networks

A computer network is a set of computers or other devices that are connected. A defining part of this connection is that the devices may communicate data. Furthermore, this communication allows devices to help or work on the same task. Today, systems that are computed on multiple devices at once are called distributed systems, which would not be possible without computer networks [19].

For the network to be able to construct communications, there must be a connection between the systems and a link by software. The software needed is a combination of network and transmission protocols. The network protocols are there to specify the set of devices included in the network. In addition, transmission protocols are used to specify the format for sending and receiving data [19].

Furthermore, there are two central models for computer networks. In both types, computers are connected to a network to communicate. The first model is the centralized computer network where there is a single computer/server that handles all requests and work. The other computers act on the master decisions. The second model is the distributed model. This model does not have a single point that manages all other computers. Instead, they all handle it together sending data to make decisions. This approach can be difficult to design, but in turn is more scalable and dynamic [19]. Figures 1a and Figure 1b, respectively, contain illustrations of the centralized and distributed network models.



(a) Centralized computer network. (b) Distributed computer network.

Figure 1: Illustrations of computer networks.

2.1.1 Cloud Computing

Cloud computing is taking the functionality of computer networks presented in Section 2.1 and distributing it over the Internet. Thus, instead of having a local network, nodes are not limited by location and can be placed to some extent all over the world [23].

In cloud computing, there is no limitation to the geographical location of users or resources. This in turn introduces issues with latency where transferring data is affected by distance. Having different locations, or even changing locations of nodes in a network, introduces challenges with latency and security. Latency problems are introduced with geographical location and transfer through different internet providers. Thus, increasing the possibility of problems and bottlenecks in transfer rates. Furthermore, securing data in and between systems becomes a greater challenge when nodes are distributed. Especially when nodes may be running in public data centres. Thus, it introduces the need to secure multiple locations from physical hardware theft [23]. Therefore, it is important that security is not an afterthought, but is considered in the early stages of implementation [15].

Today, some providers offer public cloud solutions. The two largest cloud providers are *Amazon AWS* and *Microsoft Azure* [28]. By purchasing services from one of the providers, you will have access to a set amount of resources in specified regions of the world. Thus, developers do not need to acquire hardware that they have to configure. Instead, developers can focus on developing services where the provider handles the rest [15].

2.1.2 Cloud Native

Cloud native is a term that emerged with the wide spread of distributed systems and cloud computing. The term is used to encapsulate applications that have been designed and developed to run on scale in the cloud. There are five main properties of cloud native applications that are discussed and a core features of the application [15]:

1. **Scale** – Cloud native applications often operate on a global scale. Thus, it is important to plan for this when designing systems. Having just a few servers located in a specific location will probably not be enough to provide a service for people all over the world. Therefore, having servers in different parts of the world that work together is important to service users.

2. **Scaling** – You cannot know how many users will be active in the network at one time. Therefore, it is important to have the ability to quickly and efficiently scale up or down the amount of resources that can service users. This in turn increases availability, while it can also save on costs when there are downtimes in the number of users.

3. **Failure is constant** – As servers can be placed all over the world. It can not be determined beforehand

on what hardware it will run on. Thus, it is important to plan and prepare for failure in which resources can hop in and out of the network on startup and failure.

4. **Uptime** – Cloud native networks are designed to always be accessible to a user. Moreover, there will always be some resource that is able to service a request and return the desired result.
5. **Security** – Security cannot be an afterthought when it comes to cloud native. Since developers cannot predict where the services will be hosted, the resources need to be secure to not allow access to sensitive data.

Every day we use cloud native applications without thinking about it. Larger networks such as Netflix, Microsoft Office 365, Amazon AWS, Facebook, Twitter, and many more applications are running on cloud native applications. Thus, it shows how important cloud native applications are for our day-to-day lives [15].

2.1.3 Types of Cloud

Cloud solutions are not locked in their configurations. Depending on the usage, clouds can be specified to suit different solutions where usage and security differ. The most common cloud configurations are [18]:

- **Public cloud** – The public cloud is the most common type of cloud where services are offered to the public. This is the category in which some larger providers such as *AWS*, *Microsoft Azure*, and *Google Cloud* are included. In this model, providers can connect numerous data centres and servers to give their customers access to scale their systems [18].
- **Private cloud** – The private cloud is the opposite of the public cloud, where services are mainly accessible by a single individual or organization. A large selling point of the private cloud is the security in which only a single unit has access to the system. This in turn ensures that no other service has access to sensitive data that can be used on the system.
- **Community cloud** – The community cloud is a model where multiple services require the same infrastructure to run. Using the same infrastructure, enterprises can save operational costs by sharing the cost with multiple different companies.
- **Hybrid cloud** – The hybrid cloud is a combination of public and private cloud. Here, both private and public servers are connected, where the machines serve different purposes. In case only a small amount of operations needs to be done with sensitive data, these operations can then be done on the private cloud, while the less sensitive information can be done on the public servers.
- **Virtual private cloud (VPC)** – A virtual private cloud is a private cloud that is hosted inside a public cloud. This is possible due to the creation of virtual networks that gain exclusive access to parts of the cloud. One solution to set this up is to create a subnet of IP addresses and ports to limit access.
- **Multi-cloud** – A multi-cloud is a network of two or more cloud solutions that are linked together. These cloud solutions do not have to be hosted by the same provider. Thus, allowing the end user to utilize the best of all worlds. In addition, multi-cloud solutions are not limited to only public or private clouds, and can therefore mix them in the same way as hybrid clouds have private and public nodes.

2.2 Traditional Security Models

One of the most important aspects of any system is the availability of the service it provides. In some cases, the service provided is sensitive, and only a handful of users can access it. In these cases, it is important to regulate the access in some way. Today, there are more sophisticated approaches to the security problem. However, that has not always been the case, where traditional solutions were deemed good with regard to the old standards.

2.2.1 Perimiter-Based Security Model

Perimeter-Based Security Model is built on the theory of compartmentalization. Each compartment, or set, consists of sets and subsets of information to define access to services and data. The sets are constructed to eliminate the conflict of interest between the subsets. As an example, each set and subset can be defined as companies with conflicts of interest. As an example of a *Perimiter-Based Security Model*, there is *The Chinese Wall Security Model* where access is granted to the object o if and only if it meets either of the two rules [4]:

- The object o is present in the same data set as the location from which the request originates.
- The object o is located in a different data set that is not in conflict with the origin of the request.

The concept of *Perimiter-Based Security Models* has been used for distributed systems. The main difference in terminology between the two is the content of the sets that can be defined as clusters. In distributed systems, the contents of each cluster are servers, nodes, services, or data. Each cluster defines the services within the access. Without this definition, requests could cause havoc and jump between all pathways within a cluster [12]. Figure 2 illustrates how groups and sets can be organized to limit access for *The Chinese Wall Security Model*.

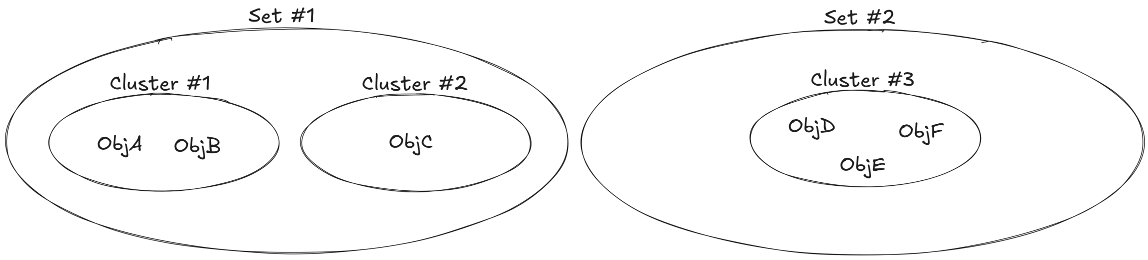


Figure 2: Illustration of compartmentalization of *The Chinese Wall Security Model*.

2.2.2 Demilitarized Zone

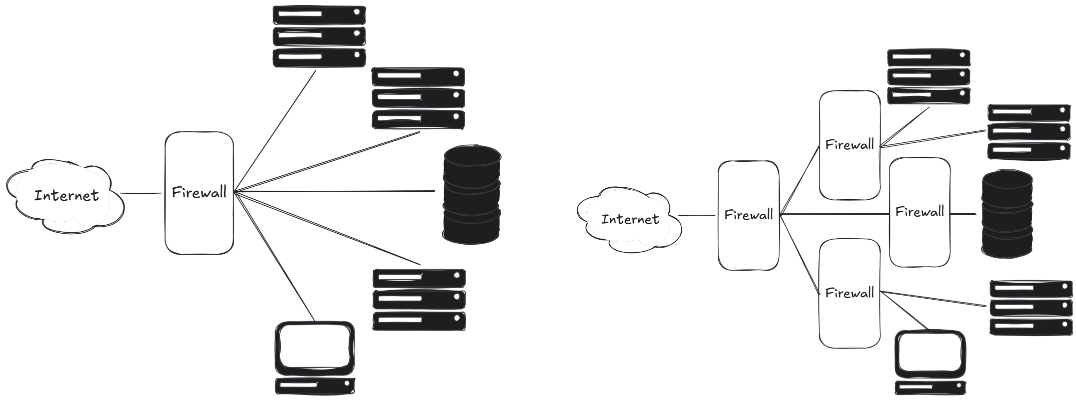
The *Demilitarized Zone* (*DMZ*) is a security model based on rough segmentation. In short, the network is split into two parts; the normal network, and the *DMZ*. The *DMZ* serves as a service in which other partners can obtain information. Since there is a clear split between the two networks, the *DMZ* can be specified with less security but with higher performance. The *DMZ* can be divided into four subpatterns [9].

- **Proper location of devices and connections** – The *DMZ* is to be located as close to the perimeter as possible. This is because it should pass through as few devices on the local network as possible before it reaches the outside world. The connections need to be configured to prioritize sending of data and traffic. This in turn reduces the complexity of the specified *DMZ* [9].
- **Dedicated systems** – The *DMZ* is to run on a dedicated system. The purpose of systems is to transfer data and nothing else [9].
- **Performance measurements** – Since the system is to be configured to prioritize performance, it is also important to measure performance. Performance can suffer from two types of problems; hard and Soft issues. Hard issues occur when an error causes complete failure of a transfer. This in turn results in a new connection that needs to be established to resend all information. Soft issues are much simpler, such as packet loss, where the specific packet can be resent to the consumer [9].
- **Security policies** – As the *DMZ* is outside the network, it is also outside the firewall. Thus, it requires its own security policies. These networking policies can then be configured to allow for high performance. This is because most of the tools today are not designed for high performance. Thus, the tools can be changed to increase performance [9].

2.2.3 Network Segmentation

Network Segmentation is a network strategy in which the network is layered behind firewalls. The network is segmented so that devices in the same segment have the same security policies. Thus, devices on the same network have different security policies depending on the location in the network [22].

In traditional network design, there is a single firewall between the network and the Internet. That is not the case for *network segmentation* where there can be multiple firewalls on the same network. Figure 3a shows the traditional network model, while Figure 3b shows the same network with segments [22].



(a) Traditional network with only a single firewall.

(b) Segmented network with multiple layers of firewalls.

Figure 3: Comparison between a traditional and a segmented network.

2.2.4 Identity-Centric Model

The next step for network security was the *identity-centric model*. The *identity-centric model* revolves around the identities of users and devices that interact with systems regardless of the present firewalls. Outside of computers, all humans have their own identities, so how can computers utilize the same system for authorization of requests? Managing the identity of users is done today in an open and standardized way. Tools as *OpenID*[1] and the archived project *Higgins*[2] are used to identity users for authentication. Thus, even if users have access to a network containing services, it still needs to verify their access by sending in their user and device identities for evaluation.

2.3 Defining Trust

Trust is a central part of our normal lives. It has been argued that it would be hard for humans to grasp the complexities of the world without trust. Every day, trust is exerted in different forms. For example, trust is used every time someone drives on the road. Although there are rules on how to behave in traffic, the driver can never know if the other drivers intend to follow them. Thus, some form of trust needs to be established towards the other drivers [21].

The concept of trust is also applicable in the world of computer science, where it is an important topic for security [21]. In this thesis, computational trust will be used to argue why security is needed in cloud networks. Marsh introduces the topic of trust in [21] as dynamic, whereas static trust would introduce problems.

- **Full trust** – Always trusting others leads to vulnerabilities. Blind trust of everyone introduces the problem of accepting requests that might include malware or malicious data [21].
- **No trust** – Without trust, the security problem becomes almost trivial, in which nothing is accepted from another machine. Therefore, the vulnerabilities introduced by trusting malicious data are removed [21]. However, it removes the functionality of coordination between systems, where each task needs to be completed on the same machine that it originates from.

As expected, neither of the static options is good in terms of security and functionality. Therefore, some other approach to dynamic trust must be applied to value both security and functionality. Marsh introduced in [21] the topic of *zero trust*. At first glance, *zero trust* and *no trust* could be interpreted as the same model. However,

[1] `https://openid.net/`
[2] `https://projects.eclipse.org/projects/technology.higgins`

that is not the case where *zero trust* is a dynamic approach to security. Initially, there is no trust for devices or users before their trust have been evaluated. Once computed, only the minimal amount of privileges needed to complete the request is provided.

The definition of trust needs to be defined differently between systems in the same way that humans trust others differently [21]. Thus, *ZT* approaches allow policies to be configured. These policies are then used in the decision-making about granting access to resources. The topic of policies will be discussed further in Section 2.4.4.

2.4 Zero Trust

Zero trust is a paradigm for restricting access to both internal and external resources. In this case, resources are all services contained in an enterprise. This restriction is done by the property of "never trust, always verify". Moreover, when a request for a resource is granted access, it only grants the minimum privileges required to fulfil the request. Thus, restrict the set of resources to which the request has access [26].

Zero trust is not a product or a service that can be applied with the click of a button. It is a security approach following a set of security principles [17].

- **Verify explicitly** – Always authorize based on all available data points in the authentication process.
- **Least privilege access** – Do not supply more access than required. Adopt just-enough-access and just-in-time policies and data protection policies.
- **Assume breach** – Minimize access between services to minimize damage in breach. Always use verified end-to-end encryption.

2.4.1 Components

Providing access to requests is done through a *Policy Decision Point* (*PDP*) and *Policy Enforcement Point* (*PEP*). The *PEP* is the service that evaluates and authenticates the access to devices and users. In addition, requests are deemed to be untrusted before passing through the *PDP* and are enforced by the *PEP*. A simple overview of this architecture can be seen in Figure 4 [26].

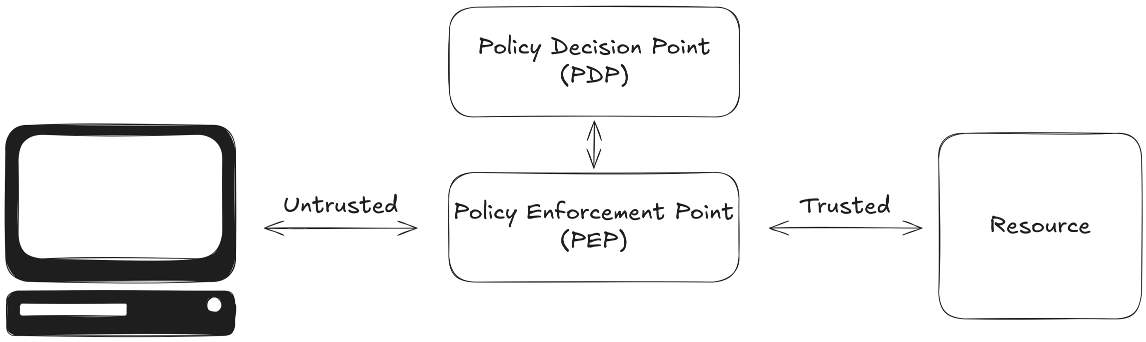


Figure 4: Overview of zero trust access.

To enforce the principle of only assigning access to resources that are needed, the trusted set is always limited. Therefore, each request will have a different set of trusted resources that it can access. Once it has passed the *PDP* and *PEP* the set of trusted resources cannot be changed. Thus, if a resource is needed and the request does not have access to it, it cannot be completed.

The architecture of *ZTA* has two layers, the control plane and the data plane. These layers are seen as trusted and untrusted, respectively. Looking back at Figure 4, everything that is not the *PDP* is located in the data plane, while the *PDP* reside in the control plane, Figure 5. This structure is important since the *PDP* requires that it runs in a trusted environment to make decisions on granting access. Without it, the *PDP* would not have access to information that is later needed to make decisions about granting access [13, 17].

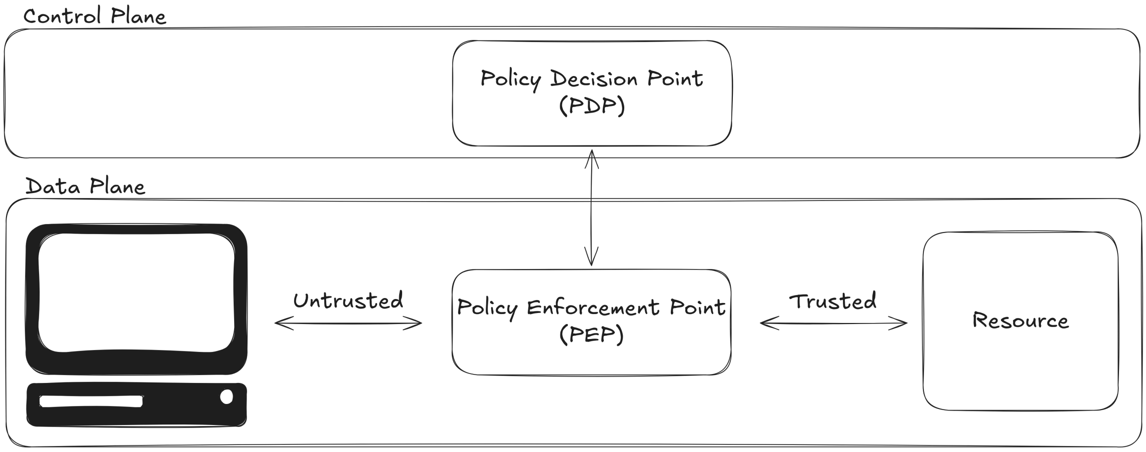


Figure 5: Zero trust architecture divided into control and data plane.

Inspecting the *PDP*, it can be broken up into two components; the *Policy Engine* (*PE*) and the *Policy Administrator* (*PA*). The *PE* is responsible for the decision to grant access to a resource. The decision is made taking input from defined policies and external sources, Section 2.4.4, as input for a trust algorithm, Section 2.4.2. The *PE* works closely with the *PA* as the *PA* is the service that establishes and destroys connections. The connections are entirely based on the decision of *PE*, where it does not make any decisions on its own [26]. Figure 6 visualizes the *PDP*.

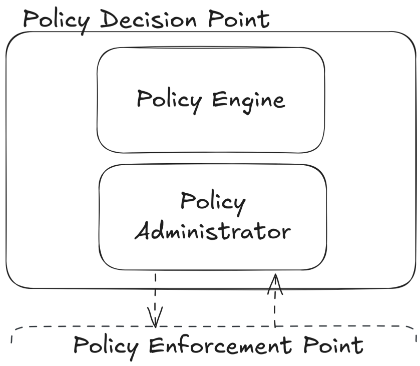


Figure 6: Main components of the *Policy Decision Point*.

In addition to the structure mentioned above, there are a few requirements for *ZTA*. The first and trivial requirement is an Internet connection. Without it, it would not be difficult for infiltrators to access and receive information from services. Alongside an Internet connection, it is required that there is full visibility of the network traffic. Furthermore, there is no access to the resources without passing through the *PEP*. There are a few more requirements that [26] raises which are linked to structure rather than access policies.

2.4.2 Trust Algorithm

As discussed in Section 2.3, trust plays a large role in *ZT*. Without trust, devices and users are not able to gain access to resources. But how is trust decided?

The process of deciding whether a user or device can be trusted is performed by some trust algorithm. The trust algorithm and content vary between solutions to better adopt system policies. However, it can be said that the algorithm is dynamic in accessing relevant information to make decisions. There is no single place to get all the information needed to make these decisions. Thus, the trust algorithms fetch information from all over the cluster. It could be from the logs of a resource or from the secure identity managers who oversee the system. In either way, all relevant information is then inserted into the trust algorithm to decide whether access is granted or not [26].

Policies, introduced in Section 2.4.4, play an important role in making a decision about access. The defined policies are inserted into the trust algorithm and used to grant access along with network metrics. Thus, the decision to grant access is made with many parts of the system working together [29]. Figure 7 shows how a set containing a request and information could lead to a trusted or untrusted result.

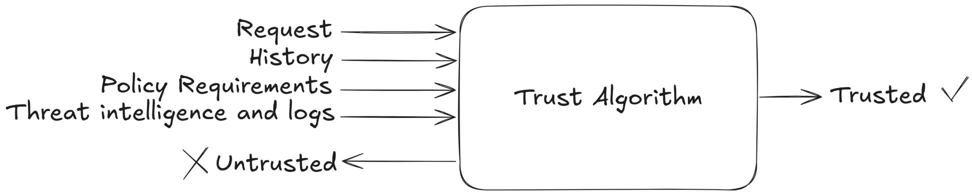


Figure 7: Parameters to the trust algorithm.

2.4.3 Network Restrictions

Limiting the access of incoming requests is not enough to uphold the policies of *ZTA*. It also requires that all requests between resources be monitored and managed based on policies. As mentioned above, some of the policies that might exist in networks will be presented in Section 2.4.4. However, how are requests monitored?

The monitoring of requests is done on the control plane of the network. This is done through services that are apart or have access to the *PDP* and checks for policy compliance. In the cases where requests are policy defiance, the request is rejected and returned to the sender.

2.4.4 Enterprise Policies

Defining a generic set of policies that can be used for all enterprises. Each solution has its policies used to grant access to users and devices. Therefore, this section will introduce a set of generic policies that could be helpful depending on the service.

- **Industry compliance system** – Rules and laws in different fields are mandatory. Therefore, compliance with them is important to avoid legal trouble. There are some companies where this may be of great importance. For example, healthcare and financial systems rely on personal information. Thus, increasing the importance of data security [26, 29].
- **Network and system activity logs –** Logging information about the network and resource activity can map access that has not been granted. Therefore, it is important to not only guard against, but also actively destroy connections if they are unlawful or have been timed out [26].
- **Data access policies –** What are the enterprise's policies for accessing data? Data access policies answer this by specifying the attributes and rules used to grant access to resources. It is usually the first policy checked to see if a device has access to the requested resource [26].
- **ID management system** – Manages the creation, storage, and removal of accounts and records. This system often uses other services to store information, where user information is stored [26].
- **Firewall provisioning** – Module that maintains a topology of services and zones within a network, including firewall information and rules [29].
- **Trade-off between operational need and security risk –** In some cases, denying access to resources could lead to increased response times or denial of service. However, could it be worth bypassing some authentication to serve users? This could be viable in cases where information is obtained that does not need to be secured [29].

As can be seen in the generic categories mentioned above, the policies very often cross industry standards and trade-offs. Thus, it is up to the development team to specify which policies are relevant to the service. To give an example, a banking system is a service where security is of great importance. Therefore, all the above-mentioned topics are used, where the operational trade-off may be left to discussion. However, a system designed to provide a public service open to all does not require policies for ID management and data access policies. Instead, the importance lies in the operational performance where security lies in securing the uptime and access of the machines the services are hosted on.

2.4.5 Observability

Always assuming a breach requires the network to be on constant lookout for malicious activities. Thus, it requires constant monitoring of requests and services inside the network. To solve this, a new layer is created in the control stack called the *infrastructure layer*. The *infrastructure layer* allows the developer to host services that manage the overall system where the services are running. Normally, the *infrastructure layer* is described as the management of hardware and software that creates and manages servers in a cluster.[3] However, it also includes services such as metrics, logging, and other management applications that are running. These services usually have access to all the services that are running on a computer. Thus, a layer can be created that can grant and manage access based on all resource utility. The logs and metrics produced in the service layer are responsible for providing the resulting values to the authentication and trust process described in Section 2.4.2.

2.5 Service Providers

In the context of the cloud, service providers offer computational capabilities and servers for businesses and users. This makes it possible for individuals, companies, or organizations on any scale to pay a sum to gain access to hardware to host their services. Service providers come on all scales, where the two largest providers today are *Amazon AWS* and *Microsoft Azure*. In general, providers offer four different models of services, *On-Premisis*, *IaaS*, *PaaS*, and *SaaS*.[4] In the following list, the four models are described in detail, where Table 1 shows what customers can configure.

- **On-premisis** – *On-premisis* is a service that allows the customer to host their own hardware. The service provider delivers space, electricity, and an internet connection.
- **Infrastructure as a Service (IaaS)** – *IaaS* is the service of providing a virtual environment where the customer has full control. Thus, the customer handles everything from the operating system, up to the running applications.
- **Platform as a Service (PaaS)** – For *PaaS*, the provider provides everything in terms of virtual machines, operating systems and runtime configurations. Thus, the only thing the customer needs to handle is the applications and data management.
- **Software as a Service (SaaS)** – *SaaS* is the service of buying access to some service that is already running. Furthermore, the customer only interacts with the service and does not interact with the underlying hosting machine.

The decision in choosing a model to use depends highly on the service that is to be hosted. In general, it is easier for the developer to not have unnecessary control over the virtual machine where the management is left to the provider. Thus, hosting a single application service would most likely use a *PaaS*, while services that require more system control on an *IaaS*. To give a representation of what each of the four models provides, Table 1 is provided, where the bold text represents the parts that are managed by the service provider.

2.5.1 Amazon Web Services

In the year 2023, *Amazon Web Services* (*AWS*) was the largest provider of cloud services in terms of *IaaS* and *PaaS*. In total, *AWS* provided 47.8% of all outsourced services [28]. *AWS* offers some tools and services

[3] `https://aws.amazon.com/compliance/data-center/infrastructure-layer/`
[4] `https://cloud.google.com/learn/paas-vs-iaas-vs-saas`

Table 1: The four deployment methods where bold text indicates the parts that are managed by the service provider.

On-Premises	**IaaS**	**PaaS**	**SaaS**
Applications	Applications	Applications	**Applications**
Data	Data	Data	**Data**
Runtime	Runtime	**Runtime**	**Runtime**
Middleware	Middleware	**Middleware**	**Middleware**
O/S	O/S	**O/S**	**O/S**
Virtualization	**Virtualization**	**Virtualization**	**Virtualization**
Servers	**Servers**	**Servers**	**Servers**
Storage	**Storage**	**Storage**	**Storage**
Networking	**Networking**	**Networking**	**Networking**

that allow enterprises to adopt ZTA [3]. Some of the services they offer consist of virtual machines and different database configurations. These services are respectively hosted on *Elastic Compute Cloud*[5] (*EC2*) and *Relational Database Service*[6] (*RDS*).

EC2 is a service provided by *AWS* that offers the user to host virtual machines with different operating systems. Looking back at Section 2.5, the *EC2* virtual machines are inserted into the hosting model of *IaaS* where the developer gains access to the virtual environment.

RDS is a service that allows users to host databases in an enclosed environment. In creation, the user only has to specify the resources that are to be assigned to the resource and database version where *AWS* solves the rest. Therefore, an *RDS* instance can be seen as a *PaaS* service model.

AWS also offers a selection of *SaaS* services that are of use in setting up *ZT* compliance. Some of these are:

- **Verified Access**[7] – Is Amazon's solution for verified access. By design, it allows authenticated agents to gain access to resources without the need for a *VPN*. In addition, it functions as a single tool that can be used to access resources by actors. The service can be used in multiple different services and can standardize the security of applications. The decision to replace traditional *VPN* with a tool as `Verified Access` is made to easier enforce access policies to services. *Verified Access* has the ability to allow access only to resources to which a user has access privileges. Thus, becoming a dynamic solution instead of having a single point of entry.
- **VPC Lattice**[8] – This is Amazon's solution to enable secure communication between services. This is done by setting up virtual connections between resources. Once the connections are established, all other forms of connection to the network can be removed. Thus, creating a single point of entry for resources that is secured by *AWS VPC Lattice*. To comply with *ZTA*, all requests sent over the connections are verified to ensure that they are authorized to continue.
- **Verified Permissions**[9] – Amazon's authorization service that utilizes the *Cedar policy* language. The service runs standalone from the production code and can therefore boost deployment times for new applications where only a fraction of the authenticating code needs to be written. In addition, the service utilizes logs from *AWS Verified Access*, *Amazon VPC Lattice*, and any other logging security service.
- **GuardDuty**[10] – *GuardDuty* is an *Artificial Intelligence* (*AI*) tool that has been trained on historically malicious data to analyse networks. It works by gathering logs and keeping track of requests that are

[5] `https://aws.amazon.com/ec2/`
[6] `https://aws.amazon.com/rds/`
[7] `https://aws.amazon.com/verified-access/`
[8] `https://aws.amazon.com/vpc/lattice/`
[9] `https://aws.amazon.com/verified-permissions/`
[10] `https://aws.amazon.com/guardduty/`

made on the network.

- **Identity and Access Management (IAM)**[11] – *IAM* is a tool for applying and managing user authentication. The tool offers routines for authentications where the *Cedar* policy language is used to customize policies.
- **API Gateway**[12] – *API Gateway* is a service provided by *Amazon* that allows the publication of secured *APIs* where authentication has been abstracted to the *IAM*. Thus, allowing the *API* to be free of the authentication process.
- **CloudWatch**[13] – *CloudWatch* is *Amazons* tool for centralized logging and metrics. It is a standalone service that gets information about systems through built-in features in applications or daemons that run in the same environment. Services as *RDS* have it built in to automatically send information, while *EC2* requires a daemon to run on the virtual machine for most metrics to be sent.

Using a combination of services mentioned above allows enterprises in *AWS* to be secure and *ZTA* compliant [3]. The process of setting up *ZTA* on *AWS* will further be discussed in Section 4.3.

2.5.2 Microsoft Azure

Microsoft Azure is the second largest provider, with a market share of 37.4% in the year 2023 [28]. They adapted the concepts of *ZT* early in the as the concepts were defined. This early adoption has allowed them to iterate and improve on the same problems. Adapting *ZT* in *Microsoft Azure* have been done using a set of tools and technologies that *Microsoft* offer. These tools are created to solve the security of one set of devices from the seven pillars of *zero trust*, presented in Section 1.2 [7].

Virtual machines are widely used in *Microsoft Azure* where they can be used to host anything that is desired. Moreover, they are not locked to any operating system, but can be configured to use any that is provided. Thus, giving the user access to a *IaaS* environment.

As with *AWS*, *Microsoft Azure* provides the same ability to service the user with databases that can easily be set up. The only thing that is needed is the type of database and the resources that it is assigned. With this information, *Microsoft Azure* sets up an instance of a database that can be configured to be available outside the network or only inside it, depending on the specified policies.

In addition, *Microsoft Azure* offers a wide selection of *SaaS* services that are used in setting up *ZT* compliance. Some of these are:

- **Network Security**[14] – Secures networks by segmentation. This division is done to minimize the access of compromised requests. Other tools are used to remove the possibility of gaining access to resources. However, if someone gains access, it does not spread to other resources [6]. The *Azure Network Security* manages the network & environment pillar where it manages and secures the network.
- **Entra Private Access**[15] – *Azure Entra Private Access* (*AEPA*) is *Microsoft's* solution for retiring *VPNs*. Using *AEPA*, the user is able to connect to *Microsoft Azure* services directly using *Azure Entra ID* as authentication in a secure environment.
- **Entra ID**[16] – *Entra ID* is the identity manager in *Microsoft Azure*. It allows the administration of users in the form of multifactor authentication and conditional access [5]. *Entra ID* manages the first pillar of the seven, the users.
- **Security Center**[17] – The *Security Center* compromises a central view of all security in the network and all *Microsoft Azure* services. Integrates with *Sentinel* and *Defender* to gain a complete overview.

[11] `https://aws.amazon.com/iam/`
[12] `https://aws.amazon.com/api-gateway/`
[13] `https://aws.amazon.com/cloudwatch/`
[14] `https://azure.microsoft.com/en-us/solutions/networking-and-network-security/`
[15] `https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-private-access`
[16] `https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id`
[17] `https://azure.microsoft.com/en-us/explore/security`

In addition, it can perform security assessments for all types of workloads [10]. The *Security Center* focuses on the *Data* and *Visibility & Analytics* pillars of *zero trust*, where data and analytics are of importance.

- **Sentinel**[18] – *Sentinel* is a tool that provides security features throughout enterprises. This is done using new and historical data, where threat intelligence is applied. With *Sentinel*, you are notified as soon as it detects malicious behaviour and how further problems can be avoided. All of this is done based on *Microsoft* years of data to train artificial intelligence to supervise suspicious activities [25]. *Azure Sentinel* falls under the automation & orchestration pillar of *zero trust*.
- **Defender for Cloud**[19] – *Azure Defender* is another product that *Microsoft* offers to apply security features in the cloud. *Defender* offers a wide range of features, where security features are the focus. Not only does it provide suggested changes, but it also automatically changes some security measures where threats are found. Thus, it can adaptively provide the customer with reports and protection on the fly [25].
- **Firewall**[20] – *Firewall* is a tool to manage and enforce firewall policies in a network. It also allows a connection to the segmented virtual networks in *Network Security* where it can specify the policies that are applied to the networks.

Using a combination of the tools and services mentioned above, the systems hosted on *Microsoft Azure* are able to comply with the principles of *ZTA*. How they can be applied to create a system that is *ZTA* compliance will be discussed further in Section 4.4.

2.6 Research

This section aims to clarify the methods to be used in the investigation of the topic. It also acts as a place to define how solutions will be evaluated.

2.6.1 Research Strategy

When researching the topic of *ZT* and *ZTA*, it will be important to find relevant and trusted articles. The search for articles will be done on Google Scholar or similar sites. Here, it is important to try to differentiate between relevant articles quite quickly as there are quite a few that could be of interest. Thus, articles are first evaluated based on the publishing year, the citations, the publisher, if they have been peer reviewed, and the journal they are published through.

The practical study will be performed as a case study in which a test bed system will be used to compare solutions. The system will be chosen as a minimal example where the focus is on implementing and measuring the network latency overhead by each solution. Thus, keeping the complexity of the service low may provide better results.

2.6.2 Comparing Solutions

The comparison of *Amazon AWS* and *Microsoft Azure* will be done using the principles of *ZT* presented in Section 2.4. In more detail, the comparison will be done by mapping the tools and services offered to the principles of *ZT*. This will be done to see how well the solutions cover the principles of *ZT*. Some areas linked to the principles that will be covered are:

- **Verify Explicitly** – What resources and information are used to assess access?
- **Least Privilege Access** – How are the networks designed to allow only access to the resources that are needed?
- **Assume Breach** – What is done to monitor for breaches?

[18] `https://azure.microsoft.com/en-us/products/microsoft-sentinel`
[19] `https://www.microsoft.com/en-us/security/business/cloud-security/microsoft-defender-cloud`
[20] `https://azure.microsoft.com/en-us/products/azure-firewall`

The solutions in the practical part of the study are done with respect to latency. This will be done using a load generator that interacts with the test-bed where it simulates user behaviour. With it, the solutions can be compared with each other and with the base case without *ZTA*. The practical part will also evaluate the steps needed to achieve an *ZT* environment.

2.6.3 Evaluating Performance

Evaluating performance in the cloud is best done in the same way as normal applications; applying load through benchmarks. These benchmarks are then recorded through information gathering tools [27]. A set of tools that are commonly used is *Prometheus*[21] for metrics scraping and *Grafana*[22] for visualization. In this project, metrics such as CPU usage, memory usage, and latency are of interest.

Performance tests will be run on two different providers. There will be inconsistencies between the hardware of the nodes. Therefore, it is also important to register the hardware for evaluation. In addition, if it is not specified in the setup, the nodes may vary between each other. Thus, it is always important to measure the available resources both before and under benchmarks [27].

In addition, the two providers must be evaluated without *ZT* to have a base-level latency. The reason for this is the same as mentioned before, that the hardware and in this case, software are different between the two solutions. Thus, the change between the base case and the applied *ZT* performance metrics that will be compared. How these metrics will be collected will be discussed further in Section 4.1.4.

3 Comparing Solutions

ZTA is a broad topic when it comes to how it should be set up for security. Thus, this section will focus mainly on the aspects that are of importance to the later practical evaluation discussed in Section 4.

Both solutions offered by *Amazon* and *Microsoft* build on the same concept of *ZT* introduced in Section 2.4. However, their solutions for *ZT* vary from each other as they are developed and maintained by two different companies. This section will compare the solutions offered by *Amazon* and *Microsoft* in terms of *ZT*.

Sections 2.5.1 and 2.5.2 introduced the solutions that are offered by *Amazon* and *Microsoft* for *ZT*, respectively. In both solutions, the principles of *ZTA* are applied. However, the solutions differ in terms of the tools and services that are used to enforce *ZT*. Table 2 shows the tools and services that are offered by *Amazon* and *Microsoft* mapped towards the three principles of *ZTA*.

Table 2: Offered tools and services mapped towards the three principles of *ZT* presented in Section 2.4.

	Verify Explicitly	**Least Privilege Access**	**Assume Breach**
Amazon AWS	Verified Access VPC Lattice Verified Permissions GuardDuty API Gateway	VPC Lattice Verified Permissions IAM API Gateway	VPC Lattice GuardDuty CloudWatch
Microsoft Azure	Entra ID Security Center Sentinel Defender in cloud Firewall	Network Security Entra ID Defender in Cloud Firewall	Network Security Entra ID Security Center Sentinel Defender in Cloud Firewall

Table 2 shows that both solutions offer tools to cover the three principles of *ZTA*. However, the tools and services offered by *Amazon* and *Microsoft* differ. At first glance, it seems that *Microsoft* offers more tools and

[21]`https://prometheus.io/`
[22]`https://grafana.com/`

services to enforce *ZTA* compared to *Amazon*. This may be the case, but it cannot be assumed on the basis of numbers alone. For this, each tool must be inspected to see how it enforces *ZT*. Where the functionality of multiple tools offered by *Microsoft* may be combined into fewer tools for *Amazon*.

3.1 Overview

Inspecting the overviews of *Amazon* and *Microsoft* architecture for *ZT* in Figure 12 and Figure 15, the two illustrations are quite similar in the overall structure of the systems. However, the tools used are different except for the implementations of the client, *API* and *database*.

3.2 One Approach Does Not Fit All

Before going into more details, *ZTA* do not offer out-of-the-box solutions in most cases. Instead, solutions must be tailored to the needs of the system through a combination of configurations and services. The set of defined policies for the system will be different depending on the use case. Thus, the set of policies will often not overlap between the systems and its services.

The services offered by the system will also determine the *ZT* solution. For example, a system that is used to store data will have a different approach to security than one that processes data. Therefore, it is important to tailor the solutions to the specific needs of the application.

However, even though there is no generic security solution, there are some techniques that are generally used by each solution. As an example, one of these techniques is used for network segmentation. Segmentation is a core concept of *zero trust* and therefore requires some service to create and secure subnets. The topic of network segmentations will be discussed further in Section 3.3.1.

3.3 Access Control

Accessing a cluster with resources has traditionally been done using a *VPN*. However, with the introduction of *zero trust*, the need for additional security measures has been introduced. Instead, both *Amazon* and *Microsoft* offer services used for connection to *ZT* clusters. *Amazon* offers *Verified Access* that allows authenticated users to connect to resources without the need for a *VPN*. Instead, this is done by a single tool that secures the connection and applies custom policies. *Microsoft* offers their tool *Entra Private Access* that is used for the same purpose [5].

Authorization is a large part of the access control process. Thus, it is important to provide services and tools that can enforce policies and keep track of identities. Therefore, both providers have services that manage identities. *Amazon* has their *AWS Identity and Access Management* (*IAM*), while Microsoft has their *Azure Entra ID* [5] service. Comparing the two services is difficult, since both parties hide most of the information about implementation. However, both services can be said to be used to enforce policies and keep track of identities.

Both services are implemented to serve authentication outside the main logic of the application. This feature is pushed by both parties as a way to accelerate the development of new applications [5]. Not having to deal with the main authentication process for applications is a huge time saver. Without it, developers would have to develop the authentication process for each new application. Now, they can use the service to authenticate users and devices in a secure and fast way.

3.3.1 Resource Segmentation

The second principle of *zero trust* is to apply the least privilege access. This is done for both solutions through segmentation. Resource segmentation is the process of splitting a network into subsets with different levels of authentication. This approach allows users to access only the services that are required based on the current task. Thus, removing the ability to move around in the network once they have connected.

The segmentation is done for *Amazon AWS* in the *VPC Lattice* service. *Amazon* is quite secretive in how its solutions work in practice, but is known to segment resources. This is done by creating a custom connection between the resource and *VPC Lattice*. Once the connection is completed, it closes all other ports and network interfaces. Thus, in theory, removing it from the internet and outer observability [31]. *VPC* also allows the creation of links between different security groups. This link will then only be accessible from one desired service. Thus, removing the possibility for unwanted external access.

Microsoft solves segmentation by the *Azure Network Security* tool. Resources in *Microsoft Azure* are separated by virtual machines (*VM*). These virtual machines are then split into groups by virtual networks. The virtual networks are then built on top of the physical network to allow segmentation between services. Thus, *Microsoft Azure* offers a greater customizability of the network for their users.[23] The grant of access between services is made using *peering*. *Peering* is the process of creating a link between two resources where access is allowed in one or both directions.

Comparing the two solutions in functionality gives the same result. Both solutions offer an effective way for the developer to configure virtual networks that restrict access to resources based on levels of authority. Therefore, the main difference is the wording and implementation in which they solve the problem similarly.

3.3.2 Applying Policies

Policies are used by all systems to specify access privileges for users and devices. Both *Amazon* and *Microsoft* utilize different languages for the writing of policies. Thus, making it difficult for users to switch between providers.

In evaluating *zero trust*, they decided that they did not want to force *IAM* on every developer. However, *IAM* is the service they offer for this purpose. Therefore, they developed a new policy language that was intended to be easy for developers to learn and use while keeping it professional. The new policy language is called *Cedar* and is used in the *Amazon Verified Permissions* service [3]. Listing 1 contains two policies written in *Cedar* that permit inserts and selects but forbid updates on a *PostgreSQL* database.

Listing 1: Creation of permit and forbid policies in *Cedar policy* language.

```
permit (
    principl,
    action in [
        Action::"INSERT",
        Action::"SELECT"
    ],
    resource == Application::"PostgreSQL"
);

forbid (
    principl,
    action == ACTION::"UPDATE",
    resource == Application::"PostgreSQL"
);
```

Microsoft uses the *JSON* format to specify policies. The choice of *JSON* as the format is based its standardization, flexibility and seamless integration with other *Microsoft Azure* services. In addition, *JSON* is a well known format by most developers, making it easy to adopt. The creation of new policies can easily be done either through the *Command Line Interface* (*CLI*) or by defining them directly in the policy manager. In both cases, *JSON* is used, where it is defined in the *CLI* by creating a policy definition. The main difference between the two approaches is the creation of the policy, either by commands in a *CLI* or manually through a file explorer.[24] Listing 2 contains the creation of an *Microsoft Azure* policy through a *CLI*.

[23]https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview
[24]https://learn.microsoft.com/en-us/azure/governance/policy/tutorials/create-and-manage

Listing 2: Creation of an *Microsoft Azure* policy through a *CLI*.

```
az policy definition create --name 'denyTestingTier' --rules '{
    "if": {
        "allOf": [{
                "fields": "type",
                "equals": "Microsoft.Storage/storageAccounts"
            },
            {
                "fields": "kind",
                "equals": "BlobStorage"
            },
            {
                "fields": "Micorsoft.Storage/storageAccounts/accessTier",
                "equals": "Testing"
            }
        ]
    },
        "then": {
        "effect": "deny"
    }
}'
```

3.4 Breach Monitoring

Assuming breach is a central principle of *ZT*. Therefore, it is important to always assume a breach and monitor every activity on the network.

GuardDuty is the tool in *AWS* for monitoring the network. As introduced earlier, *GuardDuty* utilizes *AI* and historical data to search for inconsistencies between requests. In addition, system specific policies are applied to the *IAM* where the authorization of devices is done on a time-to-time basis [31]. Thus, it can handle vulnerabilities not related to the location where the breach occurred.

Microsoft supplies a few different tools and services that are used to monitor breaches, where some of them work together and some individually. The tool used by *Microsoft* in Section 3.3.1 also monitors network breaches. It works closely with the *Azure Security Center* to monitor and filter network traffic on the network [10]. There is also *Azure Sentinel* that uses historical data [25], both from the specific application and from *Microsoft* themselves, to determine malicious activities. In network management, there is also the *Azure Defender* that is capable of applying security measures directly to the network or systems within the network to prevent breaches [25].

Comparing *Microsoft Azure* and *AWS* in breach monitoring. It can be seen that *AWS* generalizes it into a few tools that are more generalized in managing breaches on the *VPC Lattice*, while *Microsoft Azure* divides it into different solutions that are specified on one problem. Both of these approaches have strengths and weaknesses. Generalizing services makes it easier for developers to standardize their security. However, this can lead to problems where generalization removes security due to repeated configuration and oversight in different solutions. The same case is for *Microsoft Azure* but in the other direction, where too many tools lead to uncertainty in the setup and configuration.

3.4.1 Logging Access

Logging access and metrics is an important part of any system, especially when it comes to security. Having access to the access history may be the only way to determine that there has been a breach of access. Therefore, logging is needed to help find bugs and secure systems. Both *Amazon* and *Microsoft* offer logging services that are used to log access to resources.

For *Amazon*, the logging is done in the *VPC Lattice* service and collected in *CloudWatch*. In addition, all logs related to security from services are also collected and stored by *CloudWatch*. The logs are then used by services as *GuardDuty* to determine if there has been a breach of access.

On the other hand, *Microsoft* logs data from all services that are connected to the network and stores it in the *Azure Security Center*. These logs are then used to determine whether there have been any breaches of access. In addition, logs are used to determine whether a request should have access to a resource.

When comparing the two solutions, both have the same approach to logging. Where data, regardless of origin, are stored in a central location for ease of access.

3.5 Data Security

Securing data is generally an important topic in any system. Especially if the data stored are private or sensitive. Therefore, it is also of great importance to offer secure storage for services that comply with *ZT*. Without secure storage, the need for *ZTA* would be redundant since the resources can be accessed by other means.

Neither of the solutions offered by *Amazon* and *Microsoft* lock you into a specific storage system. They encourage developers to use the services they offer for storage. But that is not required. The developer can use any storage method, as long as it is deemed to be secure. Moreover, the storage system needs to be accessible by the network, where restrictions can apply by policies.

Transferring data introduces the possibility for external partners to view the data in question. This type of attack is called a “man in the middle” attack, in which a user can read or modify data between the sender and the receiver. To combat this, end-to-end encryption is used. This report will not go into detail on how end-to-end encryption works, but it is required for *zero trust* to secure the transfer of data between services and devices. Thus, both *AWS* and *Microsoft Azure* offer solutions for encrypted data.

Encrypting data at the *database* level is also of great importance. If not, the data could be accessed by anyone who has access to the storage device of the service. Therefore, it is also important to encrypt all data that is stored. Both *AWS* and *Microsoft Azure* does this by default for any *database* service hosted. Accessing the encrypted data can be done in a few ways. However, using the *IAM* for both providers is the safest option where it can apply *MFA* to the authentication process.

3.6 Zero Trust Coverage

The topic of *zero trust* coverage is new for both *AWS* and *Microsoft Azure*. On the one hand, there are no previous publications that evaluate the coverage of *ZT* for *AWS*. On the other hand, there is a publication by Vedran Dakić [8] that tries to evaluate the coverage of *Azure ZT*.

Dakić work consists of an evaluation of the *Microsoft Azure* solution for *ZT*. The report raises quite a few vulnerabilities and oversights of *ZT* in *Microsoft Azure* that come from limitations or oversights. One of these oversights is the management of resource segmentation that diverges from the usual recommendations of *ZTA*. In addition, Dakić states that there is an overhead of *ZTA* that affects the user experience. In this case, he asks when it is in the best interest of a developer to disregard some security features to improve user experience [8].

Furthermore, in the current versions of *Azure ZT* there is a technical complexity that requires prior knowledge in the domain to effectively apply the principles of *ZT*. The process of setting up *zero trust* in cloud environments is complicated, where every service requires additional authentication and policies. Moreover, these additions require constant maintenance to keep up with current threats as they are brought to light [8].

4 Case Study

This section will be used to define the system used in the practical evaluation of *AWS* and *Microsoft Azure*. The system is designed to be as simple as possible in which the properties of *ZT* can be used. Thus, it will consist of a small Application Programming Interface (*API*) service that fetches and stores data in a *database*. Figure 8 illustrates this system without surrounding services.

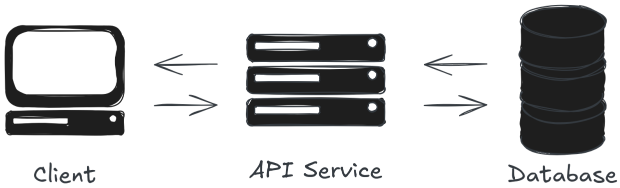


Figure 8: Illustration of test-bed architecture without surrounding services.

4.1 Overview

The system to be tested must be implemented twice without and twice with *ZT*. Therefore, it is important to generalize the code, so it can be used in multiple environments. This is important where the network *ZT* overhead should not be affected by the implementation of the code. Moreover, a generic code then provides a static latency cost regardless of the environment. However, it will still be affected by the hardware on which it runs.

Scaling services are an important aspect to consider when applications are running in the cloud. However, in measuring the network performance cost for *ZT* it adds another layer of complexity and, therefore, can be removed to easier estimate the latency overhead of *ZT*. Therefore, the environments will be static and not scalable during testing.

The core of the system consists of a *database*. The *database* will be used to store information about users, accounts, and transactions that have or are to be applied to the accounts. Before *zero trust*, the information present in the *database* did not have to be encrypted by default. Instead, it uses only static roles to access the data. This approach lacks flexibility, where sophisticated cyberattacks could easily exploit user authorization if access was compromised once.

To interact with the *database*, there must exist one or multiple *API*. The *API* is the application that extracts data from the *database* and returns relevant objects to the user.

Managing identities and authentication of users and devices is done similarly to what is presented in Section 2.4.1. Authentication of users is done by sending credentials to a *database* where the encoded log-in information is stored. This *database* then accepts or denies access based on login information, where it returns an access token. Once a user has gained access, it will have access to the data until it either leaves or is kicked out of the network. This process is illustrated in Figure 9.

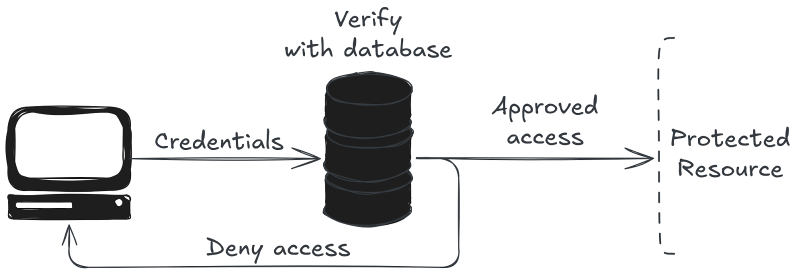


Figure 9: Illustration of the authentication process where *ZTA* is not applied.

The authorization of users is done with the log-in token previously given to the user to gain access. This token is passed into the *PEP* and the *PDP* to check the validity of the token. If the token is valid, the user is given access to the services provided. In this case, the *API* and *database*. The authorization process is illustrated in Figure 10.

Authorization and authentication are provided by *AWS* and *Microsoft Azure* by their respective services. In addition, the services provided allow for the creation of new services that utilize and depend on the procedures.

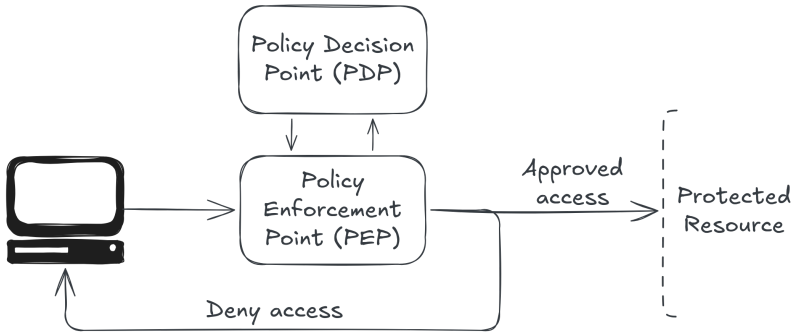


Figure 10: Illustration of authorization process when *ZTA* is applied.

Therefore, minimizing the work required by the developer to comply with policies.

4.1.1 Test Client

To perform and logic tests on the system, a test client has been developed. This client is written in Java and serves as a generic gateway to access all systems without the need to recompile the code. At runtime, the user can specify the host address, the size, and amounts of requests to send. From this, the client tries to establish a connection first with an *AWS* signature, where if it does not work, it tries with a valid *Microsoft Azure* access token. The credentials for these requests are taken as specified in the respective documentation from the local computer. For *AWS*, the signature is created from the credentials specified in the "∼.aws/credentials" file, while *Microsoft Azure* uses the active login session for the *Azure CLI* tool.

Once the application finishes, it stores all the resulting values in a local folder for later use in calculations in *CSV* format. In addition, runtime metrics are exported in the *Prometheus* format through a web service that runs alongside the application, so they can be accessed for visualization in a *Grafana* dashboard.

4.1.2 Application Programming Interface

The *API* is a *Spring Boot* application that implements an interface to access the data in the *database*. During testing using the client, only a few of the endpoints implemented allow access to stored transactions.

To increase the security of the transmissions from the *API*, *TLS* has been applied to the encryption of the data. This encryption has been performed through the *AWS API Gateway* and the *Azure Endpoint Management* services where it can be enabled by clicking a button.

For metrics, the *Spring Boot* application uses an extension that allows metrics to be exported through an endpoint in *Prometheus* format. This endpoint is added by adding a build dependency of the name *"micrometer-registry-prometheus."*

4.1.3 Database

The *database* consist of a *PostgreSQL* instance that hosts two tables of data.[25] The first table contains all open accounts with information about the owner, identification, balance, and a few values that are not important here. The second table contains all transactions that are assigned to the accounts. These transactions contain information about the withdrawal amount, the deposit amount, the registration time, the time to be applied, the account identification, and information regarding the verification of the transaction. Upon completion of the transactions, the account balance is updated to reflect the new value after deposit or withdrawal.

To allow metrics to be extracted from the *database*, a direct connection is made from the *Grafana* dashboard to the *database*. This connection allows metrics about resource usage and latencies to be used in *Grafana*.

[25] `https://www.kaggle.com/datasets/apoorvwatsky/bank-transaction-data/data`

4.1.4 Metrics

Both *Amazon* and *Microsoft Azure* have their own tools to collect data on the cloud. However, to give them a fair comparison, neither of the standard metric scrapers will be used to gather the results presented in Section 5. However, the tools offered by default will still be used to comply with the principles of *ZT*. Instead, a combination of *Prometheus* and *Grafana* will be used. *Prometheus* is the data scraping service where it scrapes and stores logs from defined systems and services. *Prometheus* is a well-defined and accepted scraping service, where most frameworks provide a default way to export metrics in the correct format. In its own, *Prometheus* do not have a good way to visualize information; this is where *Grafana* is of use.

Grafana is a service that allows data visualization when other services have retrieved data. In this case, it connects directly to the *Prometheus* instance and the *DB*.

As this implementation is used to evaluate the performance of the two *ZTA* solutions, then it is important that the client metrics can also be scraped. This cannot be done by default with *Prometheus*. Therefore, the principles of *ZT* policies will contain minor changes that allow scraping of the generating client. This change will consist of a security policy that allows access to client metrics from within the cluster, where a single port and IP combination will be opened for access. The general architecture for retrieving and managing metrics can be seen in Figure 11.

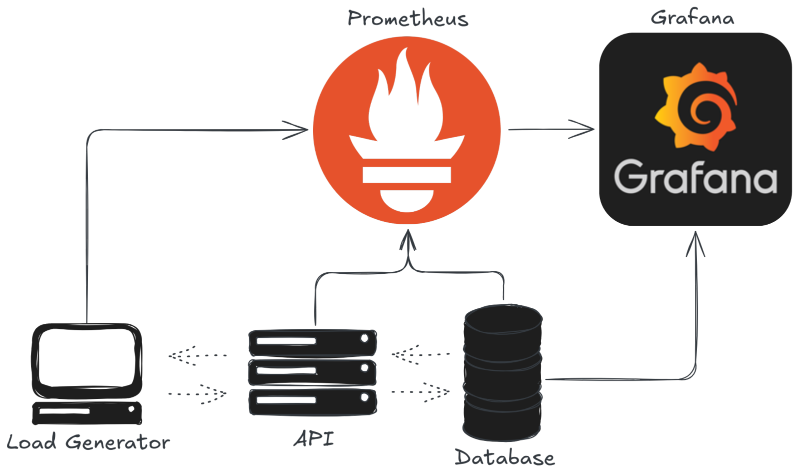


Figure 11: Overview of metrics gathering in the system by *Prometheus* and *Grafana*.

4.2 Verifying Zero Trust Compliance

The implementations of *zero trust* for the two solutions will be based on the guides provided by the service providers. As mentioned in Section 3.2, the solutions are likely not to be able to be applied directly without any modifications. Thus, the finished results must be verified for *zero trust* compliance.

Since *ZT* require segmentation of resources with different rules. Then, access to resources can be verified by removing parts of the users access. For example, a users' access to the *database* can be removed while still having access to the *API*. In this case, the request from a user would be authorized all the way to *API* before it would be rejected when trying to access the *database*. This overstepping of privileges would then be logged by one of the defence services, where measures could be taken.

To verify the *ZT* compliance on a request level, the built-in services for logging are to be used. Taking *AWS* as an example, *CloudWatch* is used to log all requests and data sent between services. *CloudWatch* have tools and daemons that are running in services that send information about the load that is computed. Thus, it gains the ability to visualize how requests are passing through the system in a centralized service. This behaviour has also been verified for *Microsoft Azure* with its respective services.

4.3 Amazon Web Services Zero Trust

Implementing the case study system in *AWS* can be done with an *EC2*[26] virtual machine and an *RDS*[27] instance. In addition, all systems hosted on *AWS* require a *VPC* where it manages the virtual network. *VPC* also provides a connection between the *API* and the *database*. Thus, almost all the components required to gain access to the *API* are provided by default.

Accessing the network and services from outside requires a connection using *Verified Access*. This connection allows users outside the network to gain access to the *API* service. The final non *zero trust* architecture can be seen in Figure 12.

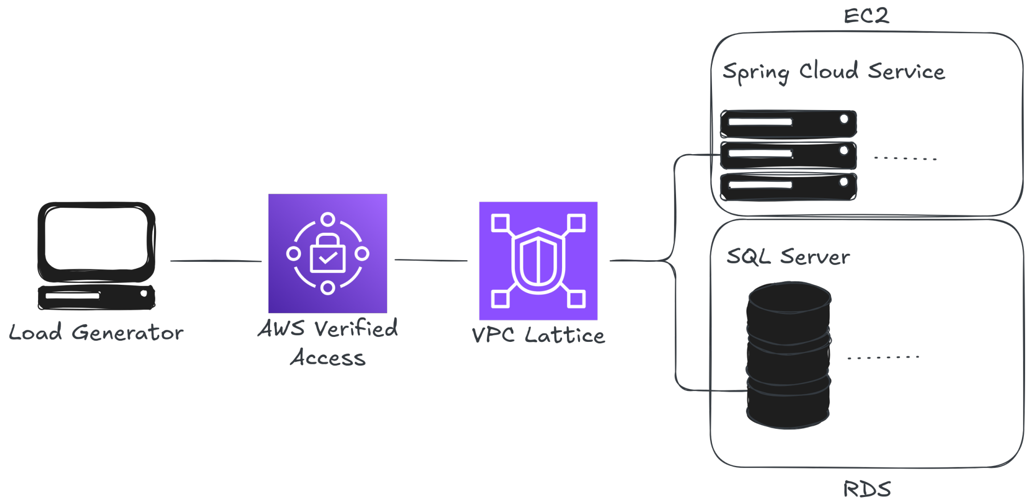


Figure 12: System overview of *AWS* implementation without *ZTA*.

As mentioned in Section 2.6.3 it is important to also specify the hardware used during testing to be able to compare two or more performance results. Thus, Table 3 contains the resources used by the different services hosted on *AWS*.

Table 3: Application placement on *AWS* with respective resource.

Application	**AWS Service**	**Resources**	**Region**
PostgreSQL database	Relational *database* Service (*RDS*)	vCPU: 2 RAM: 1 GB	EU-North-1
Spring Boot API	Elastic Compute Cloud (*EC2*)	vCPU: 2 RAM: 1 GB	EU-North-1

Moving on to applying *ZT* for *AWS*, The first step of creation is to recreate the structure illustrated in Figure 12. In creating the new environment, it is important to specify the security level of each part to align with the principles of *ZT*. Creating the *RDS* and *EC2* instances are done with a few security settings enabled. For the *RDS*, it is important to disable public access. Thus, it must be implicitly specified which services in the network have access to the *database*. Furthermore, the authentication of the *database* should be increased. It is not sufficient for *ZT* to only rely on *database* passwords. Therefore, the *IAM* needs to be integrated into the login process and replaces all other forms of authentication.

Setting up the *API* closely resembles the process of the *database* with some additions. As stated above, the network needs to be secured to allow access only from trusted sources. In addition, the *API* need to be secured

[26] `https://aws.amazon.com/ec2/`
[27] `https://aws.amazon.com/rds/`

by *TLS* and *Signature V4*[28] (*SigV4*) to allow authentication of *API* endpoints. The authentication used in *SigV4* can be specified as *IAM* to ensure that all authentication is centralized to minimize complexities in the access structure. This extra step is performed by applying a *VPC Link* and an *API Gateway*. The *VPC Link* acts as a pointer that allows requests to pass through a link in the *VPC* without opening unnecessary paths, where the *API Gateway* allows for outer access where authentication and encryption are applied. Figure 13 contains an illustration of the request path between the caller and *API*.

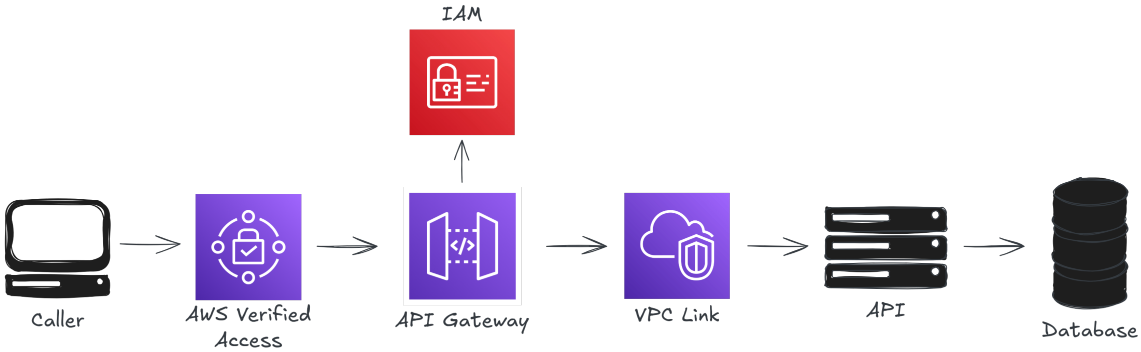


Figure 13: Request path in *AWS* between the caller and *API*.

Specifying the policies used in authentication is done directly in *IAM*. *IAM* allows for the creation of users and groups where the groups contain the policies for access to resources. In the case for accessing *API* resources, the groups require the policy to invoke *API* resources. The assignment of a user to the group then allows it to invoke the endpoints defined in the *API Gateway*.

The next step in securing the network is to modify the security groups in the *VPC*. By default, these groups do not allow any inbound or outbound request. However, during setup and testing, it is quite easy to add allowed connections that are not wanted for security reasons. For example, the *RDS* only need an inbound connection from the exact IP address of the *API EC2*.

For monitoring the connections over the network, *Amazon* offers the tool *GuardDuty*. *GuardDuty* is a tool that can be used to detect users and requests that access unprivileged resources. Once found, it can notify the security owners of the systems that there has been a breach. In *GuardDuty*, there are a few options that are of interest to this small system.

1. **Extended Thread Detection**[29] – Has the ability to monitor for attacks targeting a users account, workloads, and data. With the information provided by the *Extended Thread Detection* the maintainer should have an easier time finding and path vulnerabilities.
2. **Runtime Monitoring**[30] – Monitors the network and activities when it comes to the *EC2* virtual machines. It allows *GuardDuty* to directly intercept malicious activities to stop them from executing.
3. **Malware Protection for EC2**[31] – Scans *EC2* instances for malware and vulnerabilities.
4. **RDS Protection**[32] – Monitors the log-in event for your *database*s that can be used to check for vulnerabilities.

To monitor access and metrics in *AWS*, *CloudWatch* is configured. *CloudWatch* is a service that is hosted by *AWS* and runs in the background. Accessing *CloudWatch* allows authorized users to view access logs and metrics for all services that have *CloudWatch* logging capabilities.

[28] `https://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-authenticating-requests.html`
[29] `https://docs.aws.amazon.com/guardduty/latest/ug/guardduty-extended-threat-detection.html`
[30] `https://docs.aws.amazon.com/guardduty/latest/ug/runtime-monitoring.html`
[31] `https://docs.aws.amazon.com/guardduty/latest/ug/malware-protection.html`
[32] `https://docs.aws.amazon.com/guardduty/latest/ug/rds-protection.html`

With the security measures mentioned in this section, the application presented in Section 4.1 can be secured with the principles of *ZTA*. Thus, it is more secure than if these tools had not been used. Now, a full view of the services used can be created, which is shown in Figure 14.

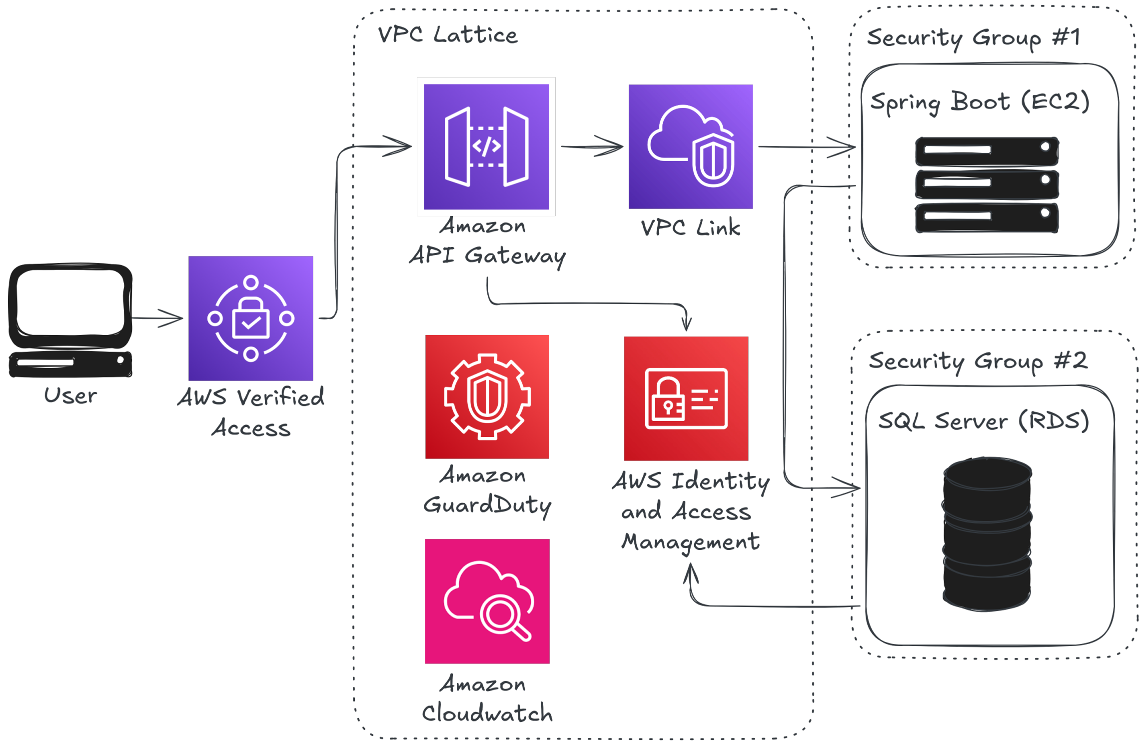


Figure 14: Full *ZTA* for system hosted on *AWS*.

4.4 Microsoft Azure Zero Trust

The system hosted on *Microsoft Azure* is to have the same functionality as for *AWS*. Therefore, it will also use the same code for the *API* and structure of *database*. Before applying *ZT* measures, it is important to get a base reading of the latency for calls. Therefore, the system is first created in *Microsoft Azure* with default configurations. The default configuration comes with a few security features, such as rough segmentation and *TLS*. However, it is not compliant with *ZT*. An overview of the non *ZT Azure* implementation can be found in Figure 15.

As stated in Section 2.6.3, it is important to list the hardware for later comparison. Therefore, the hardware specifications for the *Microsoft Azure API* and *DB* are listed in Table 4.

Table 4: Application placement on *Microsoft Azure* with respective resource.

Application	Azure Service	Resources	Region
PostgreSQL database	*Azure database* for *PostgreSQL*	vCPU: 1 RAM: 2 GB	North Europe
Spring Boot API	Linux Virtual Machine	vCPU: 1 RAM: 2 GB	North Europe

The access to the network is done using *Microsoft Entra Private Access* (*MEPA*). *MEPA* is a replacement for traditional *VPNs* where it allows the connection to directly authenticate and act on the policies present in *Microsoft Entra ID*. Thus, most of the authentication of users does not need to be implemented by the developer

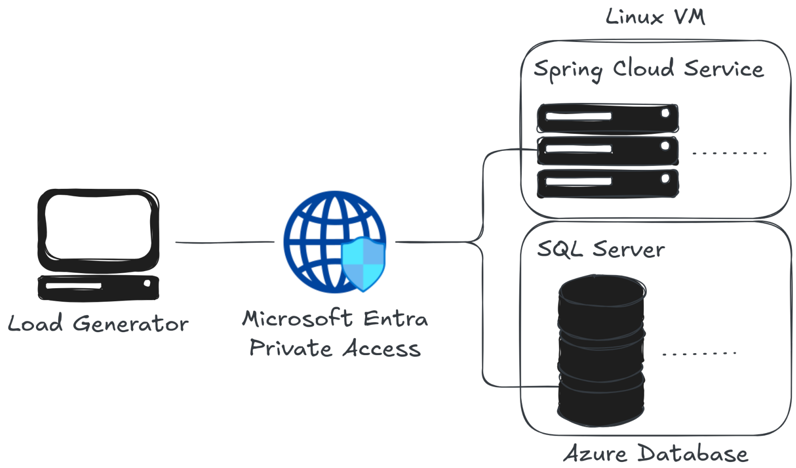


Figure 15: System overview of *Microsoft Azure* implementation without *ZTA*.

but is enabled by default with the ability for policies to specify access. Therefore, it allows developers to increase the security of applications more quickly.

Microsoft Azure utilizes virtual networks in segmenting the network. This allows for the setup of policies that limit each user and devices access to the network. In the context of the test bed presented in Section 4, there are two virtual networks that must be created. The first and most locked virtual network surrounding the *database*. This network serves as a security layer where no IP address or ports are provided. To access the contents of the *database*, the other virtual network of the *API* need to peer into the *database*. Thus, create a link that can be accessed only by authenticated users. In addition to these two virtual networks, a few services such as *Azure Security centre*, *Azure Defender*, *Microsoft Entra ID* and *Azure Sentinel* are running in the same network, but outside the virtual networks. However, security services have access to virtual networks to scan for threats.

The security of the subnets in the cluster is also done using *Azure Firewall*. *Firewall* is a tool that allows the developer to specify access policies for different resources on the network. This is done in a stateless environment to protect from threats outside the network.

Authentication to the services are done with *Microsoft Entra ID*. This is done to integrate the authentication process with *Microsoft Azure* where it does not have to be done manually. In addition, utilizing *Entra ID* allows the service to use the same accounts and users as other *Microsoft services*. Thus, centralizing all account management to one place regardless of service. Securing with *Microsoft Entra ID* also comes with *Multi-Factor Authentication* (*MFA*), where the administrator is able to force *MFA* onto users.

The access to the virtual network is done though the *Microsoft Endpoint Manager* where it sends an authentication request to *Microsoft Entra ID*. *Entra ID* either approves or denies the token based on whether it can be verified as valid. If approved, the user gains the ability to fetch resources as long as no other security measure denies access. This process can be visually seen in Figure 16.

To be able to log and detect suspicious devices on the network, *Azure Sentinel* is used. *Sentinel* is a tool that is deployed on the network without a direct connection to any other services. By default, it runs by itself where logs and other useful information is uploaded to it. The logs and data are uploaded to *Sentinel* to perform complex pattern recognition to try and catch unlawful users on the network. Thus, applying *Azure Sentinel* increase the latency, but it can be used to catch quite a few of the known security problems.

Microsoft Defender is another service that is used for monitoring a *Microsoft Azure* cluster. In addition, *Defender* is able to apply direct measures and policies based on the activity on the network. Thus, it can automatically defend against malicious activities without needing input from the developer. For *Defender* to work closely with hosted services, it is given access to *API* and *DB* for monitoring. Furthermore, it is given

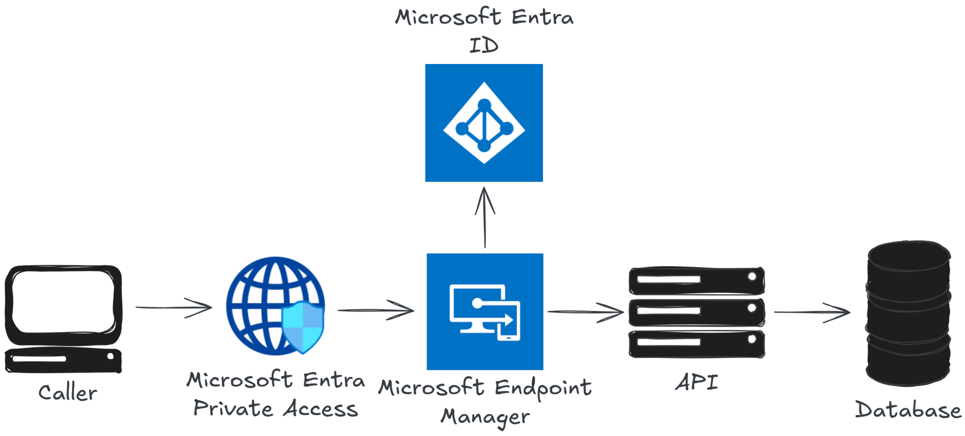


Figure 16: Request path in *Microsoft Azure* between the caller and *API*.

access to virtual networks to monitor for malicious activities.

With the security features and services presented in this section, a system hosted on *Microsoft Azure* is able to comply with the three principles of *ZTA*. Figure 17 illustrates how the tools are put together to make it possible to serve a client caller through the services.

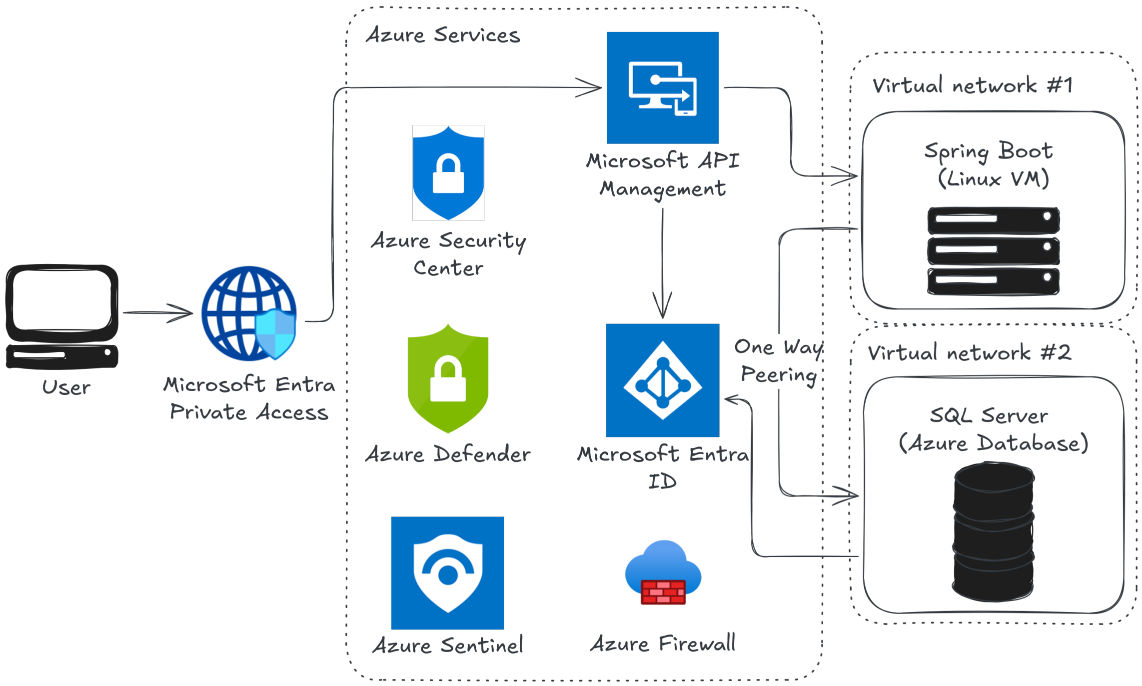


Figure 17: Full *ZTA* for system hosted on *Microsoft Azure*.

4.5 Comparing Case Study Solutions

When comparing the solutions for *ZT* for both *AWS* and *Microsoft Azure*, it can be seen that the solutions are similar in practice. The main difference between the two solutions is the services that are used. Although the services are different, they both succeed in achieving the principles *ZTA*.

Both solutions provide a service for authorization and authentication. These services are *AWS IAM* and *Azure*

Entra ID. Moreover, the services are used to solve the same problems similarly. Thus, jumping between services is as easy as setting up groups of users and policies in a new interface and policy syntax format.

Similarly to the authentication services, the *API* are both made accessible through some form of gateway. The services provided here are *Amazon API Gateway* and *Microsoft API Management*. Both services act as an interface where endpoints are specified to pass requests to the *API* service hosted on a virtual machine behind a closed virtual private network.

VPNs are not required in either of the solutions. Instead, they provide their own solutions that gain access to the network through private connections. These solutions are *AWS Verified Access* and *Microsoft Entra Private Access*. Providing your own connection to the networks allows them to have more access over the traffic that comes into the network, where requests can be authenticated before they reach the network.

Before moving on to the tests, it is important to note that the services between the providers do not have the same hardware specifications. Thus, this must be included in the performance evaluation. Compared with Table 3 and Table 4, it can be seen that the difference is for the *API*, where *AWS* provides more cores but less ram than *Microsoft Azure*. The choice for these specifications was made in line with the free tier services that are provided.

5 Results

Performance was benchmarked in a total of four different configurations. Firstly, the first two configurations will be in the minimal security examples that are set by default by *AWS* and *Microsoft Azure*. Secondly, *ZT* will be applied to the networks and new performance metrics will be recorded.

With these data, the average latency change between the baseline and *ZTA* solutions can be calculated over 1000 sequential HTTP requests. Moreover, it will be able to calculate the latency cost for not only the client to *API*, but also for *API* to *database*.

To ensure that the services do not increase performance due to caching, all requests must randomly select the elements that are fetched. Moreover, this adds an extra feature that needs to be implemented. However, it is outside the scope that is recorded for performance metrics. For each randomized test to be as close as possible, the randomizer seed will be the same for all tests. Thus, it makes it so that it is the same sequence of randomized values that is fetched after one another.

Since *Prometheus* is used to scrape metrics, it is also of interest to get metrics from the client. Thus, the client needs to make the data public that *Prometheus* can scrape. In addition, it is wanted to have as low transfer times as possible to highlight the network overhead of *ZT*. Thus, in the test, the client will be hosted on a separate virtual machine inside the cluster, so the transfer speed and distance can be neglected.

5.1 Amazon Web Services Results

The presented data in this section have been gathered using *Prometheus* or directly from the applications to better understand the performance impact of *ZTA* in *AWS*. The tests performed are those introduced previously in this section where 1000 sequential requests with 5 different payload sizes are measured. Thus, resulting in the values presented in Table 5 and Figure 18.

The first thing to note about the results is the existence of extreme outliers. Each size of payload has one or a few outliers that are far greater than the others. These values are due to the cold start of the services experience, where the machine allocates resources to handle the request. This behaviour is expected and can thus be ignored as a problem.

Secondly, it is hard to understand the difference in latency between calls made with or without *ZT*. Therefore, the average times must be further inspected. Table 6 contains the average time difference for each size. Moreover, it can be seen that the average latency difference is between one and seven seconds regardless of size.

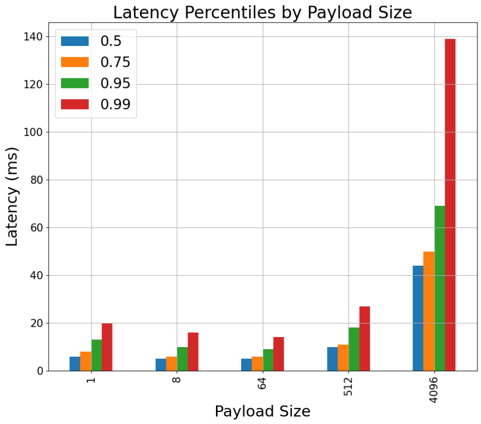
Table 5: Node metrics and latency for running tests on *AWS* in *ZT* and non *ZT* environments.

(a) Metrics for tests performed in a non *ZT* environment.

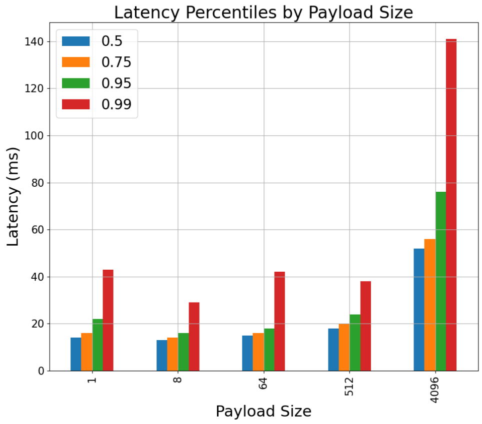
Payload Size	Max Memory	Max CPU	Average Latency
1	91%	25.5%	7.589ms
8	91%	15.9%	5.943ms
64	92%	17.8%	5.693ms
512	93%	18.4%	11.223ms
4096	93%	33.7%	48.748ms

(b) Metric for tests performed in a *ZT* environment.

Payload Size	Max Memory	Max CPU	Average Latency
1	87%	18.5%	15.696ms
8	88%	20.7%	13.470ms
64	89%	13.3%	15.612ms
512	89%	17.5%	19.694ms
4096	93%	32.2%	55.945ms



(a) Latency in milliseconds for network cost between the client and *API* without *ZTA* applied on *AWS*.



(b) Latency in milliseconds for network cost between the client and *API* with *ZTA* applied on *AWS*.

Figure 18: Calculated network latency of non *ZT* and *ZT* environments on *AWS*.

Table 6: Average latency difference in milliseconds between *ZT* and non *ZT* environments over 1000 sequential requests.

Payload Size	Average Latency Difference
1	8.107ms
8	7.527ms
64	9.919ms
512	8.471ms
4096	7.197ms

The third point of interest is the memory and *CPU* usages. Inspecting Table 5a and Table 5b, it can be seen that the memory and *CPU* usage does not differ much in usage. In all tests, the maximum memory usage was only two percentiles higher than the lowest maximum value. This is most likely due to the fairly small size of the data being transferred. Moreover, the *CPU* usage is fairly consistent. As the metrics are scraped in *Prometheus* every 15 seconds, the spikes are averaged in the calculation and thus missed. Therefore, the displayed values are the highest average of usage in 15 second intervals. With this, no correlation can be found between the load size and *CPU* usage.

Lastly, the change in latency for fetching data from the *DB* can be compared. Average times in Table 7a and Table 7b, where it indicates a very small increase in time for making the *database* call. This behaviour will be discussed further in Section 6.

Table 7: Latency metrics for *DB* queries.

(a) Latency for *DB* queries in a non *ZT* environment.

Payload Size	Average non ZT DB Latency
1	2.694ms
8	1.686ms
64	1.535ms
512	3.740ms
4096	21.766ms

(b) Latency for *DB* queries in a *ZT* environment.

Payload Size	Average ZT DB Latency
1	5.723ms
8	4.821ms
64	5.324ms
512	7.033ms
4096	24.312ms

(c) Latency difference between *ZT* and non *ZT* solutions for *DB* queries.

Payload Size	Average DB Latency Difference
1	3.029ms
8	3.135ms
64	3.789ms
512	3.293ms
4096	2.546ms

5.2 Microsoft Azure Results

In this section, the performance tests of the two *Microsoft Azure* implementations are presented. Starting of, the latency for fetching elements through the *API* is presented in Table 8 and Figure 19.

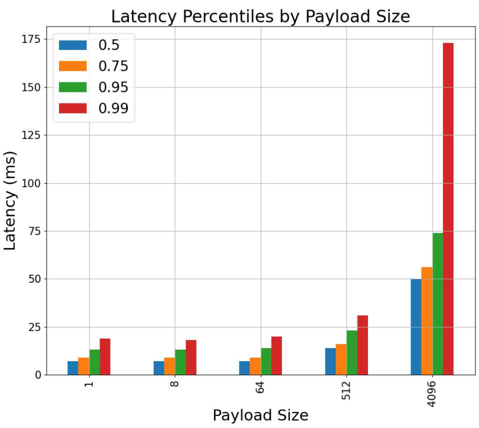
Table 8: Node metrics and latency for running tests on *Microsoft Azure* in *ZT* and non *ZT* environments.

(a) Metrics for tests performed in a non *ZT* environment.

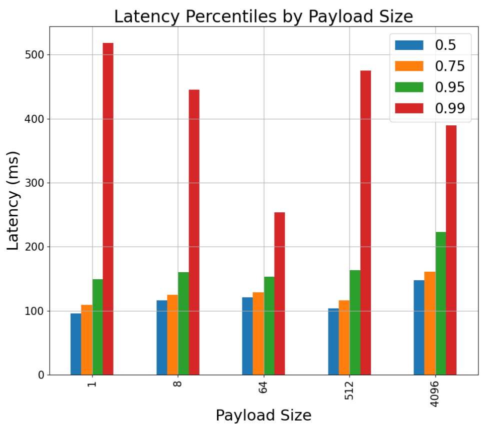
Payload Size	Max Memory	Max CPU	Average Latency
1	55%	24.1%	8.525ms
8	55%	27.6%	8.011ms
64	55%	29.7%	8.485ms
512	55%	40.0%	15.349ms
4096	55%	62.2%	54.987ms

(b) Metric for tests performed in a *ZT* environment.

Payload Size	Max Memory	Max CPU	Average Latency
1	60%	10.9%	113.758ms
8	61%	11.0%	124.941ms
64	61%	9.2%	127.484ms
512	61%	8.0%	118.425ms
4096	61%	28.1%	166.640ms



(a) Latency in milliseconds for network cost between the client and *API* without *ZTA* applied on *Microsoft Azure*.



(b) Latency in milliseconds for network cost between the client and *API* with *ZTA* applied on *Microsoft Azure*.

Figure 19: Calculated network latency of non *ZT* and *ZT* environments on *Microsoft Azure*.

Inspecting the resulting values in tables 8a and 8b indicates a clear increase in the average latency when *ZT* principles have been applied. This increase in latency can also be seen in the calculated difference shown in Table 9. The increase in latency can be averaged to approximately 110 milliseconds. In addition, the size of the request does not affect the overall latency of the request. This behaviour of a static increase in time is desirable for it to be scalable.

Table 9: Average latency difference in milliseconds between *ZT* and non *ZT* environments over 1000 sequential requests.

Payload Size	Average Latency Difference
1	105.233ms
8	116.930ms
64	118.999ms
512	103.076ms
4096	111.653ms

As with *AWS*, there are outliers in the results. Moreover, these are also caused by the cold starts that are performed for each test. Although these values contribute to a larger increase in the average results, they are worth keeping. Removing them from the calculations would remove a relevant factor from the result that may occur during normal user activity. The only case where it would be reasonable to remove them from the result is if the service is to be used continuously without downtime. However, this would also produce a lot more results, where they would drown in the result.

Another key feature that is visible when comparing Figure 19a and Figure 19b is the relation between the percentiles. Applying *ZT* increases the overall latency for all requests. However, looking just at the 99th percentile, there is a major increase in latency. In addition, the increase for the 99th percentile is not based on the payload size. Instead, it is entirely based on the cold start and how fast it is able to assign resources. Thus, there is no correlation between the 99th percentile between the different sizes in *ZT*.

As with the result of presented for *AWS* in Section 5.1, the maximum CPU usage does not follow a clear trend. The one thing that can be noticed is the decrease in max *CPU* usage for the virtual machine hosting the *API*. This is most likely because requests do not arrive as fast as without *ZT*. Thus, decreasing the load. However, there is a small increase in the maximum memory usage as the payload size increases. However, this increase is only for a few percentiles, as the payload is quite small.

Moving forward to the latency results between the *API* and the *database* in Table 10, it can be seen that there is a small increase between the *API* and *database*. In *Microsoft Azure*, securing the virtual network surrounding the *database* only restricts the location from which it can be accessed. Thus, the authentication procedure only changes to use *Entra ID* instead of a typical log-in. Moreover, since *Microsoft Azure* most likely tries to optimize the service stack, the *Entra ID* and *database* are fairly close to each other and thus the authentication latency is almost negligible.

Table 10: Latency metrics for *DB* queries.

(a) Latency for *DB* queries in a non *ZT* environment.

Payload Size	Average non ZT DB Latency
1	2.694ms
8	1.686ms
64	1.535ms
512	3.740ms
4096	21.766ms

(b) Latency for *DB* queries in a *ZT* environment.

Payload Size	Average ZT DB Latency
1	5.723ms
8	4.821ms
64	5.324ms
512	7.033ms
4096	24.312ms

(c) Latency difference between *ZT* and non *ZT* solutions for *DB* queries.

Payload Size	Average DB Latency Difference
1	3.029ms
8	3.135ms
64	3.789ms
512	3.293ms
4096	2.546ms

6 Discussion

In Section 3, the differences in theory between *AWS* and *Microsoft Azure* were compared in multiple areas of importance. As the services provided by *AWS* and *Microsoft Azure* are closed source enterprise services, there are quite a few secrets about how they actually work in practice. However, from the information given, the architecture needed to adapt to *ZTA* is mostly the same. The main difference between the two providers is the tools they offer, where each service covers different subsets of the principles of *ZT*. Thus, a different combination of services is needed depending on which provider you choose for a system.

The coverage of *ZT* is another important topic to discuss. From the previous information presented in Section 2.5.1, Section 2.5.2 and Section 3 it is summarized that both solutions cover the principles of *ZTA*. With this information, it makes it possible, regardless of the chosen solution, to develop and configure secure services. However, it might also be of interest to look at it from a business perspective, where the companies behind the solutions want to sell their product. Thus, it might be that they claim coverage of all vulnerabilities while it might not cover all of them. This is a problem with closed source enterprise solutions where only the people developing the solutions know the real coverage. Therefore, there might be reasons for each solution that make it preferable over another that customers do not know about.

Moving to *ZTA* is not done overnight. It is not something that can be inserted into most existing systems without neglecting some security features. That is why *ZTA* needs to be a central core of the original plan. However, there are some cases where systems have been living longer than *ZTA* have been around, or that systems are too vendor locked to a specific provider to be migrated. In these cases, it will be much easier for developers and architects to maintain the current provider regardless of the improvement that would come from changing provider.

Moving on to discussing the practical performance evaluation made in Section 5. The first and most important topic to discuss is if there is an increase in latency when applying *ZT*? In Section 5, it is clear that there is an increase in latency for both solutions. Thus, applying *ZT* measures increases the latency of calls to be made to a resource.

Secondly, how large is this latency increase? This question is a bit harder to answer from the results in Section 5. This is since *AWS* results in an increase of approximately 8.24 milliseconds while *Microsoft Azure* an increase of approximately 111.18 milliseconds on average. Both values indicate an increase, but at very different sizes. At first glance, this increase can be interpreted as the Azure solution affects performance in a larger way. However, that is most likely not the case. It is not just the implementation of zero trust that affects latency, but also some other factors. There is also the factor of geographical placement of the services. During the configuration of services, providers allow the specification of the region and deployment availability zone. However, availability zones may include multiple data centres. This then results in a natural latency overhead that differs for each configuration and cannot be avoided. Thus, leading to an unknown static overhead that cannot be measured. In addition, the hardware specifications of the different services vary between the solutions where the *AWS* services supply the services with 2 *vCPU* cores instead of the 1 *vCPU* core that *Microsoft Azure* provide. This small change is also an important factor in the reason why the latency for *Microsoft Azure* is greater than for *AWS*.

Another interesting factor to discuss is the static latency overhead regardless of the payload size. This can be seen in both Table 6 and Table 9 where the average latency overhead is about the same regardless of the payload size. This behaviour is desirable because the solution must be viable regardless of the transfer size.

Inspecting the percentiles in Figure 18 and Figure 19 it can be seen that the percentiles are proportional for the 50th, 75th, and 95th percentiles regardless of the addition of *ZT*. However, the 99th percentiles are the most affected by the addiction of *ZT*, where the increase is much greater than the other results. This can especially be seen for *Microsoft Azure* in Figure 19b where the 99th percentile can be described as randomized between payload sizes. This behaviour can be caused by two factors, either an unpredictable cold start for the first request or a temporary increase in latency between two resources in the network. Regardless of the cause, this is an increase in latency caused by *ZTA* for where the user may notice an increase in latency in a few cases.

The utilization of resources is another aspect that can be discussed. The payloads are quite small and will thus not increase the overall memory usage too much. That is why the results presented in Section 5 only have a small increase where they are in terms of only a few percentiles of maximum memory resources. On the other hand, the *CPU* resource usage change depending on the payload and solution. Observing the

CPU usage in both cases where *ZT* is not applied, it shows a clear trend that the usage is proportional to the size of the payload. On the other hand, for the two *ZT* implementations, there is no trivial trend that can be seen. There can be two reasons for this behaviour, where the gathering of metrics and request frequency play an important role. The metrics are calculated as an average usage over 15 seconds in *Prometheus*. Thus, the frequency of requests plays an important role in the calculation of resource usage. As stated above, the latency of each request is increased when *ZT* is applied. Therefore, when the requests are sent in sequence during the tests, fewer requests are delivered in each 15 second window. Thus, making the *CPU* usage lower than without applying *ZT*. In theory, resource utilization should be higher when more services are required, and more calculations need to be done for each request. But this is not the case where requests are made in sequence.

The measurement of network latency between the *API* and the *database* are done in Table 7 and Table 10. From these tables, it can be seen that there is a small increase of a few milliseconds in the network latency to access the *database*. This increase comes from outsourcing the authentication of requests to an outside resource that is not the *database*. Moreover, this makes latency dependent on the location of each of the resources, where a further distance between the *database* and authentication service would increase latency.

7 Conclusion

Zero trust as a solution to cybersecurity in cloud native environments is here to stay. Observing the enterprise solutions of *AWS* and *Microsoft Azure*, there is a clear need for solutions that compete and strive to be more secure. Comparison of the two solutions in theory through documentation and other published articles clearly shows similarities between the two solutions. However, there are some differences in how they are implemented and how services cover the subset of vulnerabilities *zero trust* aims to solve. As they offer similar tools and services to comply with *zero trust*, the two solutions can be argued to be similar even though they are developed and maintained by two different companies. Thus, answering **RQ1** where they are very similar in being two closed source enterprise solutions.

Comparison of performance between *zero trust* solutions offered by *AWS* and *Microsoft Azure* shows a clear trend of increased network latency between services. However, defining the size of the increase can not be done, since there are too many factors that are obscured by the user. Instead, the results presented in this work show that the latency overhead is static regardless of the payload size.

As the payload latency can not be properly defined for each of the solutions, to estimate when each solution can be preferred needs to be done using a theoretical approach. Therefore, a decision should be made based on the previous experience of developers and system maintainers. Decreasing the amount of time needed to solve a task as a developer outweighs the small performance difference between the two solutions presented in this thesis. In addition, other maintained systems also have a great impact on the choice, where keeping everything in the same domain will decrease context switches of developers and maintainability. This answers both **RQ2** and **RQ3** in which the preferred solution and the performance difference must be looked at. Resulting in an answer where neither of the solutions is preferred in terms of performance as it can not be determined other than a static increase. Instead, the decision is made based on previous experience.

8 Future Work

Section 3.6 introduced *zero trust* principles coverage where it described how previous literature have made evaluation of *Azure ZTA* coverage. It also states that there is no literature that performs the same evaluation for *AWS*. Therefore, this is an area that can be further explored to gain an understanding of how well *AWS* solution for *ZTA* cover the principles introduces in Section 2.4.

This thesis explored two of the enterprise solutions that are available at the time of writing. However, there are many more solutions out there that are both open source and enterprises. Thus, the work that has been done in this paper could be expanded to include more solutions. Furthermore, this opens up the question of cost versus service, where there may exist cases where hosting open source solutions are more cost-efficient in regard to security than purchasing services from providers.

Another question to explore is how secure the different solutions are. Dakić states that there are gaps between *ZTA* in theory and practice [8]. So, how does this compare between solutions? Could a *zero trust* test score be evaluated by executing a set of tests that verifies the compliance of *zero trust* in a system?

References

[1] Gary Adkins. "Red Teaming the Red Team: Utilizing Cyber Espionage to Combat Terrorism". In: *Journal of Strategic Security* 6.3 (2013), pp. 1–9. ISSN: 19440464, 19440472. URL: `http://www.jstor.org/stable/26485051` (visited on 05/02/2025).

[2] National Security Agency. *Advancing Zero Trust Maturity Throughout the Device Pillar*. Tech. rep. National Security Agency, 2023.

[3] Amazon. *Zero trust on AWS*. `https://aws.amazon.com/security/zero-trust/`. Accessed: 2025-02-10.

[4] David FC Brewer and Michael J Nash. "The Chinese Wall Security Policy." In: *S&P*. 1989, pp. 206–214.

[5] Jeffrey Chilberto, Sjoukje Zaal, Gaurav Aroraa, and Ed Price. "Identity Security with Azure Active Directory". In: *Cloud Debugging and Profiling in Microsoft Azure: Application Performance Management in the Cloud*. Berkeley, CA: Apress, 2020, pp. 215–234. ISBN: 978-1-4842-5437-0. DOI: `10.1007/978-1-4842-5437-0_7`.

[6] Marshall Copeland and Matthew Jacobs. "Azure Network Security Configuration". In: *Cyber Security on Azure: An IT Professional's Guide to Microsoft Azure Security*. Berkeley, CA: Apress, 2021, pp. 37–81. ISBN: 978-1-4842-6531-4. DOI: `10.1007/978-1-4842-6531-4_2`.

[7] Vedran Dakic, Zlatan Moric, Ana Kapulica, and Damir Regvart. "Analysis of Azure Zero Trust Architecture Implementation for Mid-size Organizations". In: (2024).

[8] Vedran Dakić, Zlatan Morić, Ana Kapulica, and Damir Regvart. "Analysis of Azure Zero Trust Architecture Implementation for Mid-Size Organizations". In: *Journal of Cybersecurity and Privacy* 5.1 (2025). ISSN: 2624-800X. DOI: `10.3390/jcp5010002`. URL: `https://www.mdpi.com/2624-800X/5/1/2`.

[9] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. "The Science DMZ: a network design pattern for data-intensive science". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: Association for Computing Machinery, 2013. ISBN: 9781450323789. DOI: `10.1145/2503210.2503245`.

[10] Yuri Diogenes and Tom Janetscheck. *Microsoft Azure Security Center*. Microsoft Press, 2021.

[11] David Drummond. *A new approach to China*. `https://googleblog.blogspot.com/2010/01/new-approach-to-china.html`. 2010.

[12] Saad Fehis, Omar Nouali, and Mohand-Tahar Kechadi. "A new Distributed Chinese Wall Security Policy Model". In: *Journal of Digital Forensics, Security and Law* 11.4 (2016), p. 11.

[13] Luca Ferretti, Federico Magnanini, Mauro Andreolini, and Michele Colajanni. "Survivable zero trust for cloud computing environments". In: *Computers & Security* 110 (2021), p. 102419. ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2021.102419`. URL: `https://www.sciencedirect.com/science/article/pii/S0167404821002431`.

[14] Chenquan Gan, Qingdong Feng, Xulong Zhang, Zufan Zhang, and Qingyi Zhu. "Dynamical propagation model of malware for cloud computing security". In: *IEEE Access* 8 (2020), pp. 20325–20333.

[15] Dennis Gannon, Roger Barga, and Neel Sundaresan. "Cloud-Native Applications". In: *IEEE Cloud Computing* 4.5 (2017), pp. 16–21. DOI: `10.1109/MCC.2017.4250939`.

[16] Jason Garbis and Juanita Kilpillai. *Software-Defined Perimeter (SDP) Specification v2.0*. Tech. rep. Cloud Security Alliance, 2022.

[17] Yuanhang He, Daochao Huang, Lei Chen, Yi Ni, and Xiangjie Ma. "A Survey on Zero Trust Architecture: Challenges and Future Trends". In: *Wireless Communications and Mobile Computing* 2022.1 (2022), p. 6476274.

[18] Hai Jin, Shadi Ibrahim, Tim Bell, Wei Gao, Dachuan Huang, and Song Wu. "Cloud Types and Services". In: *Handbook of Cloud Computing*. Ed. by Borko Furht and Armando Escalante. Boston, MA: Springer US, 2010, pp. 335–355. ISBN: 978-1-4419-6524-0. DOI: 10.1007/978-1-4419-6524-0_14.

[19] Joseph Migga Kizza, Wheeler Kizza, and Wheeler. "Guide to computer network security". In: 8 (2013).

[20] Office of Management and Budget. *Moving the U.S. Government Toward Zero Trust Cybersecurity Principles*. OMB Memo 22-09. 2022.

[21] Stephen Paul Marsh. "Formalising trust as a computational concept". In: (1994).

[22] Neerja Mhaskar, Mohammed Alabbad, and Ridha Khedri. "A formal approach to network segmentation". In: *Computers & Security* 103 (2021), p. 102162.

[23] Shivaji P Mirashe and Namdeo V Kalyankar. "Cloud computing". In: *arXiv preprint arXiv:1003.4074* (2010).

[24] Simone Rodigari, Donna O'Shea, Pat McCarthy, Martin McCarry, and Sean McSweeney. "Performance Analysis of Zero-Trust multi-cloud". In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. 2021, pp. 730–732.

[25] Jayant Sharma. *Microsoft Azure Security Technologies (AZ-500) - A certification guide: Get qualified to secure azure ad, network, compute, storage and data services through Security Center, Sentinel and other Azure Security Best Practices*. BPB Publications, 2022.

[26] V Stafford. "Zero trust architecture". In: *NIST special publication* 800 (2020), p. 207.

[27] Vladimir Stantchev. "Performance evaluation of cloud computing offerings". In: *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*. IEEE. 2009, pp. 187–192.

[28] Lionel Sujay Vailshery. *Global IaaS and PaaS hyperscaler market share 2020-2023*. Tech. rep. Statista, 2024.

[29] Romans Vanickis, Paul Jacob, Sohelia Dehghanzadeh, and Brian Lee. "Access Control Policy Enforcement for Zero-Trust-Networking". In: *2018 29th Irish Signals and Systems Conference (ISSC)*. 2018, pp. 1–6. DOI: 10.1109/ISSC.2018.8585365.

[30] Rory Ward and Betsy Beyer. "Beyondcorp: A new approach to enterprise security". In: *the magazine of USENIX & SAGE* 39.6 (2014), pp. 6–11.

[31] AWS Yuri Duchovny. *Steps towards a Zero Trust architecture on AWS*. https://aws.amazon.com/awstv/watch/f11edeb1c53/. 2024.