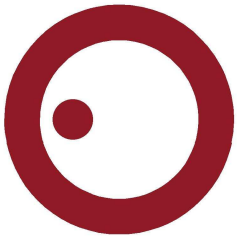


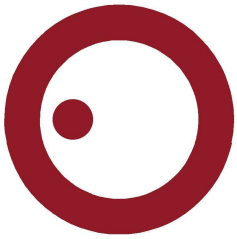
escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

IMPLEMENTATION AND ANALYSIS OF A ZTNA SOLUTION

BERNARDO DUARTE RODRIGUES FRAGOSO DE RHODES

Leiria, Setembro de 2025



escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

IMPLEMENTATION AND ANALYSIS OF A ZTNA SOLUTION

BERNARDO DUARTE RODRIGUES FRAGOSO DE RHODES
Número: 2230473

Projeto realizado sob orientação do Professor Doutor Paulo Jorge Gonçalves Loureiro (paulo.loureiro@ipleiria.pt) e do Professor Doutor Sílvio Priem Mendes (smendes@ipleiria.pt).

Leiria, Setembro de 2025

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my supervisors, Paulo Jorge Gonçalves Loureiro and Sílvio Priem Mendes, for their guidance, knowledge sharing and support throughout this work.

I am also grateful to all those who, directly or indirectly, shared their knowledge and supported me during this stage.

Special recognition goes to my family and friends for their constant encouragement, patience, and unconditional support, which were fundamental to the completion of this journey.

RESUMO

À medida que as ameaças cibernéticas continuam a evoluir, os mecanismos tradicionais de segurança baseados em perímetro, como VPNs IPsec e SSL, tornam-se cada vez mais ineficazes contra ataques baseados em identidade e movimentos laterais em ambientes distribuídos. Ao conceder um acesso amplo ao nível da rede após a autenticação, as VPNs ampliam a superfície de ataque e falham em aplicar o princípio de Zero Trust — "nunca confiar, verificar sempre". Para colmatar esta lacuna, esta dissertação investiga a viabilidade de implementar uma solução de Zero Trust Network Access (ZTNA) em contexto de pequena e média empresa (PME), utilizando o OpenZiti, uma plataforma open-source.

A investigação combina uma revisão da literatura com uma análise comparativa de soluções ZTNA/SASE, posicionando o OpenZiti face a alternativas de mercado. Foi então concebida e implementada uma arquitetura prática num ambiente semelhante a uma PME, com dois sites interligados e um controlador em Azure, simulando uma topologia híbrida. A implementação incluiu o provisionamento de controlador e edge routers, integração Okta–Active Directory e definições granulares de serviços através de configurações de intercept, host e políticas de acesso.

Para além de reproduzir modelos existentes, o trabalho fornece uma avaliação prática e original das capacidades e limitações do OpenZiti, contribuindo para o estado da arte ao evidenciar lacunas nas soluções open-source de ZTNA.

Os resultados demonstram que o OpenZiti impõe acesso rigoroso ao nível do serviço, reduzindo significativamente as oportunidades de movimento lateral em comparação com as VPNs. A revogação de identidades e a propagação de políticas ocorreram em tempo quase real, e a descoberta não autorizada de serviços foi impedida. Foram identificadas limitações ao nível da segmentação leste–oeste, do suporte a federação de identidades e da integração nativa de registos, que requerem controlos complementares ou ferramentas externas. Ainda assim, o estudo confirma que soluções open-source de ZTNA podem fornecer às PMEs uma alternativa economicamente viável e tecnicamente robusta às VPNs, proporcionando melhorias mensuráveis na postura de segurança e apoiando a transição de modelos baseados em perímetro para Zero Trust em organizações com recursos limitados.

ABSTRACT

As cyber threats continue to evolve, traditional perimeter-based security mechanisms, such as IPsec and SSL VPNs, are increasingly ineffective against identity-based and lateral movement attacks in distributed environments. By granting broad, network-level access once authenticated, VPNs expand the attack surface and fail to enforce the Zero Trust principle of "never trust, always verify." To address this gap, this thesis investigates the feasibility of deploying a Zero Trust Network Access (ZTNA) solution in a small-to-medium enterprise (SME) context using OpenZiti, an open-source platform.

The research combines a literature review with a comparative analysis of ZTNA/SASE solutions, positioning OpenZiti against market alternatives. A practical architecture was then designed and implemented in an SME-like environment, consisting of two interconnected sites and an Azure-based controller, simulating a hybrid environment.

Beyond reproducing existing models, the work provides an original, hands-on evaluation of OpenZiti's capabilities and limitations. It contributes to the state of the art by highlighting gaps in open-source ZTNA solutions. The implementation was validated using a structured test suite covering authentication enforcement, encrypted communication, identity revocation, and service visibility.

The results demonstrate that OpenZiti enforces fine-grained, service-level access while significantly reducing lateral movement opportunities compared to VPNs. Identity revocation and policy updates propagated in near real time, and unauthorised service discovery was prevented. Limitations were identified in areas such as east–west segmentation, federated identity support, and native logging integration, which require complementary controls or third-party tools. Nevertheless, the study confirms that open-source ZTNA solutions can provide SMEs with a cost-effective and technically viable alternative to VPNs, delivering measurable improvements in security posture and supporting the broader transition from perimeter-based security to Zero Trust in resource-constrained organisations.

INDEX

Appendices

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

(s)FTP	(secure) File Transfer Protocol.
(v)CPU	(virtual) Central Processing Unit.
AD	Active Directory.
API	Application Programming Interface.
BYOD	Bring Your Own Device.
CASB	Cloud Access Security Broker.
DBIR	Data Breach Investigations Report.
DC	Domain Controller.
DLP	Data Loss Prevention.
DNS	Domain Name System.
FQDN	Fully Qualified Domain Name.
GUI	Graphical User Interface.
HQ	Headquarters.
HTTP(s)	HyperText Transfer Protocol (secure).
IAM	Identity and Access Management.
IGA	Identity Governance and Administration.
JIT	Just-in-Time.

JSON	JavaScript Object Notation.
JWKS	JSON Web Key Set.
JWT	JSON Web Token.
LAN	Local Area Network.
LTS	Long Time Support.
MFA	Multi-Factor Authentication.
NGFW	Next-Generation Firewall.
NIST	National Institute of Standards and Technology.
NSG	Network Security Group.
OIDC	OpenID Connect.
OZC	OpenZiti Controller.
OZER	OpenZiti Edge Router.
PKI	Public Key Infrastructure.
POP	Point of Presence.
REST	Representational State Transfer.
SAML	Security Assertion Markup Language.
SASE	Secure Access Service Edge.
SDK	Software Development Kit.
SDN	Software Defined Networking.
SDP	Software Defined Perimeter.
SIEM	Security Information and Event Management.
SME	Small to Medium Enterprise.
SOC	Security Operations Center.

SSL Secure Sockets Layer.

SSO Single Sign-On.

SWG Secure Web Gateway.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UDP User Datagram Protocol.

UEBA User and Entity Behavior Analytics.

VM Virtual Machine.

VPN Virtual Private Network.

WAN Wide Area Network.

WS Workstation.

ZAC Ziti Administration Console.

ZT Zero Trust.

ZTA Zero Trust Architecture.

ZTNA Zero Trust Network Access.

1

INTRODUCTION

1.1 CONTEXT AND MOTIVATION

The rapid evolution of the digital landscape, driven by ubiquitous cloud adoption, edge computing, and the normalisation of remote and hybrid work models, has fundamentally reshaped organisational IT infrastructures. While these developments unlock unprecedented flexibility and scalability, they also render traditional perimeter-centric security models increasingly inadequate. Virtual Private Networks (VPNs), once considered the cornerstone of secure remote access, exhibit critical weaknesses against modern attacker tactics, techniques, and procedures (TTPs), including credential stuffing, lateral movement, and supply-chain compromise. Recent industry analyses report that insider-enabled breaches and misconfigurations accounted for approximately 83% of cybersecurity incidents in 2024 Verizon Business, 2024, underscoring the need for adaptive, context-aware security frameworks.

1.2 PROBLEM STATEMENT

Traditional Virtual Private Network (VPN) architectures remain the predominant remote access solution for small and medium-sized enterprises (SMEs) due to their affordability and ease of deployment. However, these solutions inherently rely on implicit trust models: once authenticated, users are granted broad network-level access, often extending far beyond the resources they require. This "all-or-nothing" approach not only increases the attack surface but also facilitates lateral movement, complicates forensic investigations, and undermines the principle of least privilege.

While emerging paradigms such as Zero Trust Network Access (ZTNA) and Secure Access Service Edge (SASE) address these shortcomings by enforcing identity-aware, continuous, and granular access controls, their adoption faces significant barriers in the SME context. Commercial ZTNA and SASE platforms are often subscription-based, resource-intensive, and designed for enterprises with dedicated security teams, making them economically and operationally inaccessible for SMEs. At

the same time, the landscape of open-source alternatives remains fragmented and underexplored, with many solutions claiming Zero Trust alignment but in practice failing to provide service-level segmentation or seamless integration with identity providers.

This gap between the security needs of SMEs and the availability of cost-effective, practical ZTNA solutions defines the core problem of this research. Specifically, there is insufficient empirical evidence evaluating whether open-source ZTNA platforms can provide SMEs with a viable and secure alternative to traditional VPN-based access models.

1.3 RESEARCH OBJECTIVES

The overarching objective of this research is to evaluate whether an open-source Zero Trust Network Access (ZTNA) solution can serve as a viable alternative to VPN-based architectures in small and medium-sized enterprises (SMEs). To address this, the study pursues the following specific objectives:

1. **Comparative Analysis:** Examine commercial and open-source ZTNA/SASE solutions, identifying their core features, deployment models, and alignment with Zero Trust principles, with particular attention to their suitability for SMEs.
2. **Architecture Design:** Develop a reference architecture for deploying an open-source ZTNA platform that accounts for the operational and budgetary constraints typical of SMEs.
3. **Practical Implementation:** Deploy an open-source ZTNA solution (OpenZiti) in a simulated SME environment, integrating cloud and on-premises components, and configuring identity management with Active Directory and Okta.
4. **Empirical Evaluation:** Assess the deployed solution through structured functional and security tests, focusing on service-level access enforcement, resistance to lateral movement, identity management limitations, and operational complexity.
5. **Critical Appraisal:** Analyse the benefits and shortcomings of the implementation, highlighting the extent to which open-source ZTNA solutions can realistically improve SME security posture compared to traditional VPNs, and identify areas requiring complementary controls or future research.

1.4 METHODOLOGICAL APPROACH

The research adopts a mixed-method strategy combining:

- A systematic literature review spanning peer-reviewed scholarship, standards documents (e.g., NIST SP 800-207), and industry white papers;
- A comparative feature and vendor matrix evaluating five leading ZTNA/SASE platforms;
- A hands-on deployment in a two-site SME testbed augmented by an Azure cloud environment, validated through a structured test suite comprising functional, security, and performance scenarios;
- Qualitative analysis of deployment effort, operational visibility, and forensic artefacts.

1.5 DOCUMENT STRUCTURE

The remainder of the document is organised as follows:

- **Chapter 2** reviews the state of the art in Zero Trust Network Access and Secure Access Service Edge architectures, identifying existing gaps in literature and practice, and provides a detailed contextualisation of ZTNA concepts, including comparisons with traditional VPNs and guidance from NIST;
- **Chapter 3** presents a comparative analysis of selected ZTNA and SASE solutions, followed by the proposed architecture and practical implementation using OpenZiti;
- **Chapter 4** describes the functional and security tests carried out to validate the deployment, with corresponding results;
- **Chapter 5** summarises the findings and outlines the main conclusions, limitations, and future research directions.

By the end of this thesis, readers will have a replicable framework and evidence-based guidance for transitioning from VPN to Zero Trust architectures in resource-constrained environments.

2

RELATED WORK

This section reviews the theoretical and practical foundations of Zero Trust Network Access (ZTNA) and Secure Access Service Edge (SASE), with a focus on their technical evolution, deployment challenges, and relevance to modern enterprise environments. It concludes with a contextualisation of core ZTNA concepts and a comparative framing of traditional VPN-based models.

2.1 EVOLUTION OF ENTERPRISE NETWORK SECURITY

Enterprise network security has traditionally evolved through perimeter-based approaches, beginning with firewalls and site-to-site VPNs in the late 1990s and gradually incorporating intrusion prevention systems, next-generation firewalls (NGFWs), and identity-aware controls. In large organizations, these shifts were often supported by dedicated security teams, compliance drivers, and the ability to invest in layered defenses. For small and medium-sized enterprises (SMEs), however, adoption followed a markedly different trajectory. Limited budgets and staffing constraints often meant SMEs adopted VPNs as the de facto remote access solution, prioritizing affordability and simplicity over layered security. While this approach provided basic confidentiality and remote connectivity, it also reinforced a "castle-and-moat" security mindset, in which once inside the perimeter, users gained broad access with minimal internal segmentation. Unlike enterprises, SMEs have historically lacked the resources to deploy advanced identity management, endpoint monitoring, or micro-segmentation, leaving them disproportionately exposed to credential compromise and lateral movement. The emergence of cloud computing, mobility, and hybrid work has exacerbated these disparities. Large enterprises increasingly adopt Zero Trust principles to handle distributed environments, while SMEs remain dependent on aging VPN infrastructures due to the cost and complexity of commercial Zero Trust solutions. This divergence underscores the importance of evaluating whether open-source Zero Trust Network Access (ZTNA) platforms can offer SMEs a practical migration path beyond perimeter-based security. Figure 1

summarises key milestones from perimeter firewalls in the late 1990s to contemporary Zero Trust architectures.

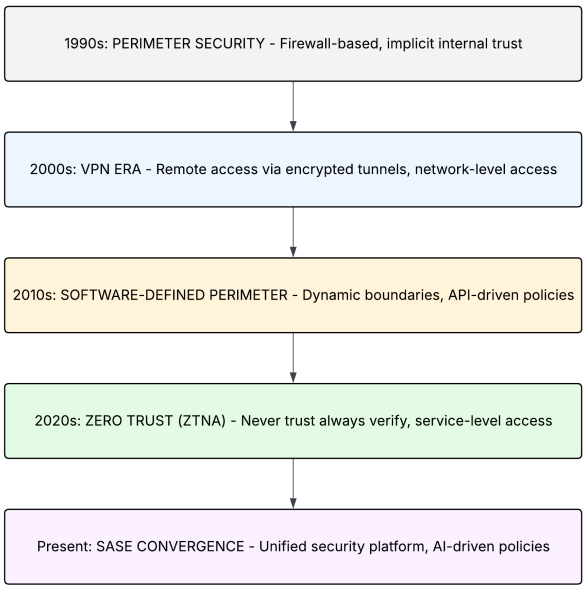


Figure 1: Simplified timeline of enterprise network security paradigms.

2.2 ZERO TRUST PRINCIPLES AND ARCHITECTURES

2.2.1 *Core Tenets*

Zero Trust (ZT) reframes security around identity, device posture, and continuous verification rather than implicit network location trust. NIST SP 800–207 formalises ZT as "a collection of concepts and ideas designed to minimise uncertainty" in enforcing access decisions (Rose et al., 2020). Key tenets include:[1]

- **Assume breach**: design for compromise persistence.
- **Verify explicitly**: authenticate and authorise every request.
- **Least privilege**: restrict access to the minimum required.
- **Micro-segmentation**: isolate workloads to limit blast radius.
- **Continuous monitoring**: adapt policies based on real-time context.

2.2.2 *Reference Models*

NIST outlines three implementation models: Identity Governance/Administration (IGA), Micro-segmentation, and Software-Defined Perimeter (SDP). Industry vendors expand on these with Unified Endpoint Management (UEM), User and Entity Behaviour Analytics (UEBA), and Secure Service Edge enforcement layers (Bradatsch et al., 2024; Colomo-Palacios et al., 2021).

2.3 ZERO TRUST NETWORK ACCESS (ZTNA)

Zero Trust Network Access (ZTNA) is founded on the principle of "Never trust, always verify" (Kang et al., 2023), where access is continuously authenticated and authorised regardless of network origin. Kang et al., 2023 provide a theoretical model of ZTNA based on continuous monitoring and segmentation, employing a simulation-based methodology using NS-3 network simulator with 100-node topologies. However, their approach exhibits several limitations:

1. the simulation environment fails to capture real-world authentication latencies and IdP integration complexities,

1 Paraphrased from Rose et al., 2020.

2. their policy enforcement model assumes homogeneous device types, ignoring Bring Your Own Device (BYOD) scenarios common in SMEs,
3. no cost-benefit analysis is provided for resource-constrained organizations.

Syed et al., 2022 review the growing importance of Zero Trust models in the context of rising cyber threats, offering the first comprehensive mapping of authentication mechanisms to ZTA maturity levels and comparing 18 solutions. However, their analysis remains largely theoretical, with performance benchmarks limited to 1,000-node simulations and no validation in production environments. More technical depth is provided by Bradatsch et al., 2024, who introduce a dynamic trust-score-based access model, addressing practical concerns through their Subjective Logic-based trust computation framework and achieving sub-second response times with 29 trust attributes. Yet, the computational complexity and hardware requirements create barriers for organisations with fewer resources, such as SMEs. Their proposal reflects ZTNA principles in practice, though it lacks broader application to distributed systems or resource-constrained organisations.

2.4 SECURE ACCESS SERVICE EDGE (SASE)

Colomo-Palacios et al., 2021 explore SASE as a cloud-native security framework that consolidates networking and security functions. SASE's advantages — including scalability, visibility, and enforcement — are widely acknowledged, yet few works evaluate its practical deployment. Yiliyaer and Kim, 2022 propose a conceptual model that merges Zero Trust with SASE components, such as SD-WAN and SSL inspection. However, they do not validate their approach through empirical testing.

2.5 PRACTICAL IMPLEMENTATIONS OF ZTNA AND SASE

Despite the growing volume of academic and industry literature on Zero Trust and SASE, most contributions remain conceptual, simulation-based, or limited to theoretical frameworks. For instance, Bradatsch et al., 2024 introduce a scorebased trust model, yet their work stops at simulation and does not address deployment feasibility in production networks.

Similarly, Syed et al., 2022 provide extensive taxonomies of authentication mechanisms but without empirical validation in organizational contexts. This lack of applied research is particularly evident in small and medium-sized enterprises

(SMEs). While large enterprises may conduct internal pilots or rely on vendor-driven proof-of-concept deployments, SMEs often lack both the resources and technical expertise to test or validate advanced security architectures. As a result, the practical implications of adopting ZTNA remain underexplored for this segment, even though SMEs are among the most dependent on remote access and the most vulnerable to credential compromise and lateral movement.

This thesis positions itself as a response to that gap. By implementing and testing an open-source ZTNA solution (OpenZiti) in an SME-like environment, it provides empirical evidence on the feasibility, strengths, and limitations of such deployments. In doing so, it contributes not only to academic discourse, where empirical studies are scarce, but also to the practitioner community, by offering a replicable framework for SMEs that need cost-effective alternatives to traditional VPNs.

2.6 CONTEXTUALISATION OF ZTNA AND SASE CONCEPTS

To bridge the gap between theory and implementation, it is essential to contextualise the application of ZTNA and SASE within real network infrastructures.

2.6.1 *ZTNA fundamentals*

ZTNA shifts security from the network perimeter to the identity level. Unlike VPNs, which grant broad access to internal networks, ZTNA provides granular, service-level access based on policies tied to user identity, device, and session context (Sarkar et al., 2022). This model is especially effective in cloud-native, hybrid, and IoT environments. ZTNA is not a formal protocol or standard, but an approach endorsed by institutions such as NIST. Protocol coverage depends on the implementation; some tools are limited to ports 80/443, while others – like OpenZiti – support broader protocol ranges.

2.6.2 *Comparison: ZTNA vs VPNs*

ZTNA offers multiple advantages over VPNs:

- Minimised attack surface;
- Elimination of implicit trust;

- Identity-based access control;
- Resistance to lateral movement;
- Improved scalability.

These comparisons are summarised in Table 1, supporting the rationale for ZTNA adoption in this study.

Table 1: Comparison of VPN and ZTNA Characteristics

Criteria	**VPN**	**ZTNA**	**References/Notes**
Access Model	Network-wide	Service/Application-level	Palo Alto Networks, 2025; Rose et al., 2020
Trust Model	Implicit after login	Continuous, context-aware	Rose et al., 2020
Attack Surface	Broad (lateral movement)	Minimised (micro-segmented)	Kang et al., 2023; Sarkar et al., 2022 Effectiveness depends on implementation quality and service configuration. See Test T03.
Identity Source	Credentials/PKI	IdP + device posture	Identity source configured depends on the tool's integrations and features.
Deployment and Scalability	Gateway appliances, limited scalability	Distributed edge proxies, high scalability	Colomo-Palacios et al., 2021; Yiliyaer and Kim, 2022
Visibility	Limited session logs	Fine-grained per request	Bradatsch et al., 2024; Syed et al., 2022

2.6.3 *SASE vs Reverse Proxy*

Although SASE and reverse proxies share the function of mediating access, their approaches differ significantly. SASE applies continuous inspection, risk scoring, and session control across diverse network layers. Reverse proxies perform initial authentication but lack mechanisms for continuous session validation or adaptive policy enforcement based on context or risk scoring, making it unsuitable as a ZTNA substitute. Some use cases were chosen to clarify where each technology may be more effective. Use case examples:

1. Public API Gateway: Reverse proxies may be sufficient for rate limiting and basic authentication.
2. Legacy monolith access: SASE is required for deep packet inspection and context-aware policies.
3. Control over shared data: SASE can control whereas private data can be shared, and with whom, through DLP policies.

This distinction further justifies the architectural choices made later in this work.

2.7 OPEN-SOURCE ZTNA SOLUTIONS LANDSCAPE

While the market offers several open-source networking and security solutions often positioned as Zero Trust alternatives, critical analysis reveals that most fail to implement true ZTNA principles. This section examines commonly cited "ZTNA" open-source solutions and evaluates their actual alignment with Zero Trust architecture requirements as defined by Rose et al., 2020.

1. **Tailscale**
   Tailscale is built on WireGuard, provides identity-aware mesh networking but fundamentally remains a VPN solution granting network-level rather than service-level access Tailscale Inc., 2024. Its centralized coordination server and lack of true micro-segmentation disqualify it as a ZTNA solution.
2. **WireGuard**
   WireGuard is explicitly a Layer 3 VPN protocol, not a ZTNA solution Donenfeld, 2017. While projects like Firezone attempt to add Zero Trust features, they merely use WireGuard as the base layer while building entirely separate

identity and policy layers, confirming WireGuard's role as a networking/VPN protocol rather than a complete ZTNA platform.

3. **ZeroTier**
   ZeroTier creates virtual Layer 2 networks that span across the internet ZeroTier Inc., 2024. All devices connected to a ZeroTier network appear to be on the same local network segment ZeroTier Inc., 2024, which is antithetical to Zero Trust principles. By granting authenticated users access to entire network segments, it enables the lateral movement ZTNA specifically prevents, making it unsuitable to be classified as a ZTNA solution.

4. **Nebula**
   Nebula uses certificate-based authentication for mesh networking but lacks the continuous verification and dynamic policy enforcement essential to ZTNA. Its static trust model and network-level access paradigm position it as an enhanced VPN rather than true Zero Trust architecture Defined Networking Corp., 2024.

2.8 SUMMARY

This chapter analysed prior work on enterprise security evolution, Zero Trust principles, ZTNA and SASE frameworks, and their practical implementations. It compared ZTNA to VPNs, highlighted differences between SASE and reverse proxies, and assessed open-source tools such as Tailscale, WireGuard, ZeroTier, and Nebula, exposing gaps in their alignment with Zero Trust requirements.

The next chapter, Development, transitions from theory to practice by presenting the rationale for tool selection, the proposed ZTNA architecture for SMEs, and the detailed implementation of OpenZiti, including topology, provisioning, identity management, IdP (Okta) integration, and service/policy configuration.

3

DEVELOPMENT

This chapter translates the Zero Trust concepts discussed in Chapter 2 into a concrete deployment that replaces a legacy VPN architecture in a small-to-medium enterprise (SME) setting. We begin with the tool-selection rationale, then describe the target environment, the proposed ZTNA architecture using *OpenZiti*, and the step-by-step deployment procedure, culminating in the functional configuration artefacts used for evaluation.

3.1 SOLUTION SELECTION RATIONALE

Table 2 compares the publicly documented capabilities of five vendors: **Zscaler ZPA**, **Netskope**, **Prisma Access**, **Cisco Umbrella**, and the open-source **OpenZiti** platform. Commercial offerings combine Secure Web Gateway (SWG) and Cloud Access Security Broker (CASB) functions with ZTNA, whereas OpenZiti focuses exclusively on identity-centred micro-segmentation without inline inspection.

Table 2: High-Level Feature Matrix for Referenced Vendors

Capability	**Zscaler ZPA**	**Netskope**	**Prisma Access**	**Cisco Umbrella**	**OpenZiti**
Licensing model	Subscription (per-user)	Subscription (per-user)	Subscription (per-user)	Subscription (per-user)	Self-host
Delivery model	150 + POPs	80 + POPs	100 + POPs	70 + POPs	Self-host VM/Container
Micro-segmentation	Yes	Yes	Yes	VLANapp	Yes
Inline inspection (SWG/DLP)	Yes	Yes	Yes	Yes	No
Indicative cost	High	High	High	High	Low (infrastructure)

3.1.1 *Vendor Summaries*

In this section, we conduct a comparative analysis of SASE suppliers, focusing on their ZTNA implementation capabilities, and ZTNA-only solutions. The solutions selected reflect market leaders, universally recognised ZTNA and SASE solutions, and open-source solutions that appear as alternatives to existing commercial solutions.

ZSCALER ZPA A mature cloud-native ZTNA service that provides granular access control, integrated SWG, CASB, and sandboxing. It publicly advertises extensive global POP coverage (150+ as of 2025) and reports < 50 ms 95th-percentile latency for US / EU users. Pricing is among the highest but includes threat intelligence and 24 / 7 SOC support.

NETSKOPE Renowned for deep visibility into SaaS and Shadow IT usage. Combines ZTNA with CASB and inline Data Loss Prevention (DLP). Deployment in hybrid networks can require extensive policy tuning, but it offers rich analytics dashboards and risk scoring.

PRISMA ACCESS (PALO ALTO NETWORKS) Integrates next-generation firewall (NGFW), SD-WAN, and ZTNA in a single service edge. Strong micro-segmentation and advanced threat prevention, yet complexity and licensing costs position it for enterprises with strict compliance regimes.

CISCO UMBRELLA SECURE ACCESS Delivers DNS-layer security, SWG, firewall-as-a-service, and CASB under a unified umbrella. Tightest integrations are achieved in all-Cisco environments; multi-vendor sites may need additional connectors. Provides robust global POP coverage but offers less policy granularity than dedicated ZTNA tools.

OPENZITI An open-source platform designed around identity-centric overlays. It enables service-level access without exposing resources on the public Internet. Lacks built-in DLP or UEBA, but its extensible API and permissive licence make it attractive for cost-sensitive organisations needing full deployment control.

3.2 ARCHITECTURE

This section presents the transformation from a traditional VPN-based model to a Zero Trust architecture implemented with OpenZiti.

3.2.1 *Traditional VPN-based Architecture*

The legacy network model built in this scenario, to represent a typical SME, relied heavily on site-to-site Virtual Private Network (VPN) connections to interconnect its two physical offices (Lisbon and Madrid), while also providing secure connectivity for remote users. In practice, this design enabled remote employees or branch offices to connect to the corporate LAN as if they were physically present within the headquarters. Once authentication was completed (typically being basic authentication, with MFA), the VPN tunnel granted broad access to internal network resources without granular restrictions. This model depicts a "castle-and-moat" approach to security: once a user passed through the "moat" (the VPN gateway), it gained implicit trust inside the "castle" (the internal network). This design introduced critical security limitations:

1. Broad attack surface: After authentication, users gain visibility into the entire internal subnet. Even if they only require access to a specific service (i.e. print server #1 or SAS #1), the VPN tunnel exposes them to unrelated servers and services. This violates the principle of least privilege and creates potential entry points for attackers exploiting compromised credentials Akamai Technologies, 2024.
2. Lateral movement risks: Once a remote user has access to the VPN, they can run discovery tools (i.e. Nmap) to probe internal services. This raises concerns about insider misuse or compromised credentials being leveraged to pivot across the environment.
3. Scalability issues: Limited scalability due to dependency on centralised VPN gateways, which can become a bottleneck.

Figure 2 illustrates an SME's previous VPN topology. The Lisbon office operated as the headquarters, hosting a print-server and establishing a VPN gateway with the Madrid office. The Madrid office connected through a permanent site-to-site IPSec tunnel, extending its entire subnet into the Lisbon network, and hosted all the key infrastructure. Remote employees established client-to-site VPN sessions into the

same gateway, gaining access to the Lisbon LAN as if physically present. All traffic was routed first through Lisbon, resulting in potential latency and bottlenecks. The figure therefore highlights the hub-and-spoke nature of the design, its dependency on a centralised gateway, and the lack of service-level segmentation, which ultimately motivated the migration towards ZTNA.

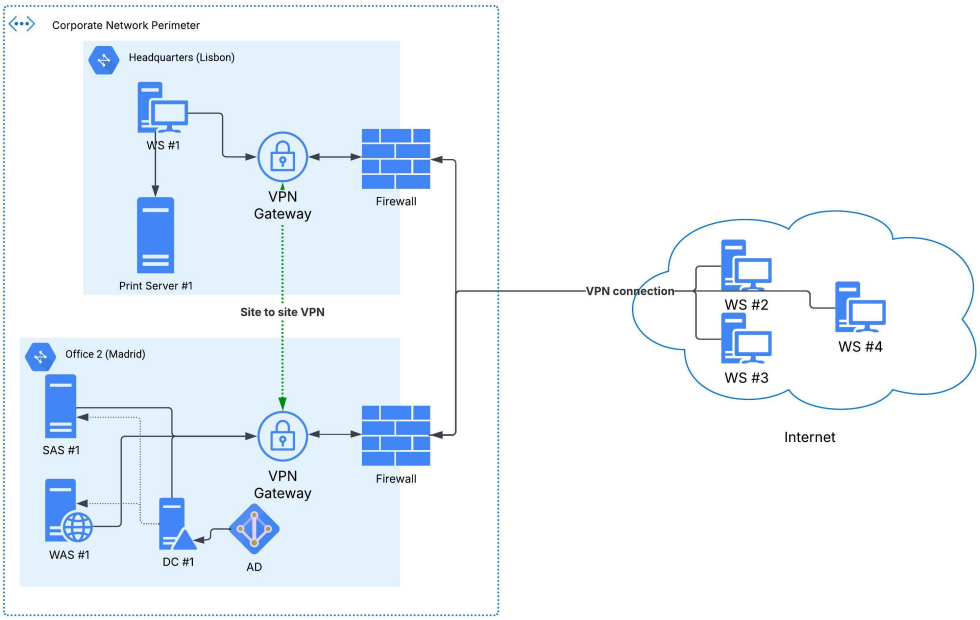


Figure 2: VPN-based Architecture

3.2.2 *ZTNA Architecture with OpenZiti*

In the new architecture, services such as the Web Application Server (WAS #1), Secure File Transfer Server (SAS #1), and Domain Controller (DC #1) were no longer exposed through broad subnet-level access. Instead, they were encapsulated behind OpenZiti Edge Routers (OZER #1 and OZER #2) deployed at the Lisbon and Madrid sites. These routers only processed traffic explicitly authorised by centrally defined identity policies, ensuring that every connection was tightly bound to a validated user and service. All traffic between endpoints and services was encapsulated within mTLS-based overlay tunnels, created by OpenZiti. Each tunnel relied on service-specific certificates issued by the OpenZiti Controller (OZC #1), which was hosted in Azure. This controller managed the issuance of identities, orchestration of policies, and routing of all service access. To strengthen identity assurance, Okta was integrated as the Identity Provider (IdP), synchronised with the on-premises Active Directory. Although OpenZiti does not support Just-in-Time

(JIT) provisioning, this integration ensured that remote access was tied to enterprise-managed user accounts, reducing administrative burden and strengthening security alignment with IAM practices. From a practical standpoint, this design addressed the limitations observed in the VPN model:

1. Granular access enforcement: Employees were only able to connect to the specific services they were authorised to use (i.e. an employee accessing WAS #1), rather than the entire subnet.
2. Reduced attack surface: No service endpoints were exposed to the public internet or accessible through a "flat" network tunnel.
3. Improved scalability: New sites or remote users could be added simply by deploying an additional edge router and issuing identities, without reconfiguring VPN gateways.

Figure 3 illustrates an SME's network after migrating to OpenZiti, showing how resources are now accessed through identity-aware overlay tunnels rather than through broad network exposure. Each on-premises site (Lisbon and Madrid) hosts an edge router that intercepts traffic to local services, while remote users connect through cloud-hosted routers. All communication flows are orchestrated by the Azure-based OpenZiti Controller (OZC #1), which acts as the central trust authority. Key services such as the WAS, SAS, and DC are shielded behind the routers, only reachable if a valid identity and policy permit access. Unlike the VPN model, where users joined the full corporate subnet, the ZTNA design enforces per-service, per-identity access, removing unnecessary visibility and reducing lateral movement risk. The figure highlights the shift from implicit trust to explicit, identity-bound trust.

3.3 IMPLEMENTATION DETAILS

This section describes in detail the implementation of the Zero Trust Network Access (ZTNA) solution using OpenZiti in a small-to-medium enterprise (SME)-like environment. The architecture was designed to replace the legacy VPN-based model (Figure 2) with an identity-centric approach (Figure 3), while maintaining operational feasibility for distributed offices and remote employees.

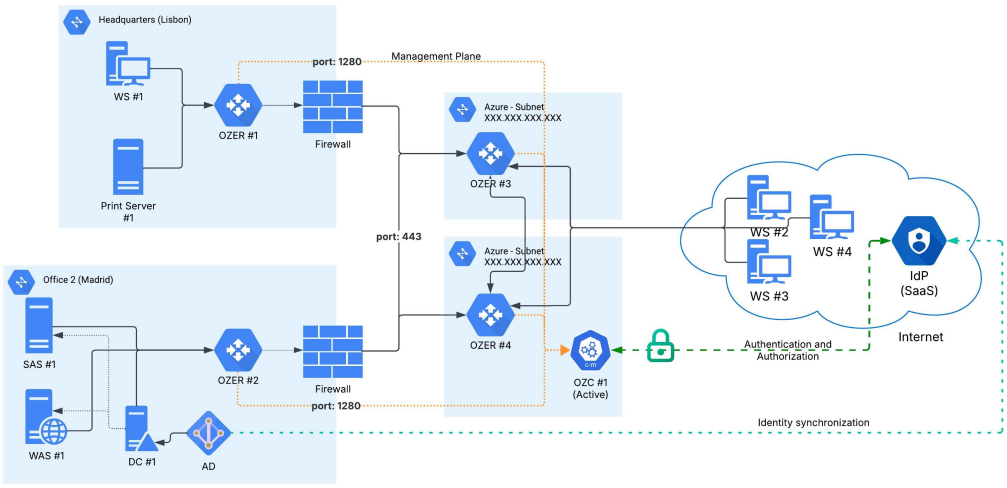


Figure 3: ZTNA Architecture, with OpenZiti

3.3.1 *Deployment Topology*

The deployment replicated an SME scenario with two physical sites (Lisbon and Madrid), one cloud environment (Microsoft Azure), and multiple remote endpoints:

- **Lisbon Site (HQ)**: Contained secondary infrastructure, such as a print server, and was a physical office where users could connect from, and is connected through a dedicated OpenZiti Edge Router (OZER #1). This router provided access to Madrid users and services, while ensuring all communication was identity-bound;
- **Madrid Site (Branch Office)**: Hosted key infrastructure components, including a domain controller (DC #1), a Web Application Server (WAS #1), and an SFTP server (SAS #1). An OpenZiti Edge Router (OZER #2) was deployed as the gateway for Madrid services;
- **Cloud (Azure)**: Hosted the OpenZiti Controller (OZC #1), and two OpenZiti Edge Routers (OZER #3 and OZER #4), to guarantee high availability. This decision was motivated by the current OpenZiti limitation of not supporting multiple active controllers for high availability (a feature still under development), and taking advantage of Cloud capabilities, such as on-demand resources, to ensure better performance. Hosting in Azure also ensured resilience and accessibility without relying on a single on-premises location.
- **Remote users**: Connected using the Ziti Desktop Edge Agent, which established overlay tunnels transparently once authenticated.

3.3.1.1 *Identity and Access Management*

Identity enforcement was central to the deployment:

- **Identity Provider**: Okta (SaaS) was chosen as the Identity Provider (IdP) to integrate with OpenZiti. It was integrated with the on-premises Active Directory to ensure all users that connected to OpenZiti's infrastructure were enterprised users, from a source of truth within the organisation.
- **Authentication**: Each remote user was issued a Ziti identity certificate (jwt token), signed by the controller. These certificates were bound to specific devices, preventing credential theft or token reuse.
- **Limitations**: Due to OpenZiti's lack of support for JIT provisioning, group membership in AD could not be automatically mapped to Ziti policies through Okta, using SAML assertions/OIDC claims. Instead, identities had to be managed manually within OpenZiti. This is a significant operational and security limitation, as it prevents fully centralised IAM.

3.3.2 *Components Provisioning*

3.3.2.1 *Inventory, Naming and Version baseline*

Assets and roles:

- OZC #1 - Azure VM hosting the OpenZiti Controller, on port 1280;
- OZER #1 - Lisbon on-prem VM, with an Edge Router, as LAN gateway, installed;
- OZER #2 - Madrid on-prem VM, with an Edge Router, as LAN gateway, installed;
- OZER #3 and OZER #4 - Cloud Edge Routers for remote access / resiliency.
- WAS #1 - Web Application Server, on port TCP 8443;
- SAS #1 - Secure File Transfer Server, on port TCP 22;
- DC #1 - Domain Controller, on ports TCP/UDP 53, 88, 123, 135, 138, 389, 443, 445, 636 (all ports should be evaluated, and limited, according to the organisation's requirements);
- Ziti desktop agent - Client agent for remote sessions;

- Okta (IdP) + AD - Identity source. Okta agent installed on DC #1 for synchronization purposes;
- Perimeter Firewalls.

Naming and DNS:

To ensure a near real scenario, a domain was obtained solely to be used in this implementation: oz-implementation.pt.

- Controller FQDN: ziti-controller.oz-implementation.pt;
- Cloud Edge Routers: pub-er-1.oz-implementation.pt, pub-er-2.oz-implementation.pt;
- Internal services (used for local intercepts): webapp.office2.ziti, sas.office2.ziti, ...;

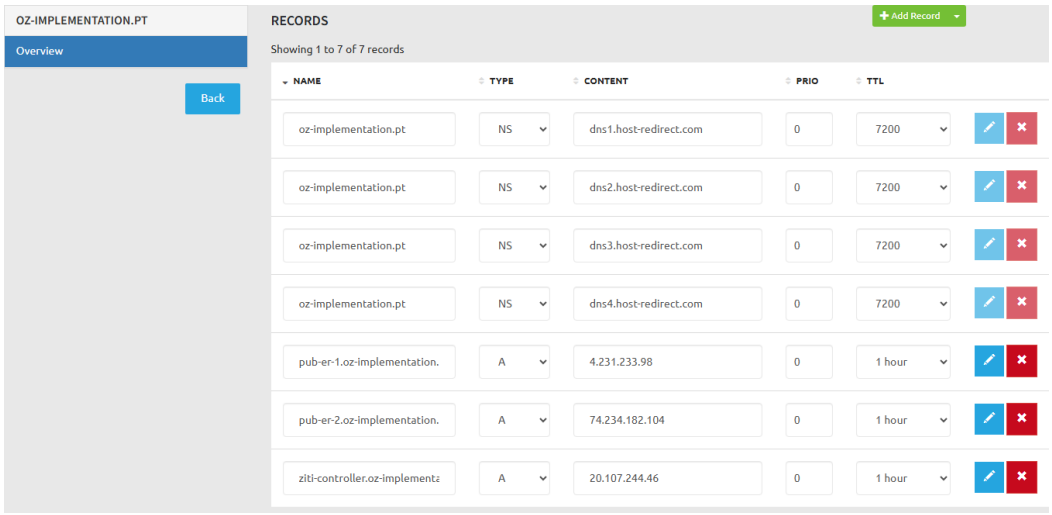


Figure 4: DNS registry

Baseline versions:

- OpenZiti Controller (OZC #1): version 3, running on Ubuntu Server 24.04.3 LTS;
- OpenZiti Edge Routers (OZER's): version 3, running on Ubuntu Server 24.04.3 LTS;
- Ziti Desktop Agent (Windows): version 2.6.4.
- Other Operating Systems: Workstations (WS #1, WS#2,..) - Windows 10 and Windows 11, Internal applications - Ubuntu Server 24.04.3 LTS, AD - Windows Server 2022.

3.3.2.2 *Pre-requisites*

In order to guarantee a successful implementation, the following pre-requisites were taken into consideration:

- Azure subscription with privileges to create Resource Group, VMs, Public IPs and NSGs;
- Administrative privileges in DC #1 in order to install Okta's AD agent;
- Firewall control on Lisbon and Madrid edges, to allow required outbound connectivities;
- Internal PKI, or acceptance to use OpenZiti's built-in PKI, for overlay certificates;
- Permission to provision new VMs within the on-prem infrastructure, for the new OpenZiti Edge routers;
- Privileges within internal applications where it is preferable to implement a ziti SDK, instead of relying on the OpenZiti Edge Routers working as LAN gateway;
- Privileges to either rely on GPOs, or to manually install, on each workstation, the new OpenZiti Desktop agent.

3.3.2.3 *Controller and Edge Routers Provisioning*

For the provisioning of the OpenZiti Controller and the two Cloud Edge Routers, the following steps were taken:

1. Azure VM Creation:
    a) Create a resource group: "rg-oz-prd";
    b) Deploy a new VM (Ubuntu server LTS) with the chosen parameters. In this scenario, it was chosen a standard B1 VM, with 1vcpu and 1 GiB memory;
    c) Create a network security group (NSG) and restrict the inbound communications to the necessary ports only (management and service ports, defined in the configuration file config.yml).
2. OS Hardening;
3. Install OpenZiti Controller:

a) Install controller binaries matching the chosen version (version 3), obtained from OpenZiti's official documentation;

b) Prepare *config.yml* with a public address and port, TLS cert/key paths (either the built-in PKI or a chosen PKI), and the location where data should be stored. Listing 15 represents a controller's yaml file.

4. Install OpenZiti Edge Routers:

a) Install edge router binaries matching the chosen version (version 3), obtained from OpenZiti's official documentation;

b) Generate an enrollment JWT on the controller, and use it to enroll the new edge router;

c) Configure *router.yml* with the controller's public address and port, a public address and port for the router, and the identity jtw file generated previously. Refer to listing 16 for an example of a *config.yml* file.

Regarding the OpenZiti Controller's configuration, it can contain sections such as *ctrl*, *db*, *identity*, *edge*, *web*, *healthchecks*, *events*, *cluster*, *network*, etc. From the sections mentioned, the following are mandatory to start the controller in a minimal "fabric-only" mode (core overlay network built between routers - it is the base functionality that guarantees connectivity between the openziti's components, while being agnostic to users and applications): *ctrl*, *db*/*cluster* and *identity*; while others - *web* and *edge* - enable extra functionality. Among these sections, the following seven sections were configured in this implementation:

1. "db", which specifies the location of the database file, used to store its persistent data - identities, services, policies, configurations, etc;

```
db: "/var/lib/private/ziti-controller/bbolt.db"
```

Listing 1: OpenZiti Controller's config.yml file - "db" variable definition

2. "identity", where it is configured the certificates, and certificate chain, used for outbound client connections, server listening, and CA bundles. By specifying the cert, server_cert, key and ca variables, it is ensured that the controller can both authenticate itself and accept peer connections. Without this section, TLS connections cannot be negotiated;

```
identity:
    cert:        "pki/intermediate/certs/client.chain.pem"
    server_cert: "pki/intermediate/certs/server.chain.pem"
    key:         "pki/intermediate/keys/server.key"
    ca:          "pki/root/certs/root.cert"
```

Listing 2: OpenZiti Controller's config.yml file - "identity" section

3. "ctrl", where it is defined the advertise address (hostname:port) and listener (address:port) for the controller. This defines how routers, and every component in the OpenZiti network, connect to the controller for control-plane operations;

```
ctrl:
  options:
    advertiseAddress: tls:ziti-controller.oz-implementation.pt:1280
  listener: tls:0.0.0.0:1280
```

Listing 3: OpenZiti Controller's config.yml file - "ctrl" section

4. "healthChecks", where it is configured periodic health checks on the controller's database to ensure it is functional;

```
healthChecks:
  boltCheck:
    interval: 30s
    timeout: 20s
    initialDelay: 30s
```

Listing 4: OpenZiti Controller's config.yml file - "healthchecks" section

5. "edge", used to enable "edge functionality", which is the identity-aware layer on top of the "fabric functionality". This is responsible for identities and their enrolment, services, policies, and for transforming OpenZiti into a Zero Trust platform rather than just a VPN.

```
edge:
  api:
    sessionTimeout: 30m #defines how long the session can remain idle.
    address: ziti-controller.oz-implementation.pt:1280 #the hostname:port where
    ↪  clients can find the controller's edge API
  enrollment:
    signingCert: # which intermediate certificate/key the controller uses to sign
    ↪  issued identities.
      cert: pki/intermediate/certs/intermediate.cert
      key:  pki/intermediate/keys/intermediate.key
    edgeIdentity: #validity period for client enrolment tokens.
      duration: 180m
    edgeRouter: #validity period for router enrolment tokens.
      duration: 180m
```

Listing 5: OpenZiti Controller's config.yml file - "edge" section

6. "events", where it is configured how the controller generated events/logs for internal actions. For the purpose of testing, this was configured with several types of events subscribed. In a production environment, it is best to filter these subscriptions, to avoid performance constraints, and for readability purposes.

```
events:
  jsonLogger:
    subscriptions: #defines which type of events the controller should generate.
      - type: circuit
      (...)
      - type: entityCount
        interval: 5s
    handler:
```

```
        type: file
        format: json
        path: /tmp/ziti-events.log
```

Listing 6: OpenZiti Controller's config.yml file - "events" section

7. "web", used to determine how the controller exposes its REST APIs, and web interface (ZAC console, edge client APIs, fabric APIs).

```
web:
  - name: client-management
    bindPoints:
      - interface: 0.0.0.0:1280
        address: ziti-controller.oz-implementation.pt:1280
    identity:
      (...)
    options:
      idleTimeout: 5000ms
      readTimeout: 5000ms
      writeTimeout: 100000ms
      minTLSVersion: TLS1.2
      maxTLSVersion: TLS1.3
    apis:
      (...)
      - binding: zac # administrative GUI console
        options:
          location: /opt/openziti/share/console
          indexFile: index.html
```

Listing 7: OpenZiti Controller's config.yml file - "web" section

Regarding the OpenZiti Router's configuration, it can contain sections such as *identity*, *ctrl*, *link*, *edge*, and *listeners*. From these, the mandatory sections to start the router in a minimal "fabric-only" mode (where the router only participates in the overlay mesh, establishing secure links with other routers) are *identity*, *ctrl*, and *link*. The *edge* section enables the router to function as an edge router, allowing client applications or tunneler agents (such as the Windows Desktop client) to connect through it using identities and policies defined on the controller. For this implementation, the following sections were configured:

1. "identity", where it is configured the certificates and keys used by the router, similarly to the identity section in the controller's configuration.

```
identity:
  cert: /opt/openziti/ziti-router/certs/cert.pem
  server_cert: /opt/openziti/ziti-router/certs/server_cert.pem
  key: /opt/openziti/ziti-router/certs/key.pem
  ca: /opt/openziti/ziti-router/certs/ca.pem
```

Listing 8: OpenZiti Router's config.yml file - "identity" section

2. "ctrl", where it is defined how the router reaches the controller, for the control plane.

```
ctrl:
  endpoint: tls:ziti-controller.oz-implementation.pt:1280
```

Listing 9: OpenZiti Router's config.yml file - "ctrl" section

3. "link", where it is configured the router-to-router data-plane links - "fabric" connections;

```yaml
link:
  dialers:
    - binding: transport
  listeners:
    - binding: transport
      bind: tls:0.0.0.0:80 # local address to listen on. Despite it being defined to port 80, the traffic is fully encrypted.
      advertise: tls:pub-er-1.oz-implementation.pt:80 #the public protocol:host:port other routers should use to reach this listener.
      options:
        outQueueSize: 16
```

Listing 10: OpenZiti Router's config.yml file - "link" section

4. "edge", used to enable "edge functionality" on the router, similarly to the controller's configuration. This is responsible for defining enrolment/renewal certificate parameters and identity-aware access for SDK and tunneler clients.

```yaml
edge:
  heartbeatIntervalSeconds: 60
  csr: #fields used when the router enrols or renews. This should be changed according to the organisation's parameters.
    country: US
    province: NC
    locality: Charlotte
    organization: NetFoundry
    organizationalUnit: Ziti
    sans:
      dns:
        - localhost
        - pub-er-1.oz-implementation.pt
      ip:
        - 4.231.233.98
        - 127.0.0.1
```

Listing 11: OpenZiti Router's config.yml file - "edge" section

5. "listeners", where it is defined non-link server listeners - for instance, edge listeners.

```yaml
listeners:
  - binding: edge
    address: tls:0.0.0.0:443
    options:
      advertise: pub-er-1.oz-implementation.pt:443
```

Listing 12: OpenZiti Router's config.yml file - "listeners" section

A full picture of both configs can be found in Appendix A.

3.3.3 *Identity issuance in OpenZiti's controller*

During the implementation and testing phases, several identities were issued and enrolled, in order to guarantee an adequate suite of different scenarios. These identities were typified into clients and hosts, using OpenZiti roles/tags, where hosts were identities used to publish services (i.e. priv-er-1, referred to as OZER #1 in the architectural diagram), whereas clients were identities used to consume those services (i.e. http-client-1, representing a WS accessing remotely).

Figure 5 is a screenshot taken after finalizing the testing phase, and demonstrates all the identities that existed in this implementation.

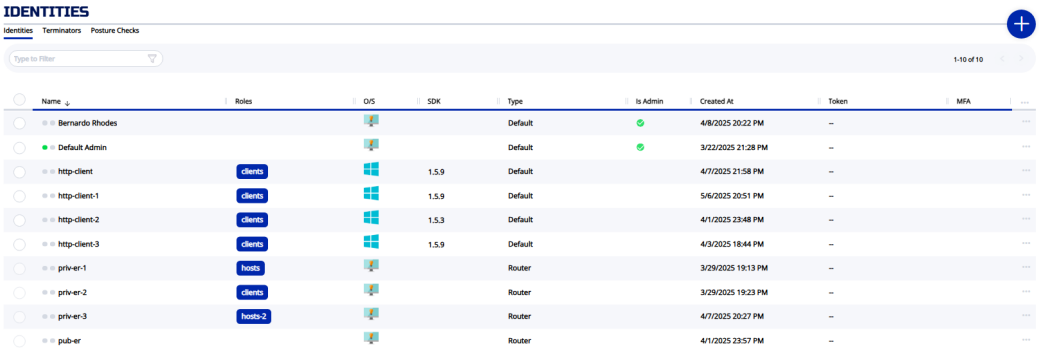


Figure 5: List of issued and enrolled identities

Additionally, there are two identities - 'Bernardo Rhodes' and 'Default Admin' - that do not fall into any of these two categories, as they can be used solely to access the Controller's ZAC (an OpenZiti's module that delivers a GUI to manage and configure the controller). As the name suggests, 'Default Admin' is the identity that is generated alongside the installation of ZAC, and should therefore be disabled. On the other hand, 'Bernardo Rhodes' was created to be used as an admin account that could only access the console through the integration with Okta.

Figures 6 and 7 each represent an example of the two identities mentioned before - 'Bernardo Rhodes' as an identity that can log in with Okta, and 'http-client-1' as a remote workstation that does not log in with Okta - respectively. As it can be seen in the figures, the major difference is in the attribute 'EXTERNAL ID', since this attribute must be filled in with the 'ID' that the IdP will send in the integration, being in this case the email. It can also be seen, in both figures, the services, and service policies, that each identity is assigned - either automatically through the defined tag ("#clients") or by manually assigning a service to an identity. It is relevant to mention that due to the limitations in JIT provisioning, it is not

possible to map these assignments to external groups (such as AD groups, that could be sent by Okta, through OIDC claims or SAML assertions), meaning that all the assignments must still be managed directly on the OpenZiti's controller. This represents a flaw in the identity lifecycle, and should therefore be taken into consideration when choosing this solution.

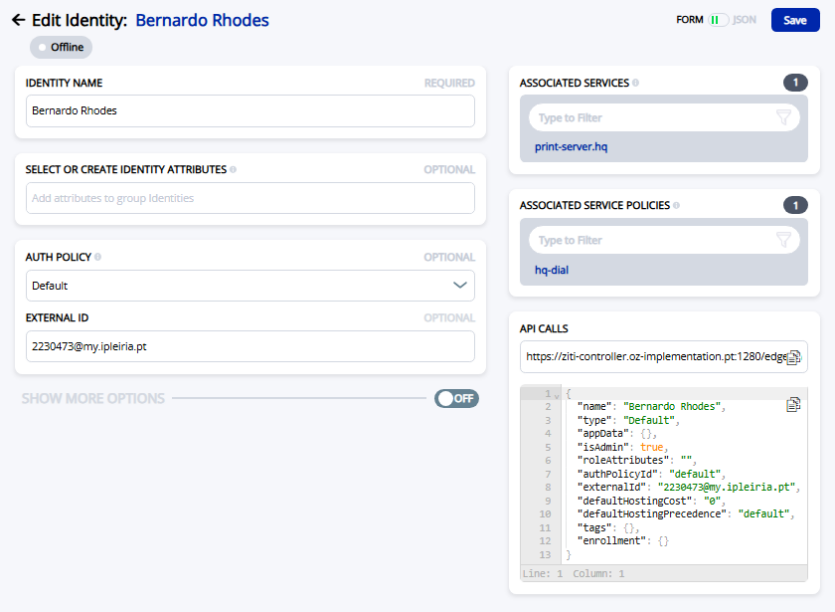


Figure 6: Representation of an identity that logs in with Okta

3.3.4 *Okta SSO configuration*

This section details the configuration of Okta as the Single Sign-On (SSO) provider for the implemented Zero Trust Network Access (ZTNA) solution. While OpenZiti does not support all the features of SAML/OIDC federation, Okta was integrated to act as the enterprise Identity Provider (IdP), synchronised with the organisation's on-premises Active Directory (AD). This ensured that all ZTNA identities were ultimately derived from centrally managed user accounts (AD accounts), even though provisioning into OpenZiti remained manual.

3.3.4.1 *Active Directory Integration*

Taking into consideration the architectural topology idealized for this implementation, figure 3, it is fundamental to integrate Okta (as the chosen IdP) with the Identity Directory the organisation has - in this case, Active Directory. As such, the

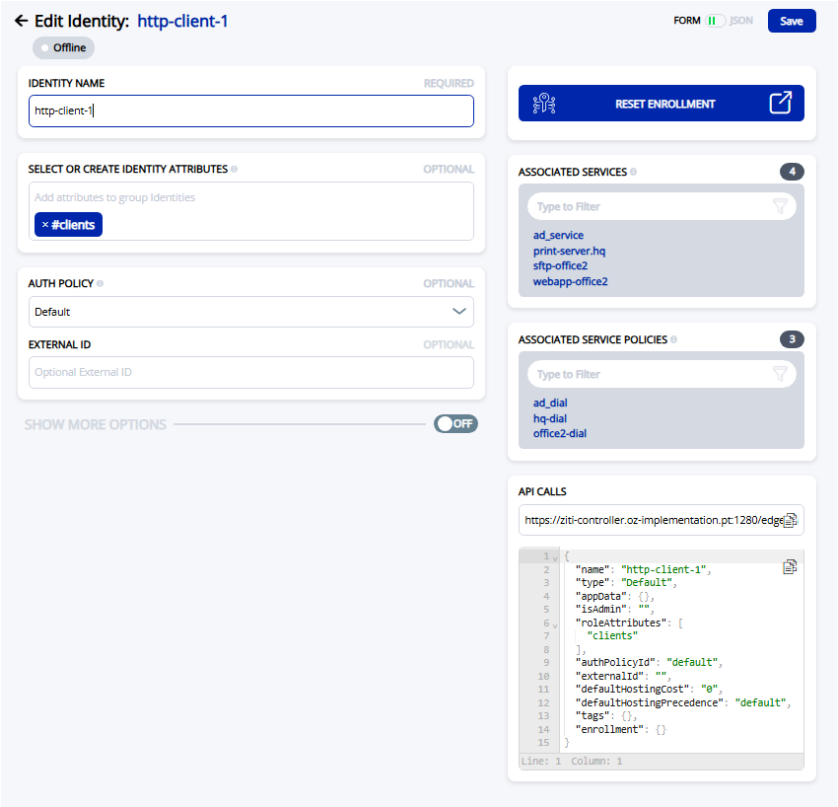


Figure 7: Representation of an identity that does not log in with Okta

following steps were taken to guarantee a correct integration, and synchronisation of identities between the AD and Okta:

- Download Okta's AD Agent directly from Okta's tenant;
- Prepare a service account that will be used for the AD Agent to authenticate against the AD;
- Access the DC #1 and install the Okta AD Agent. The installation prompts for Okta's tenant URL, service account, and Organizational Units (OUs) that will be synchronised;
- Verify that the agent can communicate with Okta's tenant, via outbound connectivity, on port 443.
- Finish the configurations on Okta's tenant:
  - Define all the attributes that will be mapped, or leave the out-of-the-box attributes. By default, Okta will map all the needed attributes;
  - Define what attribute will be mapped to the ID within Okta, and its format: usually samAccountName;

  - Activate Delegated Authentication (use AD credentials in Okta) and JIT provisioning. To clarify, JIT was enabled in Okta to populate the Okta directory from AD on first login;
  - Define import interval: 1 hour;
  - Proceed with the first full import, to synchronise all identities and groups from the previously defined OUs.

3.3.4.2 *Creating a new Application in Okta, and configuring it in OpenZiti's Controller*

This sections details the procedure of creating a new OpenID Connect (OIDC) application within Okta, and configuring it in the OpenZiti's controller. OpenZiti's documentation was used to guarantee a correct integration between the solutions.

- Creating a new application in Okta:
  - Create a new app integration, with 'OIDC - OpenID Connect' as the sign-in method and 'Single-Page Application' as the application type;
  - Define a new name for the application, i.e. 'OpenZiti', guarantee that both 'Authorization Code' and 'Refresh Token' are *ticked*, in the 'Grant type' section, and define a 'Sign-in redirect URI' according to what will be integrated: ziti-controller.oz-implementation.pt:1280/zac/callback in this implementation;
  - Configure a new Authorization Server, needed to correctly configure Okta as an External JWT signer in OpenZiti:
    * Add a new Authorization server with a name, audience and description. The audience will later be used to configure Okta in OpenZiti;
    * Add a new Policy to the Authorization server, and a new Policy Rule, with the 'Authorization Code' grant allowed. The remainder configurations can be customised according to the security standards of the organisation;
    * Add a new claim representing the email of the users that will be accessing OpenZiti's Controller, and add it to a scope: email/openid/profile. This must be sent in the JWT.
- Configuring the application in the OpenZiti Controller (Refer to figure 8):

– After creating the application and authorization server in Okta, add a new JWT signer within the OpenZiti's Controller;

– Fill in with the following attributes, obtained from Okta:

  * Name: (User defined) Okta
  * Issuer: (Obtained from the authorization server) {tenant-url}/oauth2/{authorization-server-id}
  * Audience: (Defined during the creation of the authorization server) openziti
  * Claims property: (Scope containing the email claim): email
  * Client ID: (Obtained from the application 'OpenZiti') 0oaq5**********
  * External auth URL: (Obtained from the authorization server) {tenant-url}/oauth2/{authorization-server-id}
  * Scopes: (OIDC scopes) openid, profile, email
  * JWKS endpoint: (Obtained from the authorization server) {tenant-url}/oauth2/{authorization-server-id}/.well-known/v1/keys

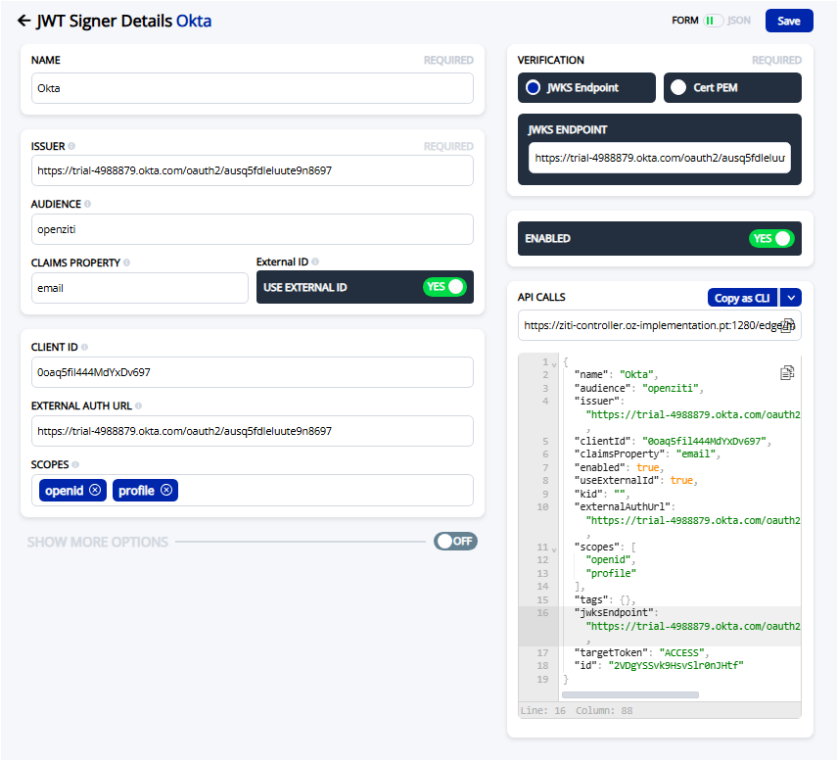


Figure 8: Okta configuration in OpenZiti's Controller

3.3.5 *Service Definition and Policies*

Once the controller, routers, identity provider, and identities were provisioned, the next critical step was to define how services would be exposed within the ZTNA overlay and how access would be controlled. Contrary to how VPNs allow access to an entire subnet, OpenZiti, as a ZTNA solution, enforces service-level access. This means that every application or server has to be explicitly described in terms of how clients will connect to it (intercept configuration) and how routers will forward traffic to the actual backend system (host configuration). As such, the following services were configured:

- ad_service (Active Directory)
- print-server.hq (Print Server #1)
- sftp-office2 (SAS #1)
- webapp-office2 (WAS #1)

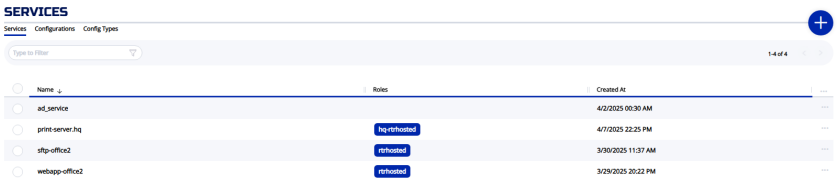


Figure 9: List of configured services

It is important to highlight that in OpenZiti a service definition is never complete without both its configurations and its policies. Each service requires at least two configuration objects - an intercept (intercept.v1 was used), which defines what the client application sees and how the Ziti agent captures traffic, and a host (host.v1 was used), which specifies how and where the service is actually provided in the private network. On top of that, access control was enforced through three types of policies: Bind, Dial, and Router. Bind policies dictated which routers were allowed to present a given service. For instance, the Madrid router (OZER #2) was the only one permitted to bind WAS #1, ensuring that the web application would only be hosted by the correct router. Dial policies controlled which identities were authorised to use a service, and they can be associated to specific users, or groups of users, by using OpenZiti tags. In this implementation, it was used a tag, "clients", to authorise a group of users to dial the services, and a tag "hosts" to identify which routers could host which services. Router policies determined which routers were eligible to forward traffic to which services. During testing, and due to the architecture chosen,

these were deliberately broad - all routers could reach all services, and all identities could reach all public routers - but in production it is a best practice to restrict these policies so that, for instance, remote users could not reach Madrid-only resources unless explicitly required. As such, a service becomes operational only when its intercept and host configurations are both defined, and at least one Bind, one Dial, and one Router policy match the involved service. Any missing element results in a service that exists in the controller but is completely invisible and unreachable by clients, by design. The following policies and configurations, represented in figures 10 and 11, respectively, were configured:

- Policies:
  - ad_bind
  - ad_dial
  - hq-bind
  - hq-dial
  - office2-bind
  - office2-dial
  - office2-webapp-dial
- Router Policies:
  - all-routers-all-services
  - all-endpoints-public-routers
- Configurations:
  - ad_host
  - ad_intercept
  - https-host-config
  - https-intercept-config
  - print-server-host-config
  - print-server-intercept-config
  - sftp-host-config
  - sftp-intercept-config

As such, the following tree structure represents all the services, and respective policies and configurations, configured within OpenZiti. For each service, policy and

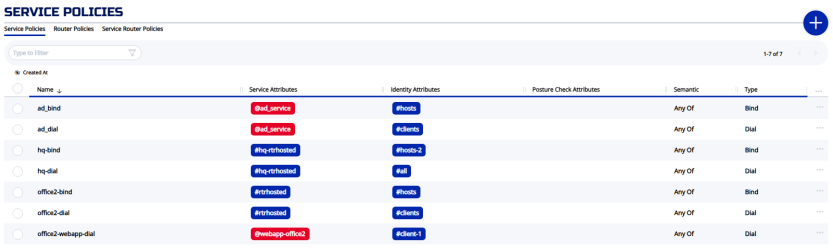


Figure 10: List of configured policies

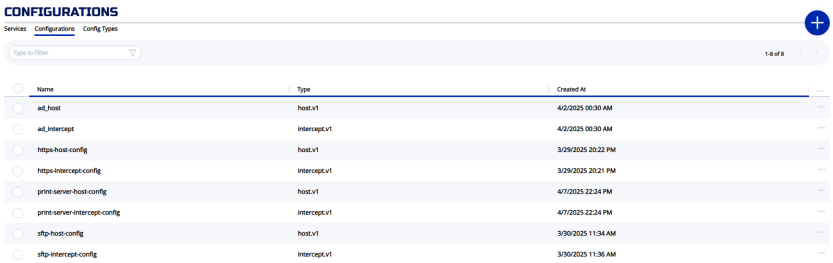


Figure 11: List of configured configurations

configuration, there is the respective screenshot, that contains all the attributes and configurations chosen for its definition.

- Service: **ad_service** - Figure 12

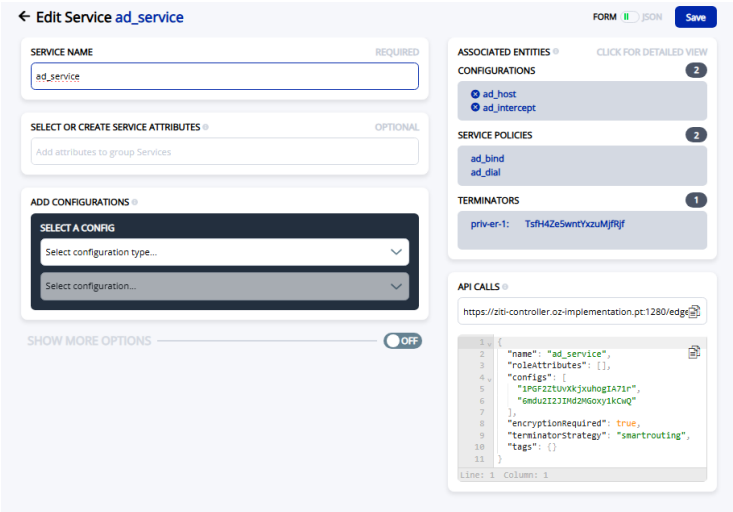


Figure 12: AD Service

  - Policies: **ad_bind** and **ad_dial** - Figures 13 and 14. The **ad_bind** policy defines what OpenZiti router will host the service (priv-er-1, or OZER #2, via the tag "#hosts"), and the **ad_dial** policy defines what OpenZiti components can reach this service (end-users http-client through http-client-3 and priv-er-2, or OZER #1, via the tag #clients). It is

important to note that priv-er-2 was allowed to reach this service due to its LAN topology - since this router worked as a LAN gateway, all workstations connected to it could therefore reach this service.

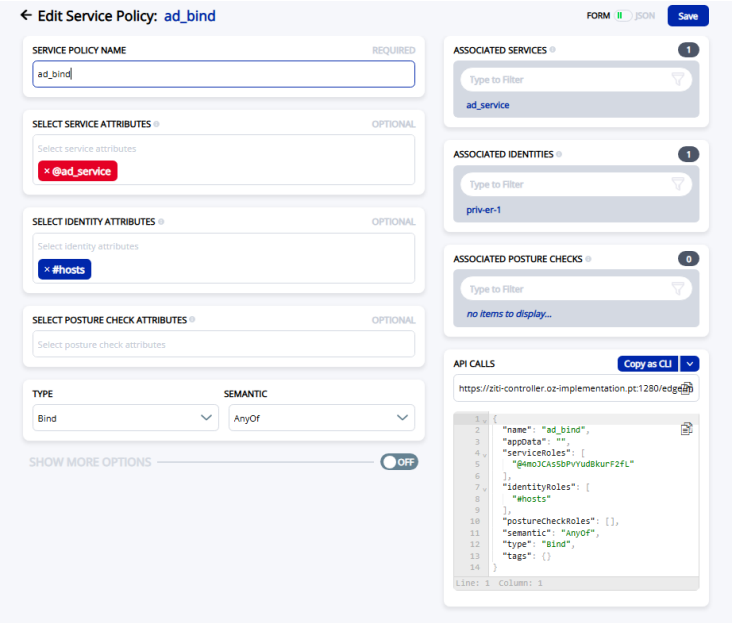


Figure 13: AD Bind Policy

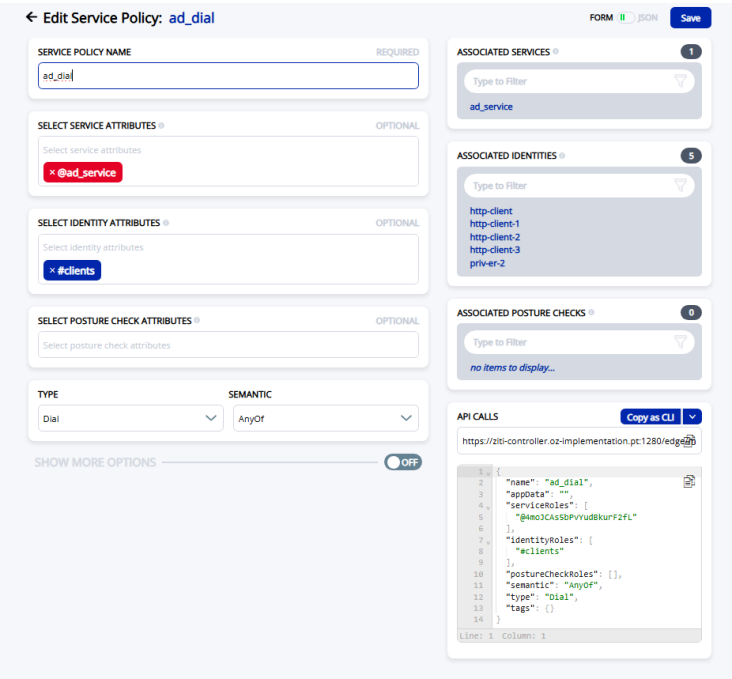


Figure 14: AD Dial Policy

– Configurations: **ad__host** and **ad__intercept** - Figures 15 and 16. The **ad__host** configuration defines where the service is hosted (since this

is a service for AD, it is used the domain name with a wilcard: *.oz-implementation.local), and what ports are allowed, telling the bind router how to reach it. On the other hand, the **ad_intercept** configuration tells the routers what connections to intercept, and redirect towards the bind router (all connections with the destination address *.oz-implementation.local).

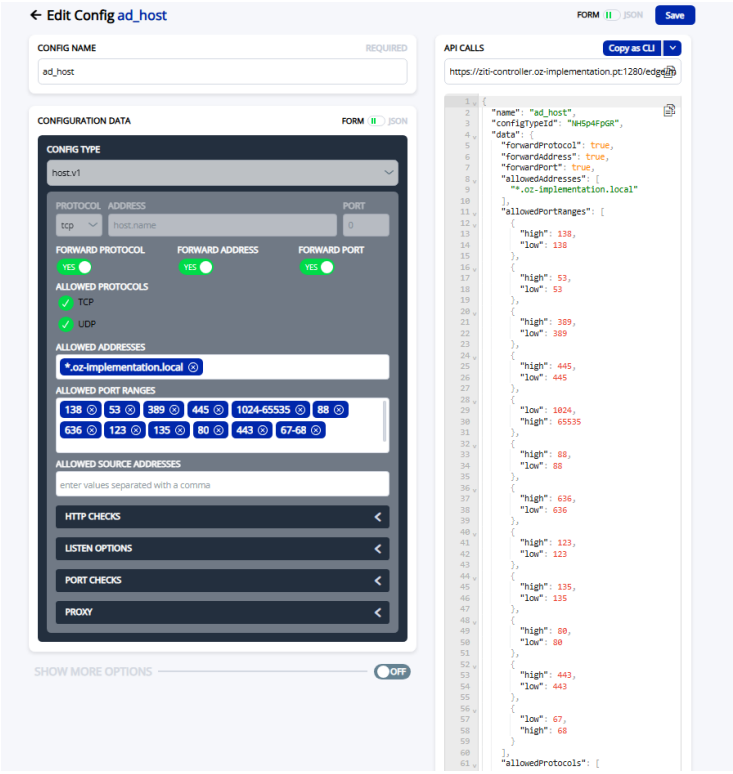


Figure 15: AD Host Configuration

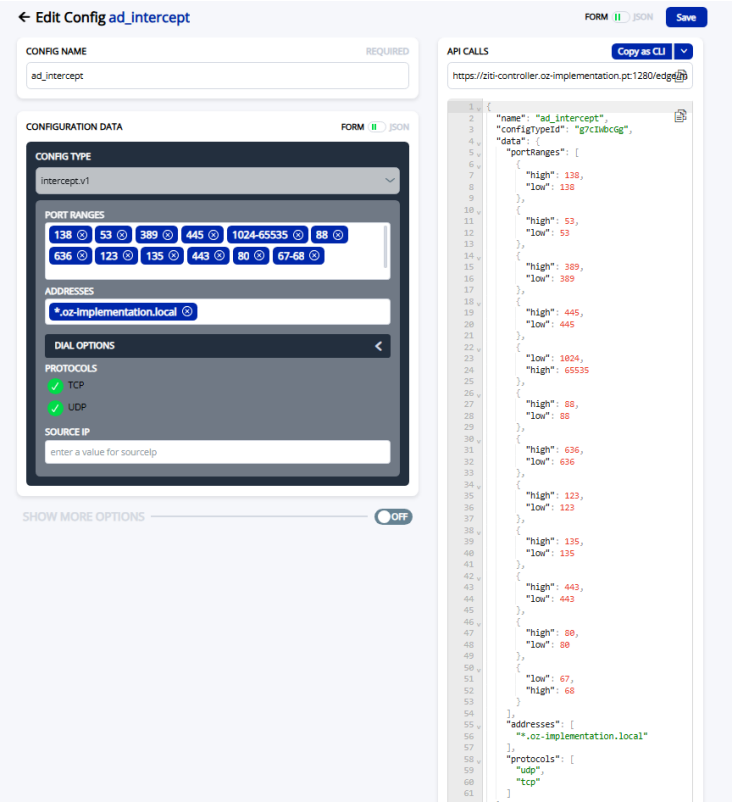


Figure 16: AD Intercept Configuration

- Service: **print-server.hq** - Figure 17

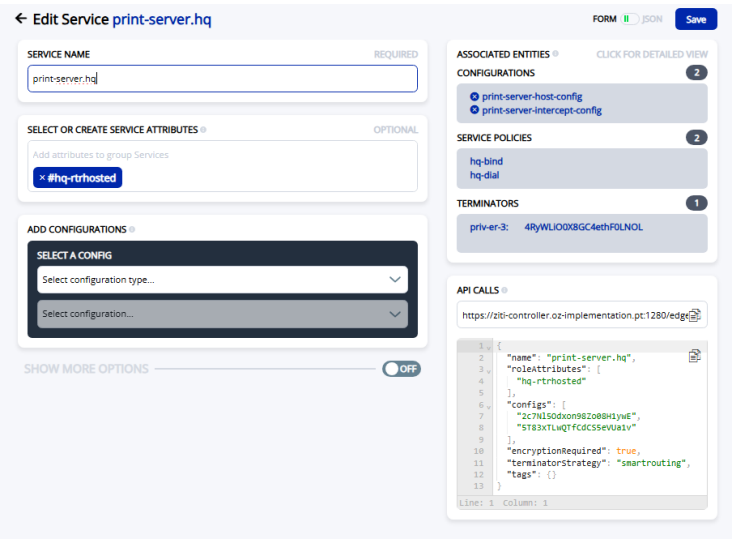


Figure 17: Print Server Service

  - Policies: **hq-bind** and **hq-dial** - Figures 18 and 19. The **hq-bind** policy defines what OpenZiti router (priv-er-3, or OZER #1, via the tag "#hosts-2") will host the services associated with it (print-server.hq), and the

hq-dial policy defines what OpenZiti components can reach this service (all end-users and routers via the tag #hq-rtrhosted).

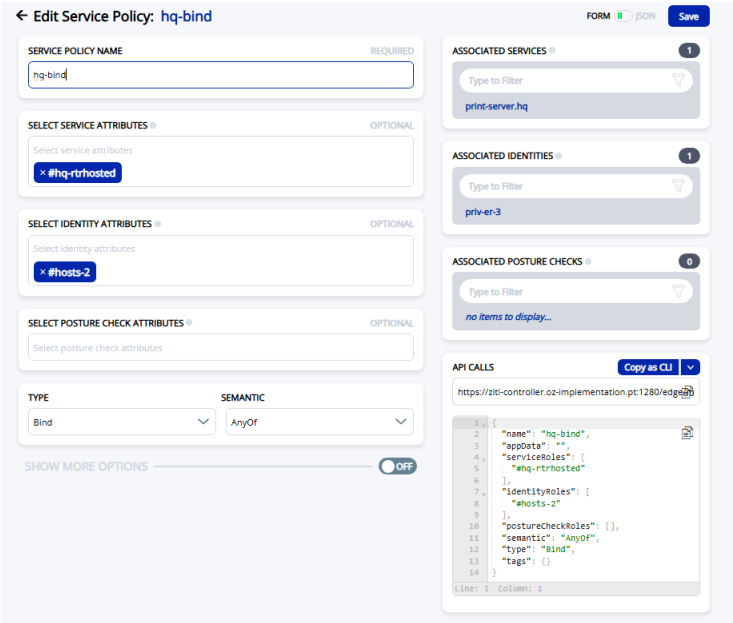


Figure 18: HQ Bind Policy

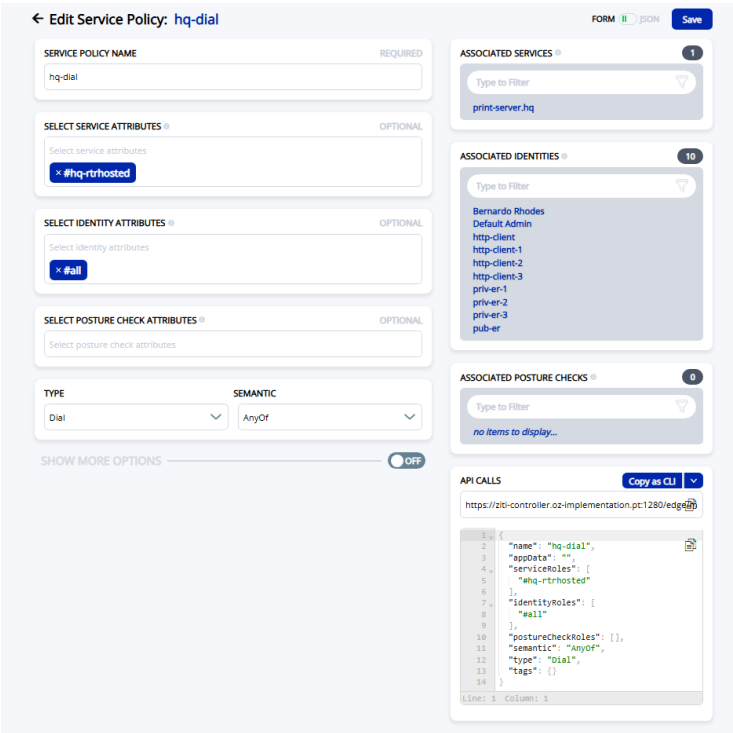


Figure 19: HQ Dial Policy

– Configurations: **print-server-host-config** and **print-server-intercept-config** - Figures 20 and 21. The **print-server-host-config** configuration

defines where the service is hosted (192.168.1.132), and what ports (631) are allowed, telling the bind router how to reach it. On the other hand, the **print-server-intercept-config** configuration tells the routers what connections to intercept, and redirect towards the bind router (all connections with the destination address print-server.hq.ziti and port 631).

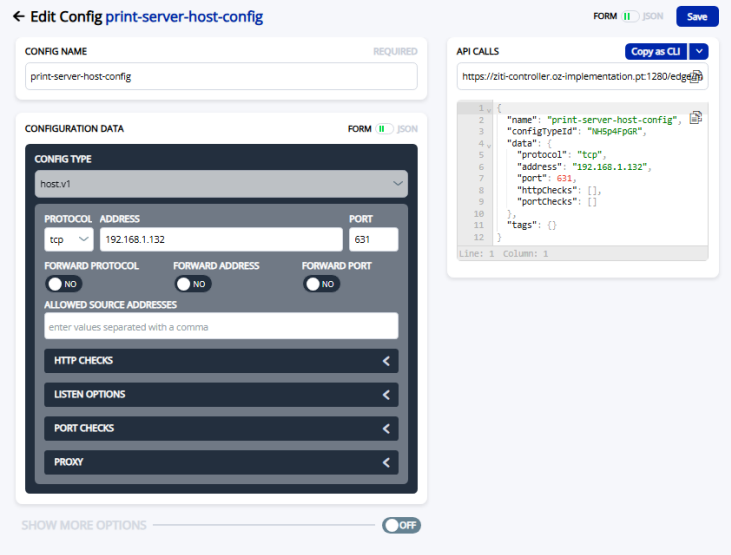


Figure 20: Print Server Host Configuration

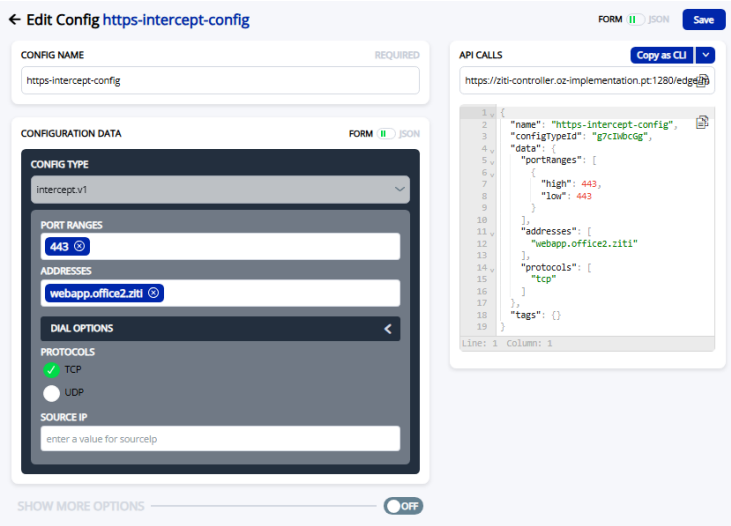


Figure 21: Print Server Intercept Configuration

- Service: **sftp-office2** - Figure 22

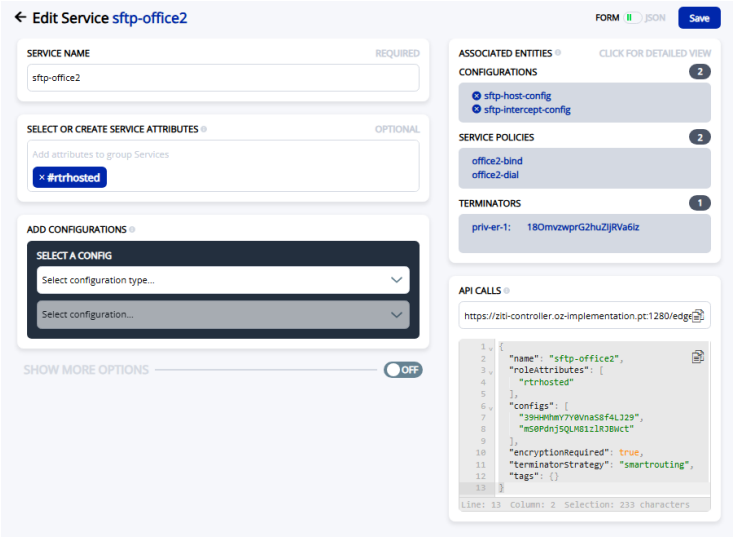


Figure 22: SFTP Service

– Policies: **office2-bind** and **office2-dial** - Figures 23 and 24. The **office2-bind** policy defines what OpenZiti router (priv-er-1, or OZER #2, via the tag "#hosts") will host the services associated to it (sftp-office2 and webapp-office2), and the **office2-dial** policy defines what OpenZiti components can reach this service (end-users http-client through http-client-3 and priv-er-2, or OZER #1, via the tag #clients).

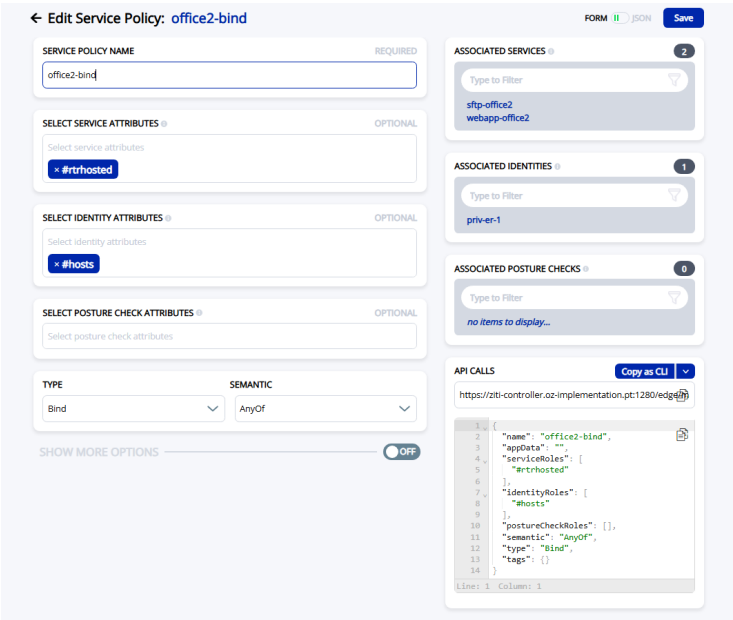


Figure 23: Office 2 (Madrid) Bind Policy

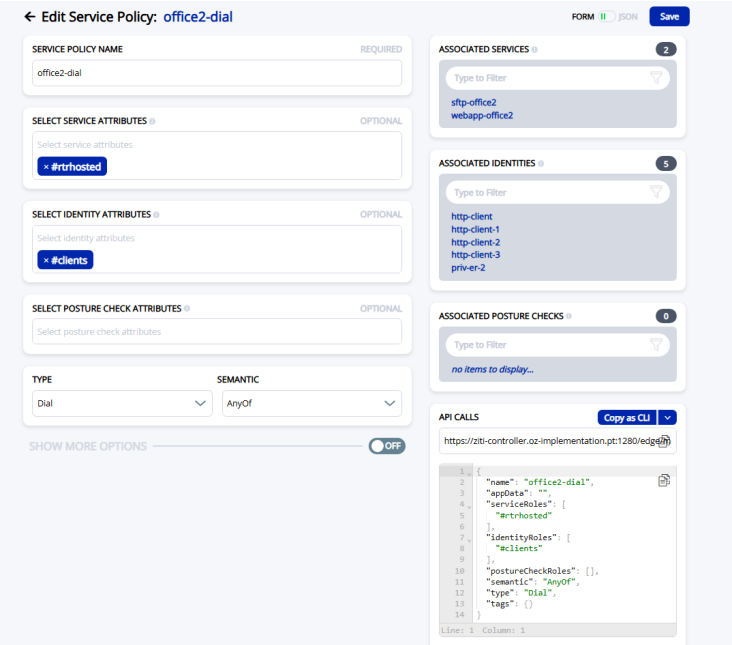


Figure 24: Office 2 (Madrid) Dial Policy

– Configurations: **sftp-host-config** and **sftp-intercept-config** - Figures 25 and 26. The **sftp-host-config** configuration defines where the service is hosted (192.168.0.174), and what ports (22) are allowed, telling the bind router how to reach it. On the other hand, the **sftp-intercept-config** configuration tells the routers what connections to intercept, and redirect towards the bind router (all connections with the destination address sas.office2.ziti and port 22).

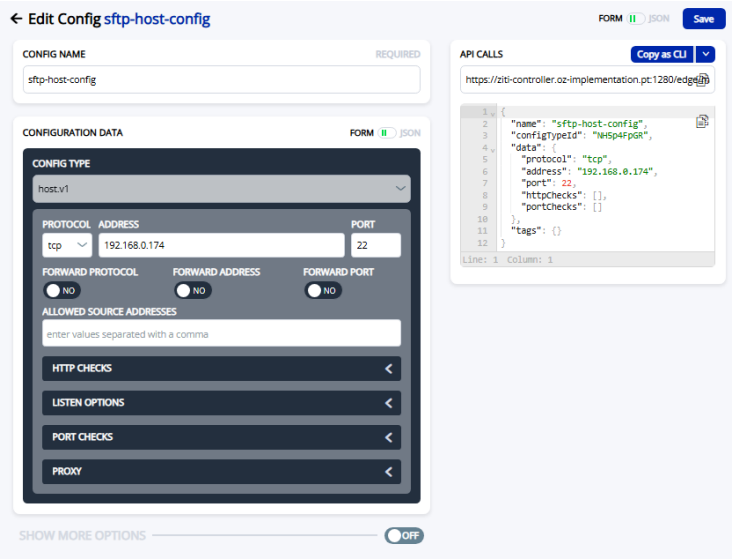


Figure 25: SFTP Host Configuration

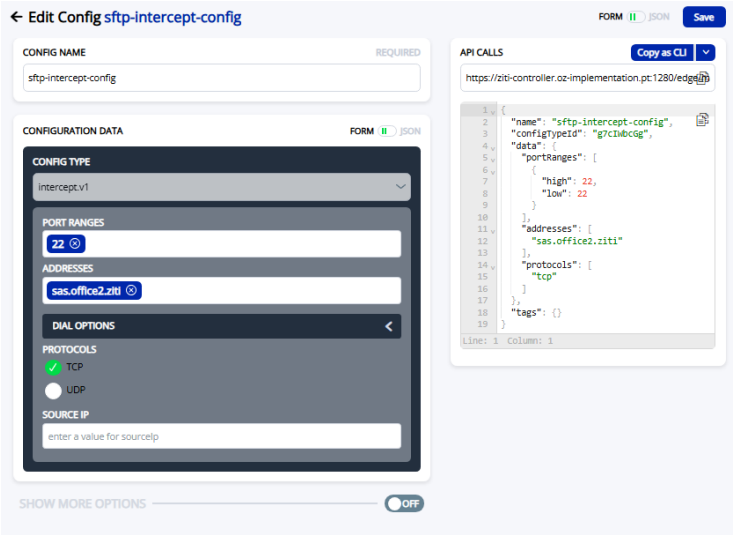


Figure 26: SFTP Intercept Configuration

- Service: **webapp-office2** - Figure 27

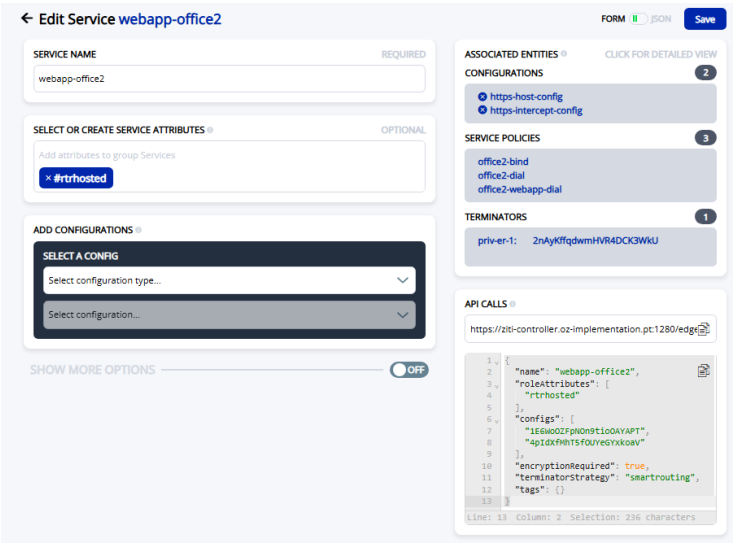


Figure 27: Webapp Service

  - Policies: **office2-bind**, **office2-dial** and **office2-webapp-dial** - Figures 23, 24 and 28. The policies used were the same used for the sftp service.

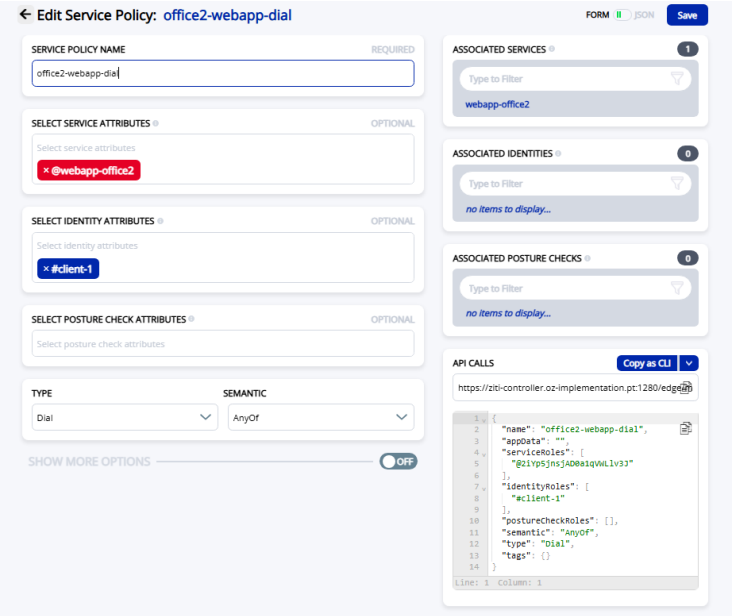


Figure 28: Office 2 (Madrid)-Webapp Dial Policy

– Configurations: **https-host-config** and **https-intercept-config** - Figures 29 and 30. The **https-host-config** configuration defines where the service is hosted (192.168.0.174), and what ports (8443) are allowed, telling the bind router how to reach it. On the other hand, the **sftp-intercept-config** configuration tells the routers what connections to intercept, and redirect towards the bind router (all connections with the destination address webapp.office2.ziti and port 443).

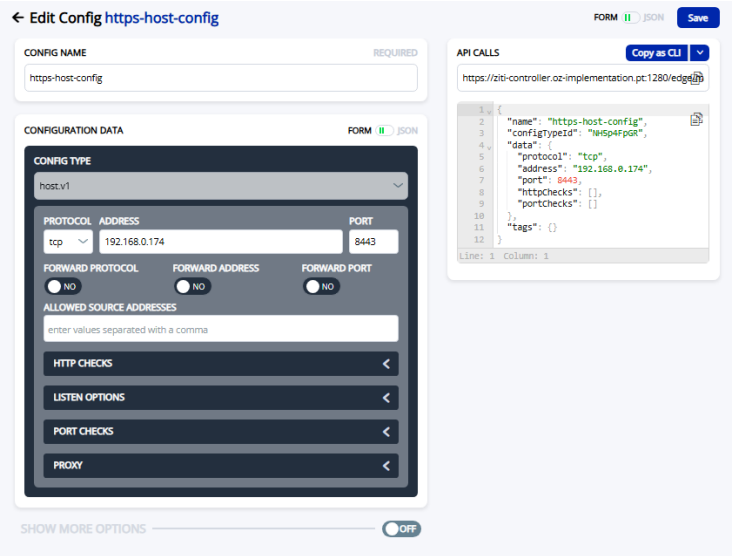


Figure 29: HTTPs (webapp) Host Configuration

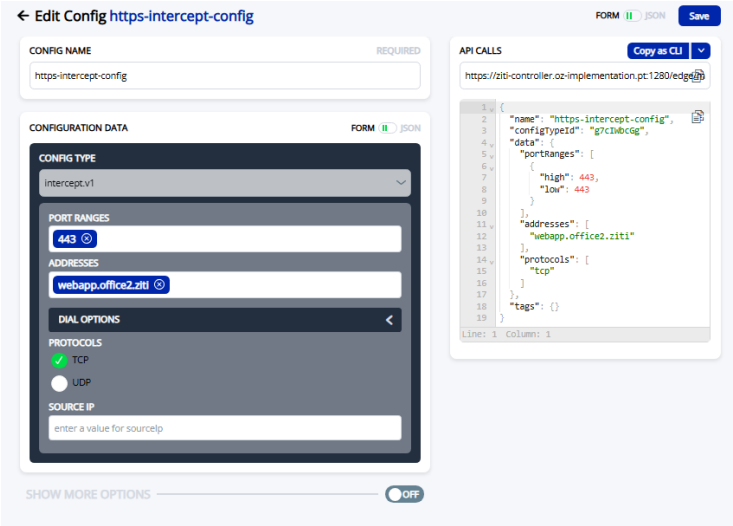


Figure 30: HTTPs (webapp) Intercept Configuration

3.4 SUMMARY

This chapter presented the rationale behind selecting OpenZiti as the open-source ZTNA solution, followed by the design of a hybrid SME architecture that replaces a legacy VPN model. It detailed, in a step-by-step approach, the deployment topology, component provisioning, identity issuance, Okta SSO integration, and service/policy definitions, supported by configuration artefacts and screenshots.

The next chapter, Tests, validates the implemented architecture through a structured suite of tests. Each test examines critical aspects such as access enforcement, encryption, lateral movement, policy propagation, impersonation prevention, and logging visibility, mapping results against real-world adversary behaviours.

4

TESTS

This section presents a set of test cases designed to validate the security, functionality, and operational behaviour of the implemented Zero Trust Network Access (ZTNA) architecture using OpenZiti. The tests focus on verifying access control, encryption, identity enforcement, segmentation and auditability.

4.1 LOGIC FLOW DIAGRAMS

Figures 31, 32 and 33 represent the logical flows underpinning all test cases in this chapter. Each diagram highlights a different sequence of actions showing how OpenZiti handles identity verification, service access, policy enforcement, and exceptional scenarios. The figures may contain three different colours: green, orange and red. The green colour represents authentication and authorisation requests, the orange colour represents all actions on the control-plane, and the red colour is meant for authentication/authorisation requests that were denied (i.e. a user requested access to a service that it was not authorised to access). Finally, the colour black is fundamentally the flow that the requests, to access the desired service, follow.

4.1.1 *High-Level Test Scenario 1 - Authorised Access*

Illustrates the basic access request path. An authenticated identity attempts to access a service, and the request is validated by the OpenZiti Controller against defined policies (dial, bind, and router). If permitted, the connection is established through the authorised Edge Router to the target service. The steps taken in this flow were as follows:

1. A client authenticates with the Ziti Desktop Edge Agent using its issued identity;
2. The Ziti Desktop Edge Agent requests all the services that the identity used is authorised to access, alongside their DNS entries;

3. The user attempts to access a specific service, through the reachable edge router (OZER #4). This step is only possible for the services the identity is authorised to access, since only for those services will the agent know their DNS entries, and be able to forward the requests;
4. The edge router requests a validation of the request to the controller, and the controller validates it against service policies (dial, bind and router);
5. If authorised (which is, in this scenario), the Controller establishes a secure overlay tunnel between client and service;
6. The client communicates with the service over an encrypted channel.

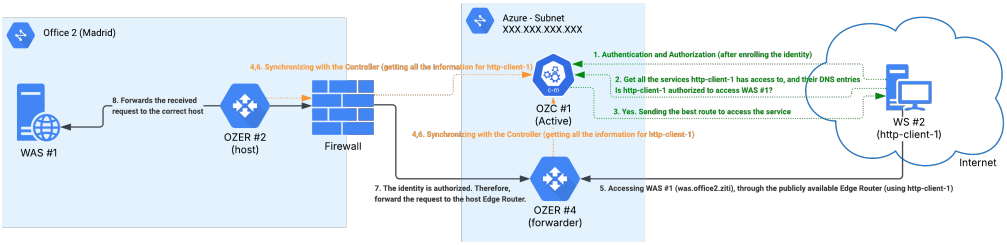


Figure 31: Test Diagram - Logic Flow #1

This logical flow represents Tests T01 (Policy Enforcement) and T02 (Encryption Validation), confirming that only authorised services can be reached and that all traffic is protected.

4.1.2 *High-Level Test Scenario 2 - Unauthorised Access*

This diagram focuses on the dynamic aspect of access control, where policies and identities may change after an initial access request. It represents the flow of the communication when an identity tries to access a service that is not authorised to access (either never was, or the authorisation was revoked). The steps taken were as follows:

1. A client authenticates with the Ziti Desktop Edge Agent using its issued identity;
2. The Ziti Desktop Edge Agent requests all the services that the identity used is authorised to access, alongside their DNS entries;
3. The user attempts to access a specific service, but there is no DNS record of that service, nor there is any known IP address for that service. As such, the requests are not forwarded;

4. The client cannot communicate with the service.

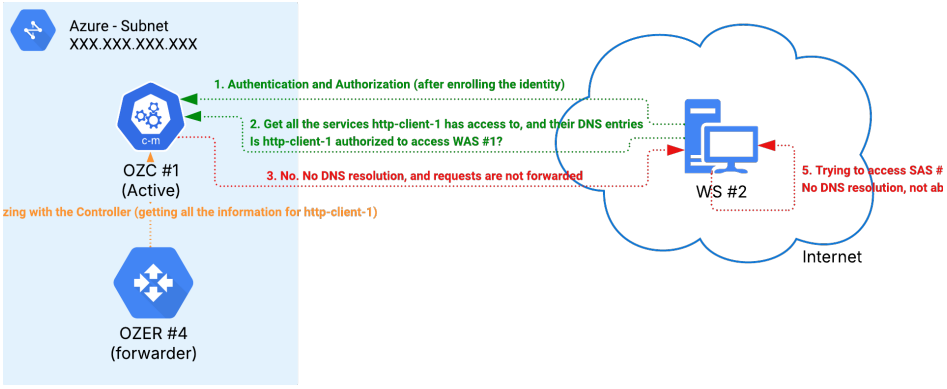


Figure 32: Test Diagram - Logic Flow #2

This logical flow represents Tests T04 (Identity Revocation) and T05 (Service Policy Update Propagation), demonstrating how quickly the system enforces changes across the network.

4.1.3 *High-Level Test Scenario 3 - Malicious Access*

The last diagram represents a scenario where a malicious actor attempts to probe the OpenZiti network for all published services, in order to discover unauthorised services, after getting initial access to an identity.

1. A client authenticates with the Ziti Desktop Edge Agent using its issued identity;
2. The Ziti Desktop Edge Agent requests all the services that the identity used is authorised to access, alongside their DNS entries;
3. The client tries to scan all the available IPs in the Ziti network;
4. For the authorised services, OpenZiti forwards them accordingly, and establishes a secure tunnel to the services that the identity is authorised to access. All other attempts are rejected/dropped as there are no DNS records, nor registered IPs for those services, for that specific identity;
5. If the targeted identity has access to a server within the organisations infrastructure, OpenZiti cannot limit lateral movement, and the malicious actor may be able to achieve privilage escalation if there are not additional controls in place.

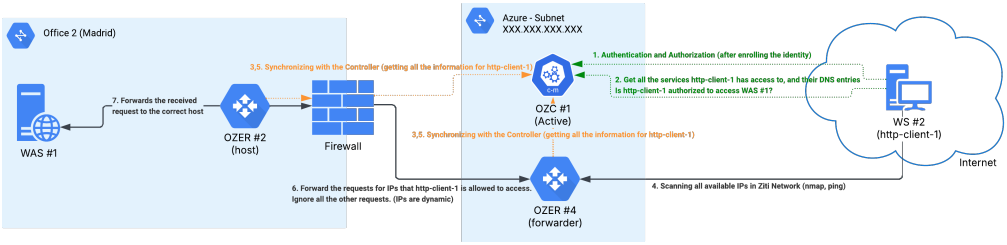


Figure 33: Test Diagram - Logic Flow #3

This logical flow represents Tests T03 (Lateral Movement Attempt) and T06 (Unauthorised Service Discovery Prevention), demonstrating how the system reacts under misuse or malicious actions.

4.2 DETAILED TEST CASES

For each test case (T01-T10), it is outlined its objective, the procedure followed, the observed results, and, where relevant, the supporting visual evidence.

4.2.1 *Test T01 – Policy Enforcement*

1. **Objective:** To verify that access is restricted to authorised services, in accordance with defined ZTNA policies.
2. **Procedure:** Authenticate to the Ziti network using the Desktop Edge client. Attempt to access three services: WAS #1 (authorised), SAS #1 and DC #1 (unauthorised).
3. **Result:** WAS #1 responded with HTTP 200 (TLS handshake via port 8443), while SAS #1 and DC #1 dropped TCP SYN requests due to missing policy bindings. This validates that Ziti's service based policy enforcement functions as expected.

Figures 34, 35, 36 and 37 identify the authorised services, and demonstrate an attempt made to access one authorised service, and two unauthorised services.

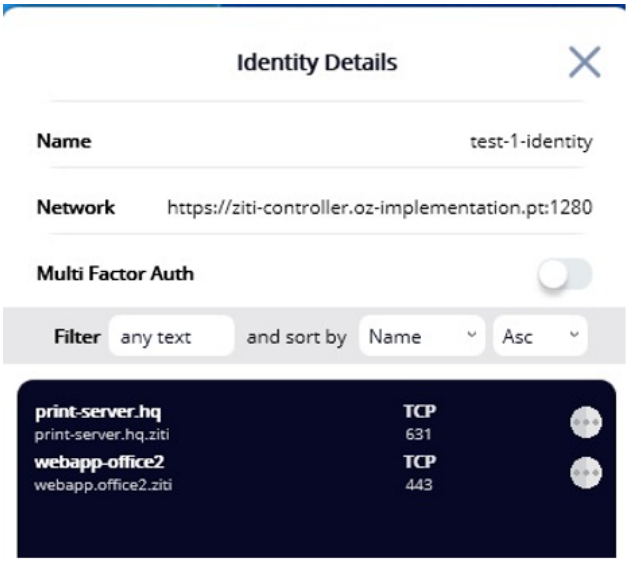


Figure 34: Authorised services for the authenticated identity

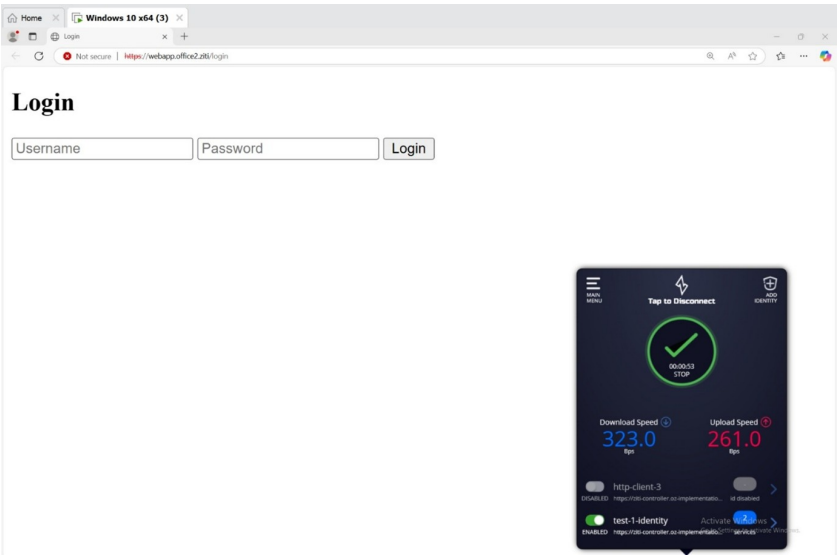


Figure 35: Successful attempt to access the authorised service (WAS #1)

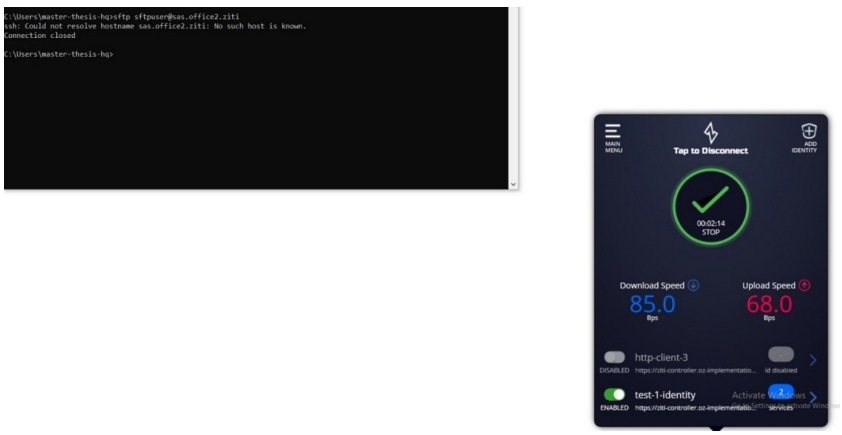


Figure 36: Unsuccessful attempt to access an unauthorised service (SAS #1)

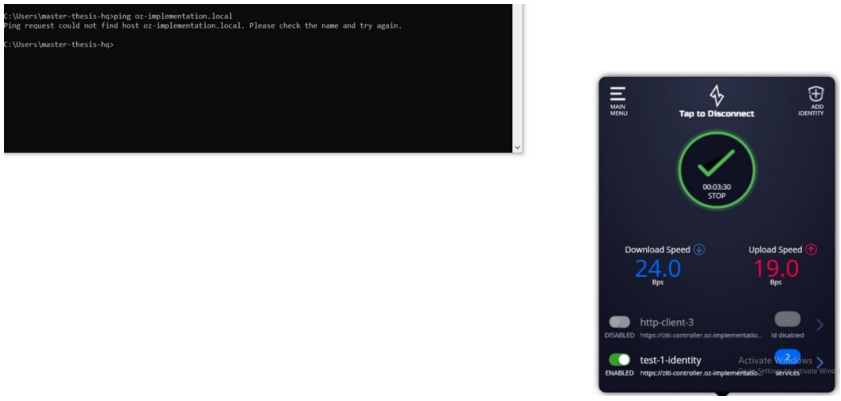


Figure 37: Unsuccessful attempt to access an unauthorised service (DC #1)

4.2.2 *Test T02 - Encryption Validation*

1. **Objective:** To confirm that service traffic is fully encrypted and internal IP addresses remain hidden.
2. **Procedure:** Monitor the local network interface using Wireshark while establishing a session with WAS #1 through the Ziti client.
3. **Result:** Captured packets showed only encrypted payloads with no visible HTTP content or internal IP metadata, and with a TLS handshake visible. A filter for the service's ports was applied (ports 8443 and 22) to isolate relevant traffic, and wireshark was configured to capture packets on the Ziti interface. This confirms successful end-to-end encryption and overlay tunnel encapsulation provided by Ziti.

As observed in the figure below, 38, the TLS 1.3 handshake (Server Hello, Change Cipher Spec) establishes encryption keys, after which all traffic appears only as encrypted Application Data, confirming that the session is fully protected and no readable content is exposed.

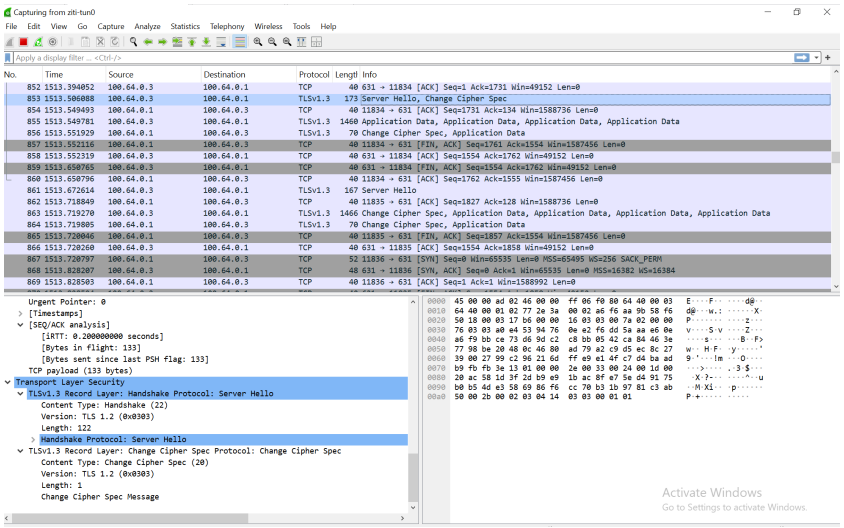


Figure 38: Wireshark screenshot confirming traffic encryption

4.2.3 *Test T03 – Lateral Movement Attempt*

1. **Objective:** To assess whether lateral movement within the internal network is possible from an authorised service.

2. **Procedure:** After authenticating and accessing SAS #1 via Ziti, initiate a scan of the internal network to detect and connect to WAS #1.
3. **Result:** Lateral movement was possible within the LAN, allowing direct access from SAS #1 to WAS #1. This outcome is expected, as Ziti restricts ingress access but does not enforce internal east-west traffic segmentation. To mitigate east-west movement, internal micro-segmentation via VLANs or host-based firewalls is required in addition to the ZTNA overlay.

According to the figures 35, 36 and 37, it is possible to conclude that it is not possible to even reach unauthorised services. However, after accessing SAS #1, via SSH, it is still possible to then access any other server/workstation on the network if additional controls are not in place, since OpenZiti does not control those communications.

4.2.4 *Test T04 – Identity Revocation*

1. **Objective:** To validate that revoked identities are immediately denied access.
2. **Procedure:** Access WAS #1 using a valid identity. Then, revoke the identity via the OpenZiti Controller and attempt to reconnect.
3. **Result:** Once the identity was revoked, any new connection attempts were blocked, and the ongoing connections were interrupted with no registered delay. This confirms that identity revocation policies are enforced in real-time.

Figures 39, 40 and 41, illustrate the behaviour observed when a valid identity was revoked, or reset, from the OpenZiti controller. After revocation, the affected user attempted to reconnect to the Web Application Server (WAS #1). As shown, the connection request was immediately denied, without noticeable delay. This confirms that OpenZiti enforces revocation policies in real time, ensuring that compromised or deactivated identities cannot retain access to protected services.

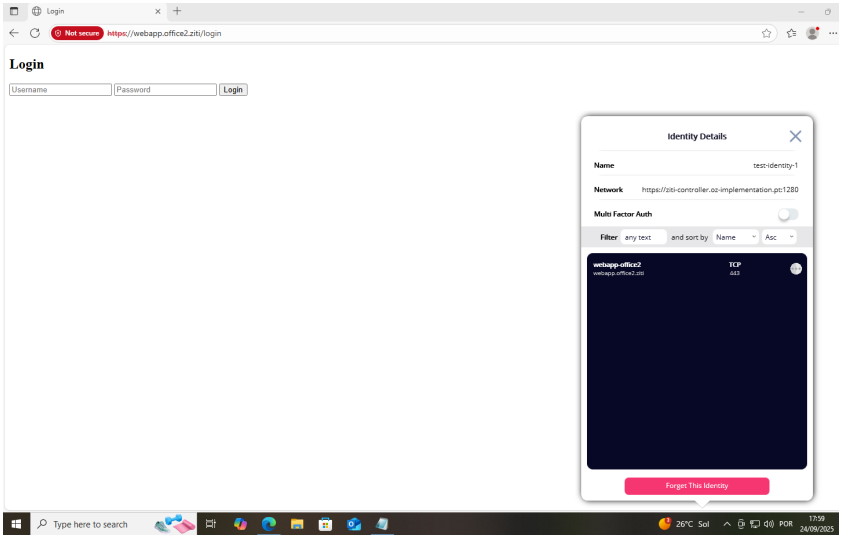


Figure 39: Accessing WAS #1 with an authorised identity

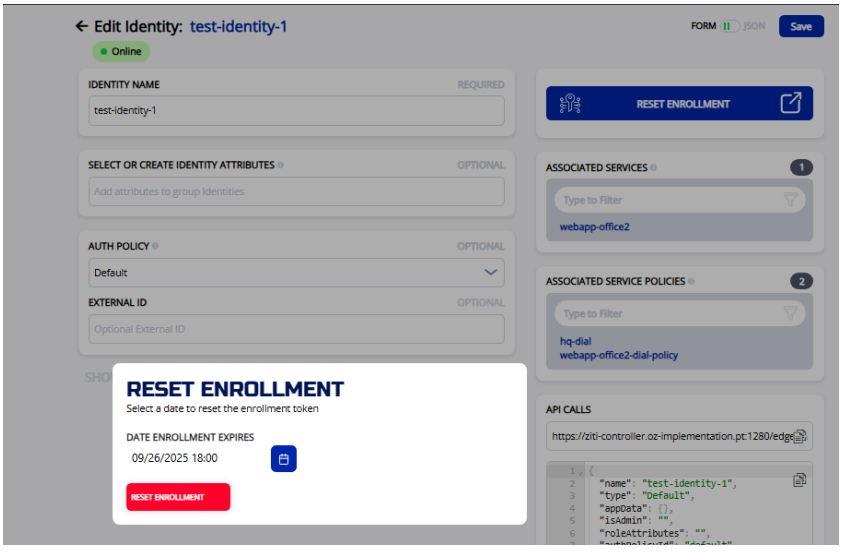


Figure 40: Resetting the enrolment of the identity

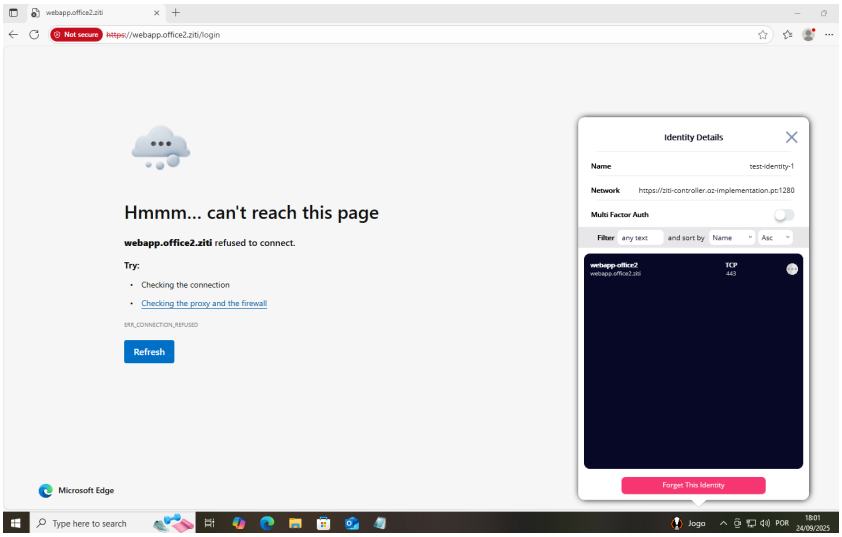


Figure 41: Attempt to access WAS #1 after revocation

4.2.5 *Test T05 – Service Policy Update Propagation*

1. **Objective:** To verify that policy changes are promptly propagated and enforced across the Ziti network.
2. **Procedure:** Modify the service policy to remove access to WAS #1 for a specific identity. Wait for policy propagation and reattempt access.
3. **Result:** Access to the service was denied immediately after the policy update, indicating that policy synchronisation across controllers and routers is effective.

Despite not being able to adequately demonstrate the propagation rate of policy updates, it was observed during the tests that in the implemented environment policy updates happened real-time, meaning that after updating a policy in the controller, any new/on-going session would be triaged against the new policy. However, the Windows OpenZiti desktop agent showed some latency updating the policy changes visually, meaning that despite all accesses were being compared against the new version of the policy, end users could still be seeing the "old policy". It was possible to fix this visual issue by restarting the agent's connection to the OpenZiti's controller.

4.2.6 *Test T06 – Unauthorised Service Discovery Prevention*

1. **Objective:** To confirm that users cannot discover or list services to which they are not authorised.
2. **Procedure:** Use a Ziti client identity configured without access to any defined services. Attempt to enumerate or probe service availability.
3. **Result:** No information about services was exposed. The client only recognises explicitly authorised services. This aligns with the Zero Trust principle of "deny by default".

4.2.7 *Test T07 – Logging and Audit Limitations*

1. **Objective:** To evaluate the granularity and visibility of logs generated by the OpenZiti Controller.
2. **Procedure:** Establish and terminate multiple sessions to different services and inspect the generated logs.

3. **Result:** Session logs included timestamped connection events with source identity, target service, and duration. However, application-layer events (e.g., file transfer, command execution) were not recorded. This indicates that while access events are auditable, deep user behaviour analytics (UEBA) would require integration with external tools. These logs are stored in a .log file, in JSON format, within each component, and there is no out-of-the-box capability to export them to a SIEM – the SIEM tool must be capable of retrieving these logs from the hosts.

Listing 13 represents a log snippet taken from the OpenZiti Controller's logs. As it can be observed, while it is possible to see events related to the authentication/authorisation process - namespace "connect" -, and the services accessed - namespace "usage" -, it is not possible to have a list of activities the user did in a determined service.

```
1 {"namespace":"entityCount","event_src_id":"NetFoundry Inc. Client
  ↪ 3V0W2Kj8s","timestamp":"2025-05-04T20:18:40.607818698Z","counts":{"apiSessionCertificat
  ↪ es":4,"apiSessions":5,"authPolicies":1,"authenticators":4,"cas":0,"configTypes":5,"conf
  ↪ igs":8,"controllers":0,"edgeRouterPolicies":5,"enrollments":0,"eventualEvents":0,"exter
  ↪ nalJwtSigners":1,"identities":9,"identityTypes":2,"mfas":1,"postureCheckTypes":5,"postu
  ↪ reChecks":0,"revocations":0,"routers":5,"routers.edge":5,"serviceEdgeRouterPolicies":1,
  ↪ "servicePolicies":7,"services":4,"services.edge":4,"sessions":13,"terminators":4}}
2 {"namespace":"connect","event_src_id":"NetFoundry Inc. Client
  ↪ 3V0W2Kj8s","timestamp":"2025-05-04T20:18:41.432386028Z","src_type":"identity","dst_type
  ↪ ":"ctrl","src_id":"ou8rXMtg-M","src_addr":"95.136.45.233:63408","dst_id":"NetFoundry
  ↪ Inc. Client 3V0W2Kj8s","dst_addr":"ziti-controller.oz-implementation.pt:1280"}
3 {"namespace":"connect","event_src_id":"NetFoundry Inc. Client
  ↪ 3V0W2Kj8s","timestamp":"2025-05-04T20:18:41.484598803Z","src_type":"identity","dst_type
  ↪ ":"ctrl","src_id":"ou8rXMtg-M","src_addr":"95.136.45.233:63408","dst_id":"NetFoundry
  ↪ Inc. Client 3V0W2Kj8s","dst_addr":"ziti-controller.oz-implementation.pt:1280"}
4 {"namespace":"connect","event_src_id":"NetFoundry Inc. Client
  ↪ 3V0W2Kj8s","timestamp":"2025-05-04T20:18:41.536276067Z","src_type":"identity","dst_type
  ↪ ":"ctrl","src_id":"ou8rXMtg-M","src_addr":"95.136.45.233:63408","dst_id":"NetFoundry
  ↪ Inc. Client 3V0W2Kj8s","dst_addr":"ziti-controller.oz-implementation.pt:1280"}
5 {"namespace":"usage","event_src_id":"NetFoundry Inc. Client
  ↪ 3V0W2Kj8s","timestamp":"2025-05-04T20:18:42.562040796Z","version":2,"event_type":"usage
  ↪ .fabric.tx","source_id":"EqBd9.YgG1","circuit_id":"t1AeCVs1A","usage":164,"interval_sta
  ↪ rt_utc":1746389820,"interval_length":60,"tags":null}
6 {"namespace":"usage","event_src_id":"NetFoundry Inc. Client
  ↪ 3V0W2Kj8s","timestamp":"2025-05-04T20:18:42.562035666Z","version":2,"event_type":"usage
  ↪ .ingress.tx","source_id":"EqBd9.YgG1","circuit_id":"t1AeCVs1A","usage":290,"interval_st
  ↪ art_utc":1746389820,"interval_length":60,"tags":{"clientId":"ou8rXMtg-M","hostId":"O7vw
  ↪ 7rMgR1","serviceId":"4moJCAs5bPvYudBkurF2fL"}}
7 {"namespace":"usage","event_src_id":"NetFoundry Inc. Client
  ↪ 3V0W2Kj8s","timestamp":"2025-05-04T20:18:42.562038262Z","version":2,"event_type":"usage
  ↪ .ingress.rx","source_id":"EqBd9.YgG1","circuit_id":"t1AeCVs1A","usage":164,"interval_st
  ↪ art_utc":1746389820,"interval_length":60,"tags":{"clientId":"ou8rXMtg-M","hostId":"O7vw
  ↪ 7rMgR1","serviceId":"4moJCAs5bPvYudBkurF2fL"}}
```

Listing 13: Snippet of OpenZiti Controller's logs

4.2.8 *Test T08 – Multi-Session Handling*

1. **Objective:** To test the system's ability to handle multiple simultaneous sessions from different identities.
2. **Procedure:** Initiate concurrent sessions from different client identities to separate services (e.g., WAS #1 and SAS #1).
3. **Result:** All sessions were independently established and managed without interference. Resource isolation and access enforcement remained consistent across sessions.

4.2.9 *Test T09 – Offline Identity Cache Behaviour*

1. **Objective:** To evaluate how the system behaves when the controller is unreachable during an identity's session.
2. **Procedure:** Establish a session to WAS #1, then disconnect the controller from the network. Attempt to continue using the session or reconnect.
3. **Result:** Sessions persisted for 1–2 minutes using locally cached policy tokens. After the token expired, reconnection attempts failed due to controller's unreachability.

4.2.10 *Test T10 – Identity Impersonation Protection*

1. **Objective:** To ensure that identity credentials cannot be reused or duplicated across unauthorised devices.
2. **Procedure:** Copy a valid identity token to an authorised device and attempt to access WAS #1.
3. **Result:** The access attempt was rejected. OpenZiti validated the device fingerprint and denied access due to mismatch, confirming protection against credential theft.

Figure 42, and the log file, listing 14, capture an attempt to authenticate against the controller using a duplicated identity token (JWT) on an unauthorised device. As illustrated, OpenZiti rejects the connection attempt, preventing the reuse of

stolen or copied credentials. As seen in the log file, the agent authenticates the JWT against the controller, receiving an "Invalid token" error from the controller. This confirms mitigation against impersonation attacks and credential theft scenarios.

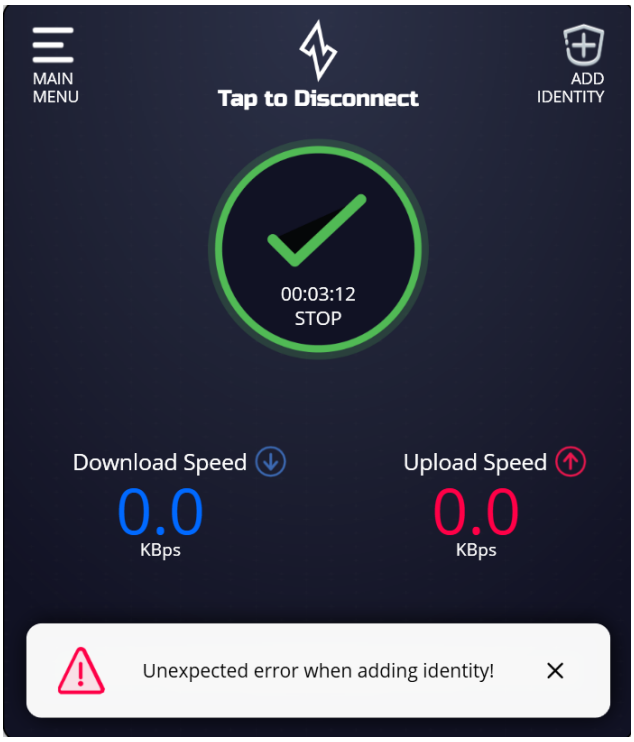


Figure 42: OpenZiti Desktop Edge client screenshot, representing the attempt to connect using a duplicated JWT

```
1 DEBUG ziti-sdk:ziti_ctrl.c:502 ctrl_body_cb()
  ↪ ctrl[https://ziti-controller.oz-implementation.pt:1280] completed GET[/version] in 0.171
  ↪ s
2 VERBOSE ziti-sdk:ziti_ctrl.c:200 ctrl_resp_cb()
  ↪ ctrl[https://ziti-controller.oz-implementation.pt:1280] received headers POST[/enroll]
3 VERBOSE ziti-sdk:ziti_ctrl.c:429 ctrl_body_cb()
  ↪ ctrl[https://ziti-controller.oz-implementation.pt:1280] HTTP RESPONSE:
  ↪ {"error":{"code":"INVALID_ENROLLMENT_TOKEN","message":"The supplied token is not valid"
  ↪ ,"requestId":"h196XjiT0"},"meta":{"apiEnrollmentVersion":"0.0.1","apiVersion":"0.0.1"}}
4 DEBUG ziti-sdk:ziti_ctrl.c:502 ctrl_body_cb()
  ↪ ctrl[https://ziti-controller.oz-implementation.pt:1280] completed POST[/enroll] in 0.221
  ↪ s
5 ERROR ziti-sdk:ziti_ctrl.c:524 ctrl_body_cb()
  ↪ ctrl[https://ziti-controller.oz-implementation.pt:1280] API request[/enroll] failed
  ↪ code[INVALID_ENROLLMENT_TOKEN] message[The supplied token is not valid]
6 ERROR ziti-sdk:ziti_enroll.c:424 enroll_cb() failed to enroll with controller:
  ↪ https://ziti-controller.oz-implementation.pt:1280 INVALID_ENROLLMENT_TOKEN[The supplied
  ↪ token is not valid] reason[]
7 ERROR ziti-edge-tunnel:process_cmd.c:60 tunnel_enroll_cb() enrollment failed: JWT not
  ↪ accepted by controller(-3)
```

```
8  TRACE ziti-edge-tunnel:ipc_cmd.c:139 on_command_resp() resp[0,len=28] =
   ↪ {"Success":false,"Code":500}
9  ERROR ziti-edge-tunnel:process_cmd.c:69 tunnel_enroll_cb() removing failed identity file:
   ↪  c:\windows\system32\config\systemprofile\appdata\roaming\netfoundry\test-identity-1.json
```

Listing 14: Snippet of OpenZiti Desktop Edge's logs

4.3 SUMMARY

This chapter described the structured evaluation of the OpenZiti-based ZTNA implementation through ten test cases, assessing access control, encryption, lateral movement, revocation, impersonation, offline behaviour, logging, and multi-session handling. Results demonstrated that OpenZiti enforces fine-grained, service-level access while preventing unauthorised discovery and impersonation, but also revealed limitations regarding east–west segmentation and audit depth.
The final chapter, Conclusion, concludes the thesis by synthesising the main findings, discussing limitations and proposed mitigations, and outlining directions for future work, particularly in standardisation, automation, and the broader adoption of open-source Zero Trust solutions.

5

CONCLUSION

5.1 ANALYSIS OF THE RESEARCH OBJECTIVES

The practical deployment of a Zero Trust Network Access (ZTNA) solution using the open-source OpenZiti platform has successfully validated the core principles of the Zero Trust model, demonstrating its applicability within real-world organisational settings. Throughout the implementation and testing phases, significant improvements were observed in access control granularity, operational visibility, and resilience to both internal and external threats, when compared to traditional VPN-based architectures.

The identity-centric architecture proved effective in reducing the attack surface, ensuring that only essential services are exposed through strict authentication and authorisation mechanisms. The conducted tests addressed critical security vectors, such as the prevention of lateral movement, the effectiveness of access revocation, service visibility, and protection against man-in-the-middle (MiTM) attacks.

This study's outcomes can be explicitly mapped to the research objectives defined in Chapter 1. The first objective, to conduct a comparative analysis of commercial and open-source ZTNA/SASE solutions, was achieved through the vendor feature matrix and critical review in Chapter 3, which positioned OpenZiti as a cost-effective but functionally narrower alternative to subscription-based platforms. The second objective, to design an SME-tailored implementation framework, was realized by developing a hybrid architecture combining on-premises edge routers with a cloud-hosted controller, balancing resilience with affordability. The third objective, to deploy and evaluate an OpenZiti-based architecture, was addressed through the step-by-step implementation across two SME-like sites and its integration with Okta and Active Directory. Finally, the fourth objective, to empirically assess security and operational outcomes, was fulfilled through structured tests validating policy enforcement, encryption, revocation, and service visibility, while also revealing limitations in east–west segmentation and identity lifecycle management. Collectively, these results confirm that the research objectives were not only addressed but also contributed new empirical evidence on the viability and constraints of opensource ZTNA in SME environments.

5.2 FUTURE WORK

Although the OpenZiti solution lacks native User and Entity Behaviour Analytics (UEBA) and advanced contextual controls, it still provides a substantially enhanced security posture, particularly when integrated with Identity and Access Management (IAM) and Identity Governance and Administration (IGA) tools. However, several limitations of the adopted solution must be acknowledged. Notably, the absence of native integration with risk-based engines such as SIEM, along with incomplete support for protocols like OIDC and SAML, present challenges in terms of scalability and user experience. Moreover, the lack of traffic inspection features (e.g., Data Loss Prevention) reduces its capabilities, which are typically embedded within full SASE solutions.

Several technical strategies can address these limitations. The absence of native Data Loss Prevention (DLP) and SSL decryption can be mitigated through integration with external DLP solutions via API-based policy enforcement, and implementation of host-based DLP agents on endpoint devices. Current federation limitations regarding incomplete SAML/OIDC support can be addressed by implementing custom authentication adapters using OpenZiti's REST API.

The observed east-west traffic vulnerabilities that enable lateral movement within LANs require complementary controls, including implementation of host-based micro-segmentation using software-defined perimeters on endpoints and integration with Software-Defined Networking (SDN) controllers for automated network isolation. Finally, the current logging limitations can be overcome by installing SIEM's agents to pull the logs directly from each OpenZiti's component.

Regarding scalability, the current two-site implementation, representing a SME, can scale by deploying new edge routers (preferably clusters of edge routers for high availability and load distribution) in new locations, taking advantage of a cloud provider's infrastructure, for example. Despite these constraints, and considering the suggested solutions, OpenZiti emerges as a compelling alternative for organisations operating under budgetary limitations. While it demands a relatively high level of technical expertise for deployment and ongoing maintenance, it nonetheless offers a robust, flexible, and cost-effective ZTNA implementation.

In conclusion, the study confirms that implementing ZTNA using open-source technologies is not only technically feasible but delivers measurable security improvements. The transition from perimeter based to Zero Trust security requires fundamental changes in security thinking, operational processes, and organizational culture. As demonstrated through our implementation, the benefits significantly

outweigh the challenges, particularly as solutions for current limitations become available. Future work should focus on standardizing ZTNA implementations, developing automated management tools, and creating industry-specific deployment templates. As Zero Trust evolves from an emerging paradigm to an operational necessity, continued research and development in open-source implementations will democratize access to advanced security architectures, enabling organizations of all sizes to protect their digital assets effectively. This transition signals a paradigm shift where identity, not location, becomes the cornerstone of enterprise security.

BIBLIOGRAPHY

Akamai Technologies (2024). *A Blueprint for Zero Trust Network Access*. White Paper. URL: https://www.akamai.com/site/en/documents/white-paper/2024/a-blueprint-for-zero-trust-network-access.pdf.

Bradatsch, L. et al. (2024). "Zero Trust Score-Based Network-Level Access Control in Enterprise Networks". In: *arXiv preprint*. eprint: 2402.08299. URL: https://arxiv.org/abs/2402.08299.

Colomo-Palacios, R., M. N. Islam, and S. Chockalingam (2021). "Secure Access Service Edge: A Multivocal Literature Review". In: *Proceedings of the 21st International Conference on Computational Science and Applications (ICCSA)*. Cagliari, Italy: Springer, pp. 188–194. DOI: 10.1007/978-3-030-87013-2_14.

Defined Networking Corp. (2024). *Nebula Technical Architecture: Certificate-Based Mesh Networking*. Technical White Paper. URL: https://nebula.defined.net/docs/.

Donenfeld, J. A. (2017). "WireGuard: Next Generation Kernel Network Tunnel". In: *Proc. 24th Annu. Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, USA. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/.

Kang, H. et al. (2023). "Theory and Application of Zero Trust Security: A Brief Survey". In: *Entropy* 25.11, p. 1595. DOI: 10.3390/e25111595.

Palo Alto Networks (2025). *What Is Zero Trust Network Access (ZTNA)*. URL: https://www.paloaltonetworks.com/cyberpedia/what-is-zero-trust-network-access-ztna.

Rose, S. et al. (2020). *Zero Trust Architecture*. Special Publication 800-207. Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST). DOI: 10.6028/NIST.SP.800-207. URL: https://doi.org/10.6028/NIST.SP.800-207.

Sarkar, S. et al. (2022). "Security of Zero Trust Networks in Cloud Computing: A Comparative Review". In: *Sustainability* 14.18, p. 11213. DOI: 10.3390/su141811213.

Syed, N. F. et al. (2022). "Zero Trust Architecture (ZTA): A Comprehensive Survey". In: *IEEE Access* 10, pp. 57143–57179. DOI: 10.1109/ACCESS.2022.3177114.

Tailscale Inc. (2024). *Tailscale Documentation: Access Control Lists*. URL: https://tailscale.com/kb/1018/acls.

Verizon Business (2024). *2024 Data Breach Investigations Report (DBIR)*. Tech. rep. Verizon Business. URL: https://www.verizon.com/business/resources/reports/dbir/.

Yiliyaer, S. and Y. Kim (2022). "Secure Access Service Edge: A Zero Trust Based Framework for Accessing Data Securely". In: *Proc. IEEE 12th Annu. Comput. Commun. Workshop Conf. (CCWC)*. Las Vegas, NV, USA: IEEE, pp. 586–591. DOI: 10.1109/CCWC54503.2022.9720859.

ZeroTier Inc. (2024). *ZeroTier Manual: Technical Documentation*. URL: https://docs.zerotier.com/.

APPENDICES

APPENDIX A

This appendix presents the full configuration files for both the OpenZiti Controller and OpenZiti Routers.

```yaml
v: 3
db:                     "/var/lib/private/ziti-controller/bbolt.db"
identity:
  cert:        "pki/intermediate/certs/client.chain.pem"
  server_cert: "pki/intermediate/certs/server.chain.pem"
  key:         "pki/intermediate/keys/server.key"
  ca:          "pki/root/certs/root.cert"
ctrl:
  options:
    advertiseAddress: tls:ziti-controller.oz-implementation.pt:1280
  listener:             tls:0.0.0.0:1280
healthChecks:
  boltCheck:
    interval: 30s
    timeout: 20s
    initialDelay: 30s
edge:
  api:
    sessionTimeout: 30m
    address: ziti-controller.oz-implementation.pt:1280
  enrollment:
    signingCert:
      cert: pki/intermediate/certs/intermediate.cert
      key:  pki/intermediate/keys/intermediate.key
    edgeIdentity:
      duration: 180m
    edgeRouter:
      duration: 180m
events:
  jsonLogger:
    subscriptions:
      - type: circuit
      - type: connect
      - type: link
      - type: router
      - type: terminator
      - type: metric
        sourceFilter: .*
        metricFilter: .*
      - type: session
      - type: apiSession
      - type: usage
```

```
      - type: service
      - type: entityCount
        interval: 5s
    handler:
      type: file
      format: json
      path: /tmp/ziti-events.log
web:
  - name: client-management
    bindPoints:
      - interface: 0.0.0.0:1280
        address: ziti-controller.oz-implementation.pt:1280
    identity:
      ca:          "pki/root/certs/root.cert"
      key:         "pki/intermediate/keys/server.key"
      server_cert: "pki/intermediate/certs/server.chain.pem"
      cert:        "pki/intermediate/certs/client.chain.pem"
    options:
      idleTimeout: 5000ms
      readTimeout: 5000ms
      writeTimeout: 100000ms
      minTLSVersion: TLS1.2
      maxTLSVersion: TLS1.3
    apis:
      - binding: edge-management
        options: { }
      - binding: edge-client
        options: { }
      - binding: fabric
        options: { }
      - binding: edge-oidc
        options: { }
      - binding: zac
        options:
          location: /opt/openziti/share/console
          indexFile: index.html
```

Listing 15: OpenZiti Controller's config.yml file

```
v: 3
identity:
  cert: /opt/openziti/ziti-router/certs/cert.pem
  server_cert: /opt/openziti/ziti-router/certs/server_cert.pem
  key: /opt/openziti/ziti-router/certs/key.pem
  ca: /opt/openziti/ziti-router/certs/ca.pem
ctrl:
  endpoint: tls:ziti-controller.oz-implementation.pt:1280
link:
  dialers:
    - binding: transport
  listeners:
    - binding: transport
      bind: tls:0.0.0.0:80
      advertise: tls:pub-er-1.oz-implementation.pt:80
```

```
      options:
        outQueueSize: 16
edge:
  heartbeatIntervalSeconds: 60
  csr:
    country: US
    province: NC
    locality: Charlotte
    organization: NetFoundry
    organizationalUnit: Ziti
    sans:
      dns:
        - localhost
        - pub-er-1.oz-implementation.pt
      ip:
        - 4.231.233.98
        - 127.0.0.1
listeners:
  - binding: edge
    address: tls:0.0.0.0:443
    options:
      advertise: pub-er-1.oz-implementation.pt:443
```

Listing 16: OpenZiti Edge Routers's config.yml file

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Implementation and Analysis of a ZTNA Solution"*, é original e foi realizado por Bernardo Duarte Rodrigues Fragoso de Rhodes (2230473) sob orientação de Professor Doutor Paulo Jorge Gonçalves Loureiro (paulo.loureiro@ipleiria.pt) e Professor Doutor Sílvio Priem Mendes (smendes@ipleiria.pt).

Leiria, Setembro de 2025

Bernardo Duarte Rodrigues Fragoso de Rhodes