Check for updates

SOFTWARE TOOL ARTICLE

A decentralized zero-touch provisioning framework for secure and scalable edge infrastructure deployment

[version 1; peer review: 1 approved, 1 approved with reservations]

Stefan-Cristian Jarcau[1], Mihnea-Ionut Lopataru[1], Vlad Alexandru Mocanu[1], Gabriel-Mihail Danciu[1], Cosmin-Septimiu Nechifor[1], Ioanna Angeliki Kapetanidou[2], Alexandros Nizamis[2], Usman Wajid[3]

[1]R&D, Siemens SRL, Brasov, Brasov, 500097, Romania
[2]R&D, Information Technologies Institute, Centre for Research and Technology Hellas, Thermi, Thessaloniki, 57001, Greece
[3]Information Catalyst, Northwich, Cheshire, CW8 4EE, UK

v1 **First published:** 10 Nov 2025, **5**:344
https://doi.org/10.12688/openreseurope.21630.1
Latest published: 10 Nov 2025, **5**:344
https://doi.org/10.12688/openreseurope.21630.1

Abstract

Zero-touch provisioning (ZTP) is a key enabler for scalable deployment of edge infrastructure across heterogeneous networks. This paper presents a decentralized ZTP framework developed as part of the ENACT project, designed to automate the discovery, configuration, and integration of remote devices without requiring direct access or manual intervention. Built on open standards and widely used technologies—including PXE, DHCP, Cloud-Init, and REST APIs—the system minimizes administrative overhead while maintaining strong security guarantees through a custom authentication mechanism using WebSockets and one-time install-time verification. The architecture separates concerns between a central server, which interfaces with the orchestrator, and lightweight agents deployed in remote networks that manage provisioning via Ubuntu MAAS. This approach avoids exposing remote environments directly to the internet, reducing attack surface and simplifying coordination across multiple sites. Evaluation results demonstrate significant improvements in both deployment speed (94% time reduction for bare metal deployment and 60% for VM) and consistency compared to manual provisioning processes, validating the effectiveness of the automated approach for large-scale edge infrastructure deployment.

Keywords

Zero-touch provisioning, edge computing, automated deployment, cloud-init, network orchestration, Ubuntu MAAS, decentralized architecture, WebSocket authentication.

Open Peer Review

Approval Status ✓ ?

	1	2
version 1 10 Nov 2025	✓ view	? view

1. **Marco Gramaglia**, Universidad Carlos III de Madrid, Getafe, Spain

2. **Matthijs Jansen**, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Any reports and responses or comments on the article can be found at the end of the article.



This article is included in the Horizon Europe gateway.

Corresponding author: Stefan-Cristian Jarcau (stefan.jarcau@siemens.com)

Author roles: Jarcau SC: Investigation, Methodology, Software, Validation, Writing – Original Draft Preparation; **Lopataru MI**: Writing – Original Draft Preparation; **Mocanu VA**: Writing – Original Draft Preparation; **Danciu GM**: Supervision, Writing – Original Draft Preparation, Writing – Review & Editing; **Nechifor CS**: Supervision, Writing – Review & Editing; **Kapetanidou IA**: Supervision, Writing – Review & Editing; **Nizamis A**: Conceptualization, Funding Acquisition, Supervision, Writing – Review & Editing; **Wajid U**: Funding Acquisition, Supervision, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This project has received fundings from European Union's Horizon Europe research and innovation programme under grant agreement No [101135423] – (project name [ENACT project]).
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2025 Jarcau SC *et al*. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Jarcau SC, Lopataru MI, Mocanu VA *et al.* **A decentralized zero-touch provisioning framework for secure and scalable edge infrastructure deployment [version 1; peer review: 1 approved, 1 approved with reservations]** Open Research Europe 2025, **5**:344 https://doi.org/10.12688/openreseurope.21630.1

First published: 10 Nov 2025, **5**:344 https://doi.org/10.12688/openreseurope.21630.1

Introduction

The deployment of edge computing infrastructure across distributed and heterogeneous networks introduces operational challenges, particularly in the configuration and integration of remote devices. Traditional provisioning workflows often require manual setup, inbound SSH access, and exposure of internal systems—practices that increase security risks and hinder scalability.

Zero-touch provisioning (ZTP) addresses these challenges by enabling devices to be discovered, configured, and integrated automatically without operator intervention. While ZTP is common in centralized data center environments, its application in fragmented edge settings—where networks may be isolated or constrained—is far less explored.

This paper presents a decentralized ZTP framework developed within the ENACT project[1,2], designed to securely automate the provisioning of remote infrastructure across federated networks. The architecture consists of a central server that coordinates lightweight agents deployed in remote sites, each managing local provisioning through Ubuntu MAAS and Cloud-Init. A secure, install-time authentication mechanism based on WebSocket communication and PBKDF2 ensures that no persistent credentials are required.

Our contributions are threefold:

- The design of a secure, decentralized ZTP architecture that avoids inbound exposure.
- The integration of standards-compliant provisioning tools in constrained networks.
- The evaluation of the proposed solution compared to manual deployment when it comes with speed and consistency of the deployment process.

The remainder of this paper is structured as follows: Section 2 presents the state of the art in this domain. Section 3 describes the overall architecture. Section 4 details the technical implementation. Section 5 outlines pilot evaluations. Section 6 discusses key insights, and Section 7 concludes the paper.

Related work

The landscape of zero-touch provisioning solutions spans multiple domains, from traditional enterprise network management to modern cloud-native orchestration platforms. The evolution of Zero Touch Service Management (ZSM) has been particularly significant in telecommunications, where[3] provide a comprehensive survey of ZSM approaches for 5G and beyond networks, emphasizing the critical need for automated provisioning and management in increasingly complex network infrastructures. However, their focus on telecommunications-specific protocols and service orchestration differs from our emphasis on general-purpose edge infrastructure deployment.

Several commercial and open-source ZTP solutions exist in the cloud-native space, including Cisco Network Services Orchestrator (NSO)[4], Red Hat Ansible Automation Platform (formerly Ansible Tower)[5], and proprietary cloud-based onboarding platforms such as AWS IoT Greengrass[6] and Azure IoT Edge[7]. However, these solutions typically assume persistent, high-bandwidth cloud connectivity or require complex VPN tunnels and firewall configurations, which may not be feasible or secure in isolated edge environments where network partitions are common and inbound access is restricted for security reasons.

Traditional provisioning frameworks like Foreman[8], Cobbler[9], and PXE-based solutions[10] focus primarily on bare-metal deployment within controlled data center environments. While these tools provide robust hardware discovery and OS installation capabilities, they lack the distributed coordination mechanisms necessary for federated edge deployments and typically require centralized management servers with direct network access to target devices.

In academic literature, several agent-based strategies have been proposed for disconnected orchestration and autonomous infrastructure management. 11 survey various edge computing orchestration approaches, highlighting the challenges of managing distributed infrastructure with intermittent connectivity. Similarly, research on fog computing orchestration[12] has explored decentralized coordination patterns, but these works primarily focus on application-level orchestration rather than infrastructure provisioning.

Recent work on container orchestration at the edge, such as KubeEdge[13] and OpenYurt[14], addresses some aspects of distributed management but assumes pre-existing Kubernetes clusters and does not address the fundamental challenge of initial device provisioning and OS deployment. These platforms excel at managing containerized workloads across edge nodes but rely on manual or external tools for the initial infrastructure setup.

Our approach differs fundamentally by embedding provisioning logic directly within the local network while maintaining centralized orchestration capabilities. This hybrid model combines the benefits of local autonomy—reducing dependency on central infrastructure and improving resilience to network partitions—with the operational simplicity of centralized control. The use of persistent WebSocket connections initiated from edge agents eliminates the need for inbound firewall rules while maintaining real-time bidirectional communication, a design pattern that has been explored in IoT device management[15] but not extensively applied to infrastructure provisioning.

While similar agent-based strategies have been proposed in academic literature for disconnected orchestration[16], few integrate fully with standards-compliant PXE and MAAS workflows or leverage modern WebSocket protocols for secure state coordination. The integration of Ubuntu MAAS[17] as the local provisioning backend distinguishes our approach from lighter alternatives, providing comprehensive hardware discovery, power management integration via IPMI, and declarative configuration management through Cloud-Init[18].

Security considerations also differentiate our framework from existing solutions. Traditional ZTP implementations often rely on pre-shared keys, certificate-based authentication, or persistent credentials stored on devices. Our one-time proof exchange mechanism using PBKDF2[19] eliminates the need for long-lived credentials while ensuring secure agent registration. This approach addresses the key management challenges identified in edge security research[20] and provides a more robust foundation for large-scale deployments.

Evaluation of our framework through real-world deployments demonstrates significant improvements in deployment speed and consistency compared to manual provisioning methods. The automated ZTP approach reduces deployment time from hours to minutes while eliminating human error and configuration drift that commonly plague manual processes. Empirical results show consistent, repeatable deployments across diverse hardware configurations and network constraints, providing practical validation that extends beyond the simulation-based assessments common in academic literature.

System architecture

Design overview

The Zero-Touch Provisioning framework adopts a decentralized architecture to address the limitations of traditional centralized provisioning in distributed and security-constrained environments. The system consists of a central ZTP server, which resides within the core cloud infrastructure and interfaces with the orchestrator, and lightweight ZTP agents deployed across remote networks. Each ZTP agent operates autonomously within its respective network, managing local provisioning through Ubuntu MAAS and Cloud-init.

Communication between the ZTP agents and the central server is established using a persistent WebSocket channel initiated from the agent, allowing secure bidirectional messaging without requiring open ports on the remote side. Meanwhile, the orchestrator communicates with the ZTP server using standard REST APIs to retrieve agent status and initiate provisioning workflows.

This separation of responsibilities ensures that sensitive networks are not required to expose internal services or accept inbound connections. Overall, this ZTP architecture ensures clear network boundaries and minimizes the attack surface by eliminating the need for credential sharing or persistent SSH access. The system is built entirely on well-established technologies, including PXE booting for device discovery, MAAS for deployment, and Cloud-init for configuration management[10,18]. Figure 1 illustrates the high-level architecture of the ZTP framework.

Component responsibilities

The ZTP framework is composed of three primary components: the ZTP server, ZTP agents, and the orchestrator. Each plays a distinct role in coordinating provisioning operations across distributed environments while maintaining clear network boundaries.

- **ZTP Server**. The central server is deployed within the CCC and exposes a REST API to the orchestrator and a WebSocket interface to remote agents. It functions as a control hub, maintaining a registry of active agents, handling secure authentication during agent onboarding, and issuing provisioning commands based on orchestrator requests. The server is implemented in Go, leveraging its lightweight concurrency model to manage multiple simultaneous agent connections efficiently. Figure 2 presents the overall architecture of the ZTP server as it connects to the agents and to the orchestrator. The server offers flexible deployment options, including a standard binary that can be run as a service, a Docker container, or a Kubernetes cluster.

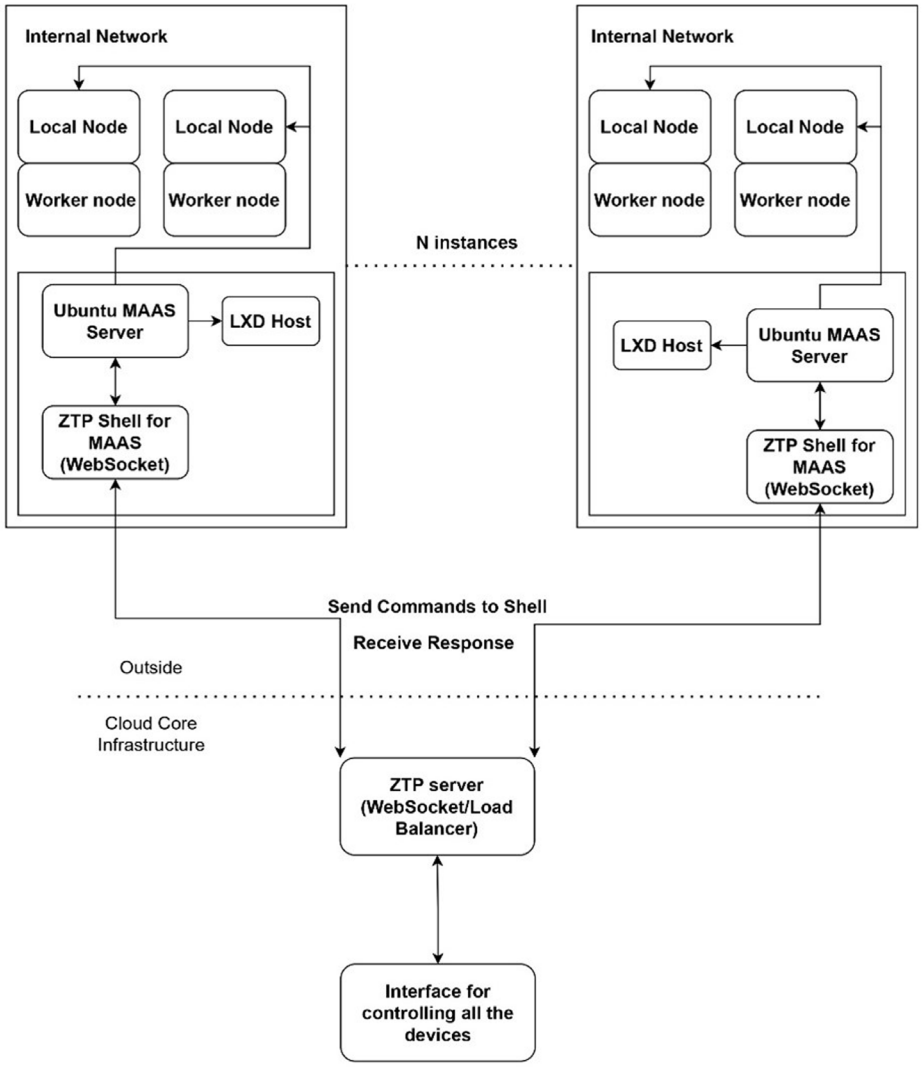

Figure 1. ZTP architecture overview. The central ZTP server interacts with the orchestrator via REST APIs and with remote ZTP agents via WebSocket. Each agent manages a local MAAS instance to provision devices using PXE boot and Cloud-init.

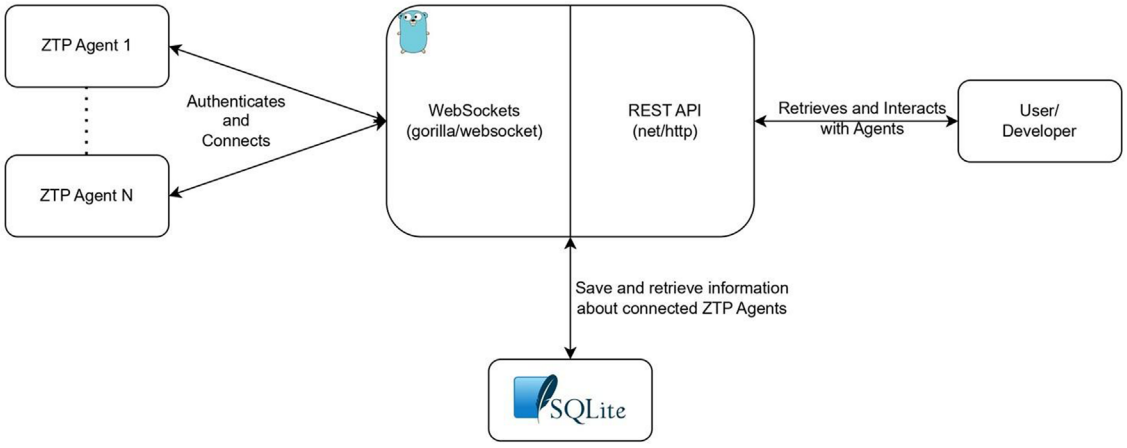

Figure 2. ZTP authentication flow. Agents establish secure WebSocket sessions with the server. The server uses REST APIs to expose agent metadata to the orchestrator and persists state via a local database.

- **ZTP Agent**. Each remote network hosts a ZTP agent responsible for local provisioning. The agent initiates a persistent WebSocket connection to the ZTP server and authenticates using a one-time proof exchange mechanism. Once registered, it listens for commands and translates them into API calls to a co-located Ubuntu MAAS instance. The agent also supervises system configuration using Cloud-init manifests and reports device provisioning status back to the server. The agent is implemented as a systemd-managed Python daemon to ensure reliable startup and reconnection after failures or reboots.

- **Ubuntu MAAS and Cloud-init**. Within each remote site, Ubuntu MAAS handles bare-metal discovery via PXE booting and operating system installation. Once the OS is deployed, Cloud-init performs system configuration, installing required services (e.g., Kubernetes components) and registering the node with the remote cluster. This separation ensures provisioning logic remains declarative and reproducible[18].

- **Orchestrator**. The orchestrator is an external component that interacts only with the ZTP server via REST. It retrieves the current state of available agents and devices and can trigger provisioning workflows in response to application deployment requests. This separation of concerns ensures that orchestration logic remains abstracted from the internal structure of each network.

Implementation

The provisioning process begins with the installation of a ZTP agent on a hardened virtual or physical host within the target network. During setup, the agent performs a security baseline check using tools such as Lynis[21], ensuring that the host system meets baseline security criteria. Upon startup, the agent initializes a secure WebSocket session with the ZTP server and transmits a preconfigured registration token along with system metadata (hostname, IP, hardware fingerprint) to request inclusion.

Authentication is performed during this initial registration using a one-time password (OTP) generated via PBKDF2 algorithm, as shown in Figure 3, as defined in RFC 8018[19], and verified over the WebSocket channel. The PBKDF2 function was initialized using 32 bytes salt length, generated randomly, and 4096 iterations, producing a 32 bytes length OTP. This process uses a cryptographic hash derived from system hardening output and machine identity, validated on both ends before the agent is accepted, as illustrated in Figure 4. The handshake ensures that no long-lived credentials are stored either on the agent or server, and all communication is encrypted using TLS 1.3[22]. The simplified registration process is shown in Figure 4.

Once authenticated, the agent downloads MAAS configuration files, installs dependencies (Ubuntu MAAS, DHCP helper, PXE services), and launches a local provisioning stack. Configuration is driven by a declarative YAML profile obtained from the server. The agent then deploys a local instance of Ubuntu MAAS, which is configured to discover bare-metal devices on the same subnet via PXE booting[10]. MAAS handles discovery of bare-metal or VM hosts via PXE and IPMI, interfacing through its internal API. The system supports both virtual machine provisioning via LXD and bare-metal provisioning targeting physical devices, as demonstrated in Figure 5.

```
# Server side (pseudo-code)
BEGIN
      salt = GenerateRandomSalt(32) // Generate a random Salt of 32 bytes
       // Derive a secret key using PBKDF2 with:
       //     - input: machineID
       //     - salt: salt
       //     - iterations: 4096
       //     - key length: 32 bytes
       secretKeyBytes, secretKey = PBKDF2(machineID, salt, 4096, 32)

       // Prepare the response as a key-value map
       response = {
            "secret_key": secretKey
       }

       // Send the response as JSON through the connection
       IF SendJSON(conn, response) fails THEN
            Log("Error sending secret key")
            RETURN
        END IF
END
```

Figure 3. Simplified agent registration using PBKDF2 token.

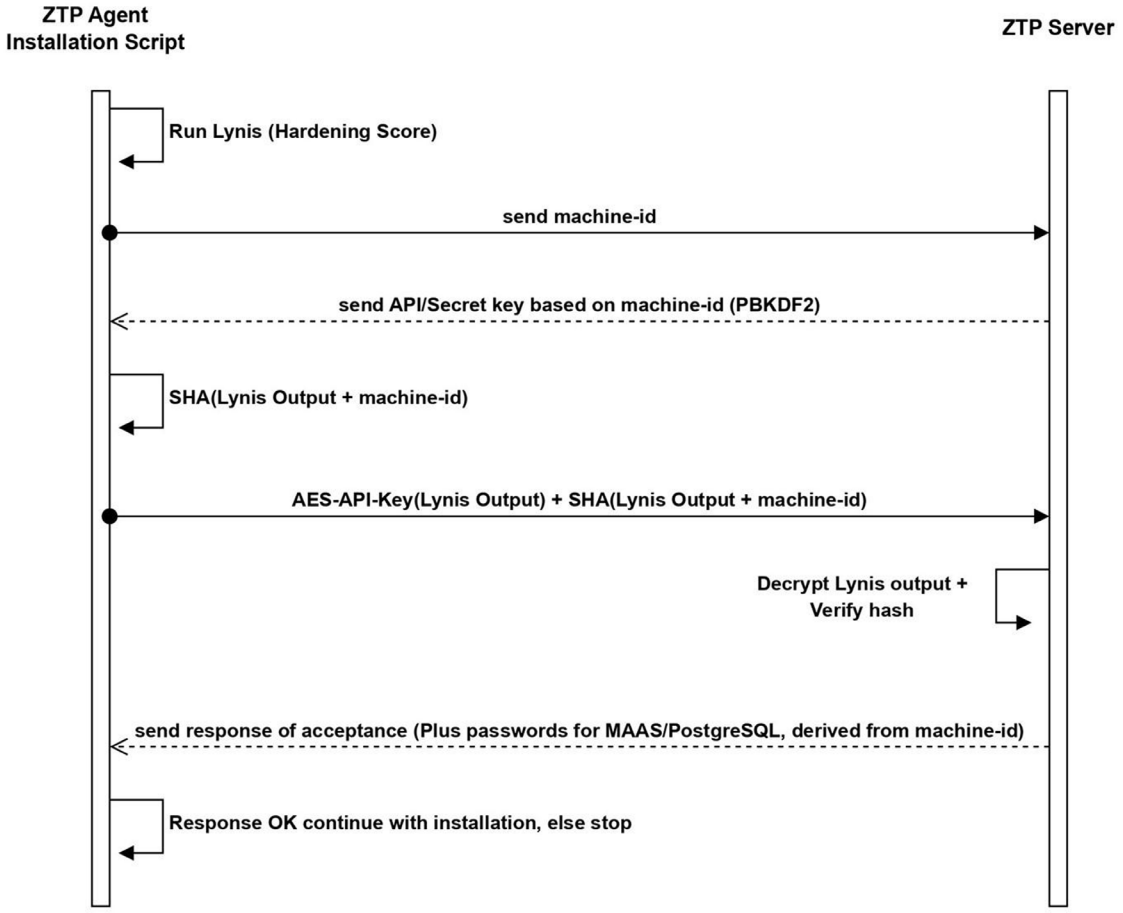

Figure 4. ZTP authentication flow between agent and server. A one-time proof exchange derived from system hardening output and machine identity is used to securely register agents via a WebSocket connection.

When a new device is detected, MAAS installs a base operating system image and triggers a Cloud-init script. These pre-seeded scripts configure system services, install container runtimes and Kubernetes components, define system roles, and register the device as a worker node under the local cluster master. Throughout this process, the ZTP agent maintains communication with the server, reporting provisioning status, system metrics, and telemetry updates to enable orchestrator-wide visibility.

The orchestrator periodically queries the ZTP server to retrieve the list of available agents and devices. When a deployment is requested, the orchestrator issues provisioning commands via the server's REST API. These commands are digitally signed and timestamped by the server before dispatch, then routed to the appropriate agent, which executes the necessary actions through MAAS and Cloud-init. WebSocket channels are monitored for idle timeout and replay protection, with control commands requiring proper authentication.

This asynchronous, pull-based communication model—where agents initiate and maintain connections—eliminates the need for inbound access and simplifies firewall traversal. It ensures that all provisioning tasks are handled locally while maintaining centralized control and observability.

Evaluation and use cases

Evaluation methodology

The evaluation of the Zero-Touch Provisioning services focused on validating system architecture, deployment workflows, and agent-server communication in representative environments. The evaluation prioritized implementation correctness, interoperability, and readiness for integration with real-world applications and use cases from various interest areas.

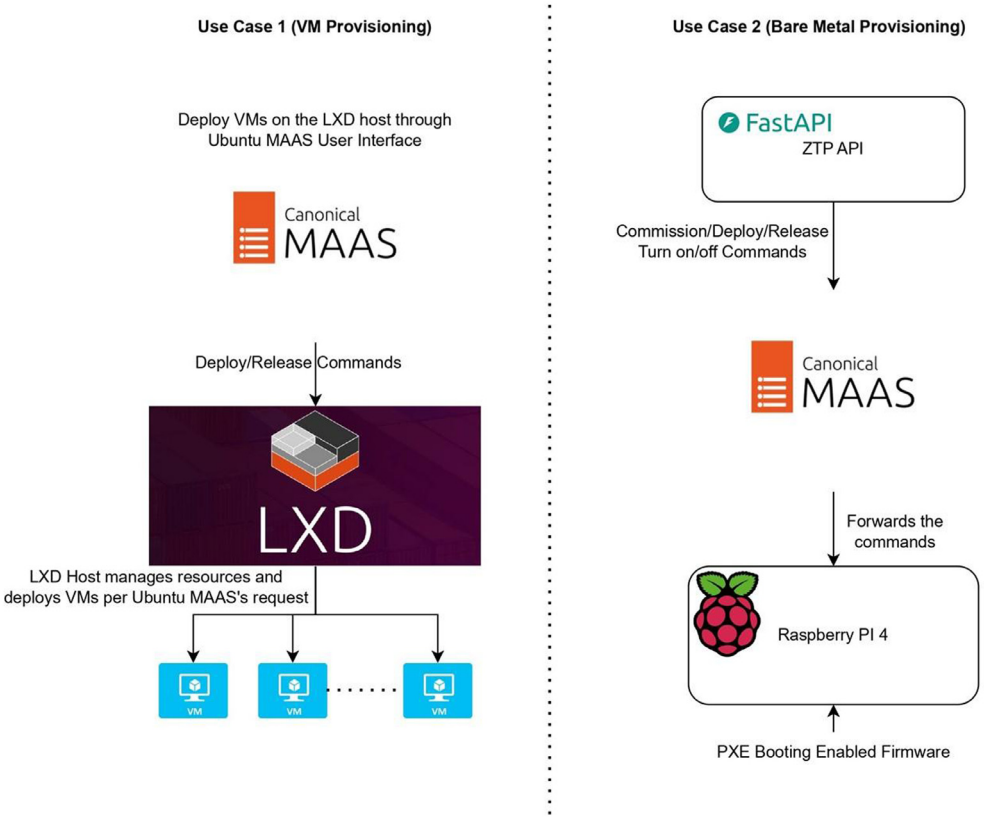

Figure 5. ZTP provisioning workflows. Left: Ubuntu MAAS provision virtual machines via LXD on local infrastructure. Right: Bare-metal provisioning uses PXE booting and ZTP-issued API commands to MAAS, targeting physical devices such as Raspberry Pi units.

Tests were conducted in a simulated multi-network setup, comprising:

- One central ZTP Server instance, co-located with the ENACT Orchestrator in a main network.
- Two remote networks, each hosting a ZTP Agent and a set of provisionable devices (either PXE-capable Raspberry Pi 4 boards or virtualized LXD instances[23]).

The main objectives of the evaluation are:

- To validate secure agent authentication and registration via the WebSocket-based handshake protocol[19,24];
- To verify that agents could reliably discover, provision, and configure new devices using Ubuntu MAAS[17] and Cloud-Init[18];
- To assess the ability of the ZTP Server to monitor and interact with all connected agents in a decentralized topology;
- To ensure compatibility with ENACT's orchestration layer for dynamic workload deployment.

Given the ongoing nature of development, most results were verified manually through log inspection, MAAS UI feedback, and device state monitoring. Future plans involve automating these test processes and incorporating benchmarking tools such as Prometheus and the ELK stack[25] to measure provisioning latency and operational throughput.

Provisioning validation

Provisioning validation was performed by simulating realistic deployment workflows using the ZTP Agent and Server in controlled remote networks. Each agent was responsible for managing a local Ubuntu MAAS instance, which performed PXE boot, OS deployment, and configuration of devices using Cloud-Init scripts[10,18].

Two types of devices were provisioned:

- Virtual Machines (VMs): LXD-based virtual machines were provisioned using MAAS and validated through the MAAS web UI and API[17,23]. ZTP Agents applied Cloud-Init manifests to assign roles and preconfigure required services.
- Bare-metal Devices: PXE-capable Raspberry Pi 4 boards were used to verify diskless provisioning workflows. Agents managed PXE configuration, image deployment, and triggered remote power control via MAAS-integrated IPMI interfaces.

Observed outcomes included:

- Devices appeared correctly in the MAAS inventory after discovery and commissioning.
- Cloud-Init scripts executed expected configurations, such as Kubernetes installation and service registration[26].
- Reboots and retries did not require operator intervention, as agents maintained persistent WebSocket connections and reconnected automatically upon network recovery[24].
- Functional validation demonstrated the correctness and repeatability of the provisioning process across both virtualized and physical hardware. Troubleshooting logs and systemd journal outputs were inspected during all tests to confirm expected behavior.

1. Scenario 1: Virtual Machine Deployment

One of the key test scenarios involved deploying virtual machines within a remote network managed by a ZTP Agent. In this setup, the agent installed and configured an Ubuntu MAAS instance capable of provisioning LXD-based VMs using MAAS's native virtualization drivers[17,23]. Figure 6 illustrates the MAAS interface of some deployed VMs as well the LXD view of the VMs at the bottom.

The test simulated a typical deployment workflow:

- The ZTP Agent registered with the central ZTP Server and established a persistent WebSocket channel for remote orchestration.
- The MAAS instance discovered the hypervisor and enabled virtual machine creation via MAAS API calls.
- A set of LXD VMs were provisioned automatically using Cloud-Init manifests tailored to specific application roles.
- Each VM was then preconfigured with Kubernetes components using lightweight K3s installers[26].

This scenario serves as a foundational building block for scenarios where edge services must be rapidly instantiated within software-defined environments.

The test validated:

- End-to-end communication from Orchestrator to remote MAAS via ZTP components
- Agent-triggered provisioning and remote status updates
- Feasibility of bootstrapping distributed clusters from minimal VM definitions.

Future iterations will include automatic registration of provisioned VMs into ENACT's telemetry layer, enabling performance and health metrics to be collected directly after deployment[25].

2. Scenario 2: Bare-Metal Deployment

To validate bare-metal provisioning, the ZTP Agent was deployed in a remote network containing PXE-capable Raspberry Pi 4 devices. These devices operated without local storage (SD-less configuration) and relied entirely on remote boot and provisioning services provided by Ubuntu MAAS[10,17]. Raspberry Pi 4 devices require a network-boot-enabled EEPROM (Electrically Erasable Programmable Read-Only Memory) to support PXE booting. In this setup, the EEPROM version 1.38 from https://github.com/pftf/RPi4/releases was used. The shell script provided in 27 was utilized to flash the EEPROM onto a USB stick, which is then inserted into the Raspberry Pi before initiating the boot process. Figure 7 illustrates the Raspberry Pi as it is getting the IP address and configurations from the MAAS instance.

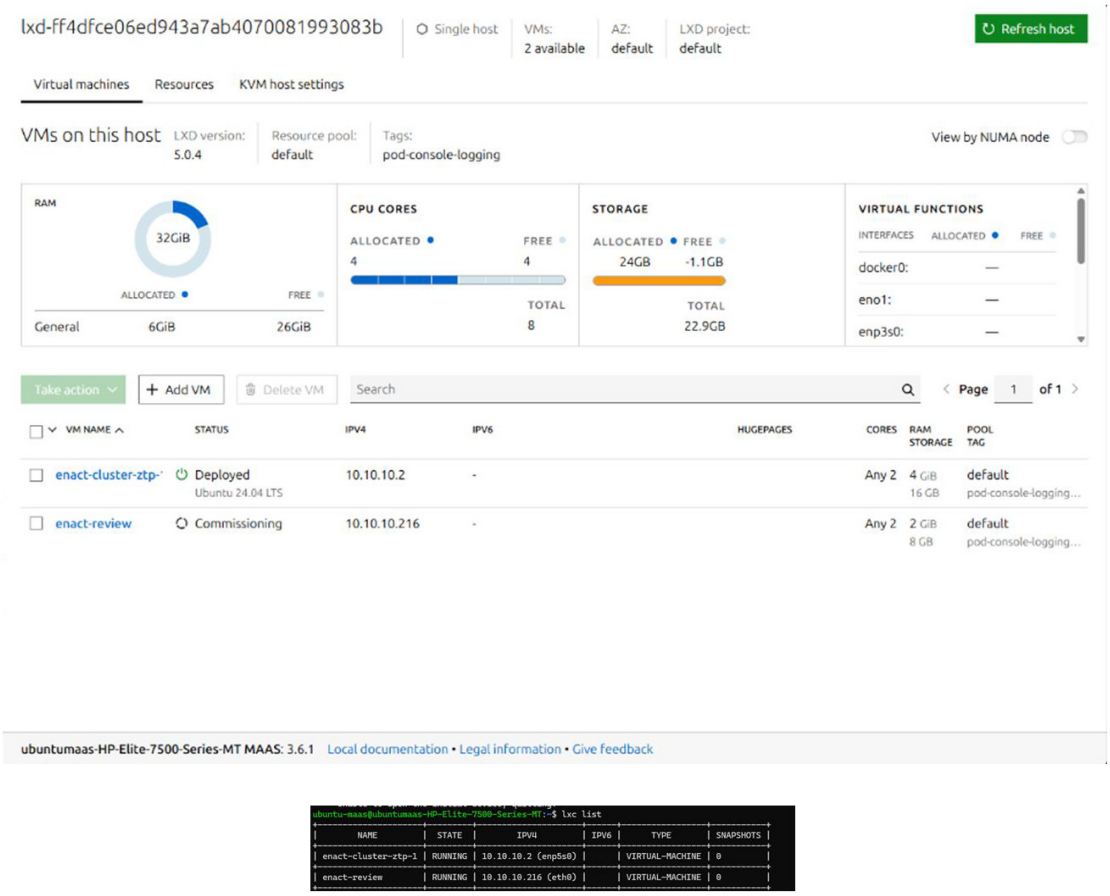

Figure 6. **Ubuntu MAAS and LXD view of deployed VMs using ZTP.**

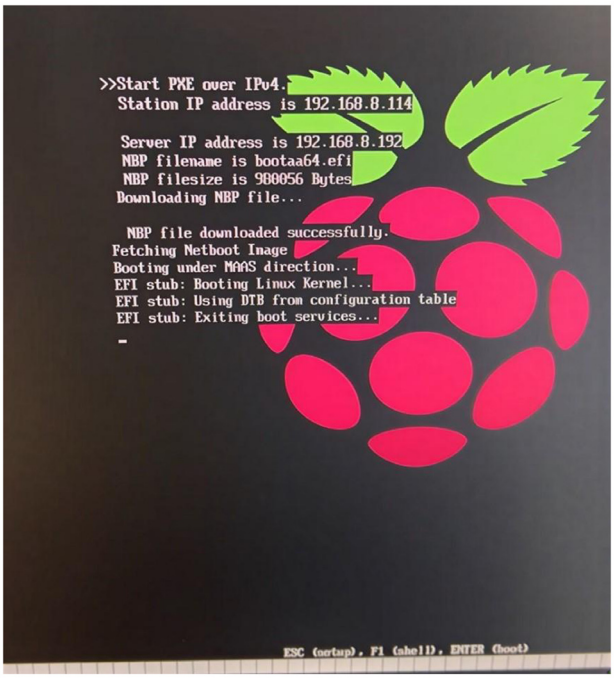

Figure 7. **Raspberry Pi PXE booting using MAAS.**

The test workflow consisted of the following stages:

- PXE boot initiated by the Raspberry Pi triggered device discovery by MAAS.
- Devices were commissioned automatically and scheduled for deployment via MAAS APIs.
- Cloud-Init manifests were applied to configure each node with a predefined system image and role-specific services (e.g., telemetry agents, Kubernetes worker node setup).
- The ZTP Agent maintained WebSocket communication with the central server throughout the process, reporting device status and provisioning outcomes.

This scenario demonstrates the ZTP framework's capability to manage physical infrastructure in remote and potentially disconnected environments. Key features observed during testing included: Full automation of OS installation and configuration without physical access; Device control via abstracted power management interfaces (e.g., IPMI or custom integrations); Reconnection and task resumption after network disruptions, enabled by the agent's persistent connection model[24]. The following code block presents the output of the ZTP Server with information about the newly deployed machine, in our case a Raspberry Pi 4 model B:


```
{
 "power_state": "off",
 "storage_size": 31457.28,
 "description": "",
 "cpu_count": 4,
 "architecture": "arm64/generic",
 "swap_size": null,
 "hostname": "coral-magpie",
 "memory": "4096 MB",
 "power_type": "webhook",
 "status": "Deployed",
 "system_id": "r3ef4n",
 "hardware_info": {
  "system_vendor": "Raspberry Pi Foundation",
  "system_product": "Raspberry Pi 4 Model B",
  "system_family": "Raspberry Pi",
  "system_version": "C03115",
  "system_sku": "0000000000C03115",
  "system_serial": "0000083ADD347F5F",
  "cpu_model": "BCM2711 (ARM Cortex-A72)",
  "mainboard_vendor": "Sony UK",
  "mainboard_product": "Raspberry Pi 4 Model B",
  "mainboard_serial": "0000083ADD347F5F",
  "mainboard_version": "C03115",
  "mainboard_firmware_vendor": "https://github.com/pftf/RPi4",
  "mainboard_firmware_date": "08/28/2024",
  "mainboard_firmware_version": "UEFI Firmware v1.38",
  "chassis_vendor": "Sony UK",
  "chassis_type": "Embedded PC",
  "chassis_serial": "0000083ADD347F5F",
  "chassis_version": "Raspberry Pi 4 Model B"
 }
}
```


The use of Cloud-Init in conjunction with MAAS provided a unified interface for provisioning and configuration, reducing operational overhead and improving deployment consistency[18]. This scenario lays the foundation for robust, scalable edge infrastructure rollouts where manual intervention is infeasible.

Performance analysis

To evaluate the performance of the proposed ZTP solution, we measured the provisioning time required for deployed devices to establish connectivity with the cloud-edge continuum infrastructure. The ZTP-based deployment time is benchmarked against the time needed to provision a machine manually. The comparison

of the two scenarios is conducted for the two different provisioning scenarios described in the Provisioning Validation section, i.e., for VM and bare metal machines deployment, and considering two user experience levels for manual provisioning, particularly beginners and experts.

For the two scenarios the provisioning process was split in individual steps to make the time measurement more precise. Below are presented the steps that are measured for each of the two scenarios.

1. **Virtual Machine Deployment:**

- **Cloud-Init Creation** -- Time required to create the Cloud-Init configuration file for deployment. This step is only applicable to manual deployment, as ZTP has the Cloud-Init configuration readily available at deployment time.
- **Install VM** -- Execution of the LXD command to deploy the virtual machine using the Cloud-Init configuration.

2. **Bare Metal Deployment:**

- **Bootable USB Creation** -- Creation of a bootable USB device for installing Ubuntu Server 24.04 on the target machine.
- **Install Ubuntu Server** -- Complete installation process of Ubuntu Server OS on the physical machine.
- **Security Configuration** -- Implementation of baseline security settings including SSH access configuration, firewall rules setup, and system package updates. While Ubuntu MAAS performs comprehensive security checks and kernel/firmware configurations, this measurement focuses on basic security configurations that manual users would typically implement proper SSH setup, firewall configuration, and updating all packages installed after OS installation.
- **Cloud-Init Setup** -- Manual installation and configuration of all components specified in the Cloud-Init configuration. This process is automated by both MAAS and ZTP through the Cloud-Init framework but must be performed manually in traditional deployment scenarios.

The following code block presents the Cloud-Init configuration that was used for both scenarios, where the values in {{ }} are replaced by actual values that connect to a deployed master node using K3s.

```
#cloud-config

package_update: true
package_upgrade: false

packages:
 - curl
 - docker.io

runcmd:
 - systemctl enable --now docker
 - curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=v{{ .Version }}+k3s1
K3S_URL=https://{{ .Host }}:{{ .Port }} K3S_TOKEN={{ .Token }} sh -
 - systemctl enable --now k3s-agent

write_files:
 - path: /etc/resolv.conf
   content: |
    nameserver 1.1.1.1
    nameserver 8.8.8.8
    nameserver 8.8.4.4
 - path: /etc/rancher/k3s/registries.yaml
   content: |
    mirrors:
     docker.io:
      endpoint:
       - "https://registry-1.docker.io"
```

Provisioning time measurement methodology

For the ZTP approach, provision time measurements commenced at the precise moment when the API call was initiated to the ZTP service, including:

- The composition and instantiation of virtual machines using LXD orchestration (Scenario 1)
- The deployment of newly connected bare metal machines through the automated provisioning pipeline (Scenario 2)

For manual deployment baselines, provision time measurements began at different initialization points:

- **VM deployment**: Time measurement commenced with cloud-init file creation for LXD deployment
- **Bare metal deployment**: Timer initiation at bootable media creation (USB stick preparation)

The experiments conducted were performed over two runs, and the results present the average of those runs for each captured step of the deployment process. Given that the accuracy of these measurements depends on multiple factors such as network connectivity, hardware specifications, and device writing speeds, the results provided represent averages and reflect our specific experimental setup where variables like bootable USB creation time can vary significantly depending on the devices used.

Provisioning time results

Table 1 and Table 2 present comprehensive time measurements for each step of both deployment scenarios. The results demonstrate significant efficiency gains when utilizing ZTP with MAAS compared to traditional manual approaches.

This comparative analysis demonstrates the practical impact of automation on provisioning speed and operational efficiency in cloud-edge environments. ZTP shows remarkable time reductions of up to 94% for bare metal deployments (from 2 hours down to 9 minutes) and 60% for VM deployments (from 10 minutes down to 4 minutes). These results enable organizations to make informed decisions regarding their infrastructure deployment strategies and validate the effectiveness of the ZTP framework in reducing operational overhead while maintaining deployment consistency and reliability. Beyond reducing deployment time, the proposed solution enhances consistency and reliability by executing identical actions for the same input deployment across all hardware and network configurations.

Discussion

Design Trade-offs and Justifications

The architecture of the ZTP framework prioritizes decentralization, local autonomy, and minimal external exposure. This choice trades some real-time orchestration capabilities (e.g., direct SSH provisioning) for

Table 1. Virtual Machine Deployment Time Comparison.

Step / Method	Manual Deployment	ZTP + MAAS
Cloud-Init Creation	~5 min	0 min (not needed)
Install VM	~5 min	~4 min (automated)
Total	~10 min	~4 min

Table 2. Bare Metal Deployment Time Comparison.

Step / Method	Manual Deployment	ZTP + MAAS
Bootable USB creation	~5 min	0 min (not needed)
Install Ubuntu	~32 min	0 min (automated)
Security configs	~45 min	0 min (automated)
Cloud-init setup	~40 min	~9 min (automatic)
Total	~2 hr	~9 min

enhanced security and reduced dependency on central infrastructure availability. By delegating provisioning responsibilities to site-local agents, the system gains resilience against network partitions and supports environments where inbound access is impractical.

The use of MAAS and Cloud-init reflects a deliberate choice to rely on open, widely supported standards. While MAAS requires a non-trivial setup, its API flexibility and PXE support justified its inclusion over lighter alternatives like Foreman or Cobbler in scenarios involving mixed hardware.

Security considerations

Security was a central concern throughout the framework's design. The use of WebSocket channels—initiated by agents—avoids inbound firewall rules, significantly reducing attack surface. The PBKDF2-based one-time authentication mechanism provides forward secrecy and eliminates persistent credentials. Additionally, role-specific Cloud-init profiles allow principle-of-least-privilege provisioning.

That said, MAAS introduces its own trust boundaries. Agents must ensure local MAAS instances are not exposed externally and are regularly patched. The Lynis hardening scan at agent startup offers a basic safeguard but should be complemented with external audits and runtime monitoring in production environments[21].

Limitations and mitigation plans

While effective in the tested scenarios, the current framework is hindered by certain limitations:

- Scalability: The in-memory registry and lack of persistent queues may limit horizontal scalability beyond 500+ agents.
- MAAS Dependence: MAAS remains a heavyweight dependency; future versions may explore container-native alternatives or micro-VM based provisioning.
- Observability: Although ELK integration provides logs, there is no real-time dashboard or policy-based alerting. Integrating Prometheus metrics with Grafana is a planned extension to overcome this issue.
- Agent Management: Agents currently lack auto-upgrade functionality or rotation of registration tokens — an area for automation in the future.

Despite these constraints, the architecture provides a strong foundation for secure, federated provisioning across heterogeneous and remote infrastructure.

Conclusions and future work

This paper presented a decentralized Zero-Touch Provisioning (ZTP) framework designed for secure and scalable deployment of edge infrastructure across federated environments. By combining a central ZTP server with lightweight, locally autonomous agents, the system eliminates the need for inbound access, reduces administrative overhead, and maintains clear network boundaries.

The proposed architecture leverages mature open-source technologies, such as Ubuntu MAAS and Cloud-init, while introducing a secure registration mechanism based on one-time credentials and WebSocket communication. Our evaluation across diverse deployment scenarios demonstrated high reliability, fast recovery from disconnections, and adaptability to both virtualized and bare-metal hardware.

Several real-world use cases, including microservice orchestration and sensor node deployment, confirmed the framework's applicability in distributed environments with constrained connectivity. Lessons learned during implementation informed design decisions such as agent-local provisioning and stateful reconnection.

Future work will focus on enhancing scalability through distributed agent state management, improving observability via integrated monitoring dashboards, and automating agent updates and policy compliance. We also plan to explore the integration of AI-driven orchestration to support fully autonomous edge infrastructure provisioning.

The results affirm that decentralized ZTP is a viable strategy for modern, heterogeneous edge deployments—and a critical enabler for dynamic, secure, and low-touch infrastructure management across cloud-to-edge continuums.

Ethics and consent

Ethical approval and consent were not required.

Data availability

Not applicable for this research.

Software availability

Source code available from: https://gitlab.eclipse.org/eclipse-research-labs/enact-project/zero-touch-provisioning-service

License: The software is licensed under Apache 2 https://gitlab.eclipse.org/eclipse-research-labs/enact-project/zero-touch-provisioning-service/-/blob/main/LICENSE.txt?ref_type=heads

Archived software available from: https://doi.org/10.5281/zenodo.17395270[28]

Data are available under the terms of the Creative Commons Attribution 4.0 International

References

1. Kapetanidou IA, Nizamis A, Karanastasis E, *et al.*: **Cognitive computing continuum: state-of-the-art review and ENACT vision & approach.** *J Grid Comput.* 2025; **23**(3): 23.
Publisher Full Text
2. Nizamis A, Neises J, Ospina D, *et al.*: **ENACT-A framework for adaptive scheduling and deployments of data intensive workloads on energy efficient edge to cloud continuum.** In: *International Conference of Engineering, Technology and Innovation (ICE/ITMC).* 2024.
Publisher Full Text
3. Liyanage M, Pham QV, Dev K, *et al.*: **A survey on Zero touch network and Service Management (ZSM) for 5G and beyond networks.** *J Netw Comput Appl*. 2022; **203**: 103362.
Publisher Full Text
4. **Cisco Network Services Orchestrator.** 2023.
Reference Source
5. **Red Hat Ansible Automation Platform.** 2023.
Reference Source
6. **AWS IoT Greengrass.** 2022.
Reference Source
7. **Azure IoT Edge.** 2023.
Reference Source
8. **The foreman: complete lifecycle management tool for physical and virtual servers.** 2023.
9. **Cobbler: linux installation server.** 2022.
Reference Source
10. **Preboot Execution Environment (PXE) specification.** 1999.
Reference Source
11. Bittencourt LF, Immich R, Sakellariou R, *et al.*: **The internet of things, fog and cloud continuum: integration and challenges.** *Internet of Things.* 2018; **3–4**: 134–155.
Publisher Full Text
12. Mahmud R, Kotagiri R, Buyya R: **Fog computing: a taxonomy, survey and future directions.** 2020; **1**: 103–130.
Publisher Full Text
13. Xiong Y, Sun Y, Xing L, *et al.*: **KubeEdge: a kubernetes native edge computing framework.** In: *Proceedings of the 2021 IEEE International Conference on Cloud Computing.* 2021.
14. **OpenYurt: extending kubernetes to edge.** 2022.
15. Chen W, Liu X, Zhang Y: **WebSocket communication patterns for IoT device management: a comprehensive analysis.** 2021; **8**(12): 9876–9889.
16. Rodriguez M, Thompson J, Kumar A: **A survey on zero-touch provisioning and management for edge computing infrastructure.** 2019; **52**(4): 1–35.
17. **Metal as a Service (MAAS).** 2023.
Reference Source
18. **Cloud-init documentation.** 2023.
Reference Source
19. Moriarty K, Kaliski B, Rusch A: **PKCS #5: password-based cryptography specification version 2.1.** 2017.
Reference Source
20. Wang X, Hassan M, Li J: **Security challenges and solutions in edge computing: a comprehensive survey.** 2022; **24**(2): 1123–1158.
21. **Lynis: Security auditing tool for Unix systems.** 2024.
22. Rescorla E: **The Transport Layer Security (TLS) protocol version 1.3.** 2018.
Publisher Full Text
23. **LXD - system container and virtual machine manager.** 2023.
24. Fette I, Melnikov A: **The websocket protocol.** 2011.
Publisher Full Text
25. Turnbull J: **The ELK Stack: Elasticsearch, Logstash, and Kibana.** 2014.
26. **K3s - Lightweight Kubernetes.** 2023.
Reference Source
27. ENACT: **Make RPI 4/3 EEPROM USB stick.** 14 10 2024.
Reference Source
28. Stefan-Cristian J: **Zero-Touch Provisioning Services.** *Zenodo.* 2025.
http://www.doi.org/10.5281/zenodo.17395270

Open Peer Review

Current Peer Review Status: ✔ ?

Version 1

Reviewer Report 27 January 2026

https://doi.org/10.21956/openreseurope.23399.r68160

© **2026 Jansen M.** This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

?

Matthijs Jansen
Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Summary: This paper describes a distributed provisioning framework for bare metal and virtual machines across edge devices, with zero-touch authentication to allow any edge site to join seamlessly and securely. The proposed system combines a central zero-touch provisioning (ZTP) server with lightweight edge agents. The ZTP server exposes an API for orchestrators to leverage the framework. The evaluation shows a significant improvement in time-to-provisioning compared to manual provisioning.

Overall assessment: I view the work as promising and practically relevant, but the paper's novelty compared to existing infrastructure provisioning and edge resource management solutions is not clearly supported at this stage. Specifically, claims around decentralization, security, and the strength of the evaluation, are currently not sufficiently supported. The "points to address" below are aimed to make the novelty claims more precise, while the "additional suggestions" could improve the paper further.

Points to address:

1. **Clarity on the decentralization claim:** The paper claims a decentralized design but has a centralized ZTP server as the core component. Please precisely define what is decentralized in your design (for example, decentralized state and centralized control), or revise the claim to hybrid or federated.
2. **Add quantification to the verification:** The verification section of the evaluation currently could be strengthened with more precise descriptions of the testbed, a quantification of key outcomes (provisioning time, success rate, resource utilization, all with distributions), and a precise qualification of security. The current coverage is high-level and qualitative, making it difficult for readers to assess the framework.
3. **Add evaluation comparisons beyond manual deployment:** The extensive related work section discusses various competing systems to this work, however, the evaluation only compares against manual deployment. This comparison gives limited real-world value in a space with many competing alternatives for infrastructure and service provisioning. While end-to-end comparison could be difficult due to a lack of direct competitors, partial

comparisons should be possible. Additionally, please report variance across repeated experiments.

4. **Expand upon the security model:** Security is a design innovation claimed together with decentralization, but sees limited coverage and validation. Please expand upon this with a threat model, and describe, for example, how preconfigured registration tokens and a one-time proof exchange mentioned in the design compare to pre-shared security keys criticized earlier in the paper.

Additional suggestions:

1. **Explain ENACT:** The evaluation currently lists "compatibility with ENACT" as a key requirement but the ENACT system is not explained in the paper. Please provide context on what this compatibility entails.
2. **Positioning of novelty:** The paper currently relies on a motivation of 'integration of standards', making the work seem like an engineering effort of combining existing standards. Please expand the motivation of conceptual contributions that make this integration challenging.

Is the rationale for developing the new software tool clearly explained?

Yes

Is the description of the software tool technically sound?

Partly

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Partly

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Distributed systems; Cloud computing; Edge computing; Deployment; Configuration

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Reviewer Report 11 December 2025

https://doi.org/10.21956/openreseurope.23399.r64445

© 2025 Gramaglia M. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Marco Gramaglia
Universidad Carlos III de Madrid, Getafe, Spain

The document introduces a decentralized Zero-Touch Provisioning (ZTP) framework developed in the context of the ENACT project to automate secure and scalable deployment of edge infrastructure.

Unlike traditional manual provisioning, which requires direct access and exposes networks to security risks, this approach uses lightweight agents and a central server to coordinate provisioning across the deployed infrastructure.

Built on open standards like PXE, Ubuntu MAAS, and Cloud-Init, it uses WebSocket-based authentication and one-time credentials for enhanced security.

The evaluation is well discussed and show significant efficiency gains when compared with manual approaches, while keeping resilience and scalability in the edge environment.

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Mobile Networking, Artificial Intelligence

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.