



Bachelor Thesis

## **Dynamic Pentest Lab Framework**

Building an Azure Pentest Deployment Framework

14. June 2024

**Authors:** Janosch Bühler, Samuel Maissen, Dante Suwanda

**Supervision:** Ivan Bütler

**Project Partner:** Hacking-Lab AG

# Abstract

---

In collaboration with Hacking-Lab AG and under the guidance of Ivan Büttler at the Department of Computer Science, Ostschweizerische Fachhochschule (OST), this thesis presents the development of the Dynamic Pentest Lab Deployment Generator, designed to facilitate the creation and deployment of cybersecurity penetration testing environments. The project addresses the need for flexible, easily deployable lab environments tailored to specific scenarios, streamlining the process for cybersecurity professionals and educators.

The framework elaborated, leverages modern infrastructure-as-code tools and cloud services to automate the deployment of virtual labs. Key components include Terraform for infrastructure management, Django for backend operations, and Angular with TailwindCSS and DaisyUI for the frontend interface. A centralized source has been established via a JavaScript Object Notation (JSON) Model, streamlining extension and enabling dynamic content hydration for the frontend. This approach guarantees a scalable and maintainable application.

The fundamental aspects of the framework involve the creation and deployment of labs to provide snapshots for quicker deployments for end-users, along with the capability to save and reload pre-designed labs. The lab creation process allows the configuration of Virtual Machines (VMs) and containers, the establishment of subnets and firewall rules to control communication, and also supports custom installations using private repositories or customized Docker images. Additionally, the framework automatically sets up an OpenVPN Server to facilitate secure access to the penetration testing lab. These features enable cybersecurity educators to construct complex, customized networked environments for educational use.

While the project effectively establishes a solid foundation, there remains room for enhancement, particularly in the realms of access technology and user experience. Additionally, the integration of the existing deployment manager directly into the generator application should be considered.

In summary, it can be concluded that a robust foundation has been established for the framework that leverages existing architecture, thereby ensuring a smooth incorporation into the Hacking-Lab platform.

**Keywords:** Pentesting, Hacking-Lab, Infrastructure as Code

# Acknowledgments

---

We would like to express our appreciation to the individuals below for their assistance with this bachelor thesis:

**Ivan Büttler** for the supervision and guidance during the course of this thesis.

**Hannes Badertscher** for providing the OSTReport Latex template.

# Contents

---

<b>1</b>	<b>Management Summary</b>	<b>1</b>
1.1	Initial Situation . . . . .	1
1.2	Procedure and Technologies . . . . .	2
1.3	Results . . . . .	3
1.4	Implications . . . . .	4
1.5	Conclusion . . . . .	4
<b>2</b>	<b>Introduction</b>	<b>5</b>
<b>3</b>	<b>Requirements</b>	<b>8</b>
3.1	Functional Requirements . . . . .	8
3.1.1	Roles . . . . .	10
3.1.2	Epics . . . . .	10
3.1.3	User Stories . . . . .	11
3.2	Non-Functional Requirements . . . . .	31
3.2.1	Functional Suitability . . . . .	31
3.2.2	Reliability . . . . .	31
3.2.3	Compatibility . . . . .	32
3.2.4	Security . . . . .	32
3.2.5	Portability . . . . .	33
3.2.6	Maintainability . . . . .	34
3.2.7	Costs . . . . .	34
3.2.8	Usability . . . . .	35
3.2.9	Tracking of the NFRs . . . . .	35
<b>4</b>	<b>Design / Architecture</b>	<b>36</b>
4.1	Domain Model . . . . .	36
4.2	System Overview . . . . .	38
4.3	Azure Lab Architecture . . . . .	44
4.3.1	Base Infrastructure . . . . .	45
4.3.2	Core Resources . . . . .	46
4.3.3	Additional Resources . . . . .	46
4.3.4	Resource Naming . . . . .	47
4.4	Backend Architecture . . . . .	49
4.4.1	Core Components . . . . .	49

4.4.2	Key Interactions and Workflows . . . . .	50
4.4.3	Scalability and Maintainability . . . . .	51
4.5	JavaScript Object Notation (JSON) Model . . . . .	51
4.6	Frontend Architecture . . . . .	53
4.6.1	Conceptual Mockup . . . . .	53
4.6.2	Final Architecture . . . . .	56
4.7	Architectural Decision Records . . . . .	59
4.7.1	ADR: 001 - Web Framework Backend . . . . .	59
4.7.2	ADR: 002 - Web Framework Frontend . . . . .	60
4.7.3	ADR: 003 - Frontend Graph Framework . . . . .	62
4.7.4	ADR: 004 - Frontend CSS Framework . . . . .	63
4.7.5	ADR: 005 - Repository Structure . . . . .	65
4.7.6	ADR: 006 - VM Customized Installation . . . . .	66
4.7.7	ADR: 007 - Public Docker Images . . . . .	67
4.7.8	ADR: 008 - Private Hacking-Lab (HL) Docker Images . . . . .	69
4.7.9	ADR: 009 - Restrict Communication between Resources . . . . .	70
4.7.10	ADR: 010 - Container Deployment . . . . .	72
4.7.11	ADR: 011 - Persistence of Labs . . . . .	76
4.7.12	ADR: 012 - Terraform Templating . . . . .	77
4.7.13	ADR: 013 - Student Access to the Lab . . . . .	78
4.7.14	ADR: 014 - Authentication Frontend . . . . .	79
4.7.15	ADR: 015 - Authentication Backend . . . . .	81
4.7.16	ADR: 016 - DNS Resolution . . . . .	82
<b>5</b>	<b>Technologies</b>	<b>84</b>
<b>6</b>	<b>Quality measures</b>	<b>86</b>
6.1	Git Workflow . . . . .	86
6.2	Code Quality . . . . .	87
6.3	Static Code Analysis . . . . .	88
6.4	CI/CD Pipeline . . . . .	88
6.4.1	Backend . . . . .	88
6.4.2	Frontend . . . . .	89
6.5	Testing Strategy . . . . .	90
6.5.1	Unit Tests . . . . .	90
6.5.2	Integration Testing . . . . .	91
6.5.3	End-to-end testing . . . . .	93
<b>7</b>	<b>Implementation</b>	<b>94</b>
7.1	Frontend . . . . .	94
7.1.1	General . . . . .	94
7.1.2	Maintability . . . . .	95
7.1.3	Development Setup . . . . .	97
7.1.4	Pages . . . . .	98

7.1.5	Components . . . . .	101
7.1.6	Services . . . . .	107
7.1.7	Styling . . . . .	110
7.2	Backend . . . . .	111
7.2.1	General . . . . .	111
7.2.2	Basic Setup and Structure . . . . .	111
7.2.3	Software . . . . .	111
7.2.4	Django Applications . . . . .	112
7.3	Deployment with Terraform . . . . .	121
7.3.1	Terraform Structure . . . . .	121
7.3.2	Templating Methodologies . . . . .	122
7.3.3	Data Management . . . . .	125
7.3.4	Resource Information . . . . .	126
7.3.5	Baseline Deployment . . . . .	127
7.3.6	Virtual Machine Deployment . . . . .	132
7.3.7	Virtual Machine Snapshot Technology . . . . .	133
7.3.8	Virtual Machine Custom Installations . . . . .	135
7.3.9	Container Deployment . . . . .	139
7.3.10	Network Restrictions . . . . .	142
7.4	Adjustments Deployment Manager . . . . .	145
<b>8</b>	<b>Results</b>	<b>148</b>
<b>9</b>	<b>Conclusion and Outlook</b>	<b>152</b>
<b>10</b>	<b>Personal Reports</b>	<b>156</b>
	<b>List of Figures</b>	<b>159</b>
	<b>List of Tables</b>	<b>160</b>
	<b>List of Listings</b>	<b>162</b>
	<b>Bibliography</b>	<b>164</b>

# Management Summary **1**

---

## **1.1 Initial Situation**

In today's digital age, the significance of cybersecurity cannot be overstated, given the escalating reliance on technology and internet by the society. Traditionally, creating environments to test the security of computer systems, known as Penetration Testing (pentest) labs, has been a labor-intensive and time-consuming process. This thesis aims to address these challenges by developing a Dynamic Pentest Lab (DPL) deployment generator, streamlining the setup and management of these labs.

The primary objective of this project was to develop a system capable of automatically creating and managing pentesting labs. This system is designed to be user-friendly and efficient, reducing the manual effort required and enabling cybersecurity professionals to focus more on the concept and potential learnings of a lab rather than the setup process. The setup process should facilitate easy assembly of multi-network environments through an interface, similar to Visio [1], involving Virtual Machines (VMs) and containers. Moreover, the system should allow customization of these resources, including the communication restrictions between them as well as applying custom installation scripts. By automating the deployment and configuration of pentesting environments, the project aims to streamline the workflow for cybersecurity educators to develop and deploy pentest labs in an automated way. It eliminates the recurring need for building static deployments and the requirement for cloud deployment knowledge in every instance. Additionally, the system is intended to support a wide range of testing scenarios, making it versatile and adaptable to various cybersecurity training and operational needs.

---

## 1.2 Procedure and Technologies

One of the first tasks of this thesis was to analyze the fundamental components and requirements using Architecture Decision Records (ADRs) [2], followed by developing relevant prototypes. Based on these findings, the following core components were established:

**Backend:** Django [3], is a versatile web framework and was used to handle server-side operations, database interactions, and implementing the core functionalities.

**Frontend:** Angular [4] was chosen for building the dynamic and interactive user interface, making the lab configuration and deployment process straightforward for the end users. In regard to the graphical component, where resources can be created and integrated in a visio-like manner, the vis-network framework [5] was utilized.

**Integration:** The product is designed to be seamlessly integrated into Hacking-Lab (HL) [6]. This integration allows lab-creators to log in with their HL credentials, configure their labs, deploy them with minimal effort, and make them readily accessible to users.

**Infrastructure Automation:** Terraform [7], a tool that automates the setup and management of infrastructure, was chosen to dynamically parse and construct all the resources configured by the pentest lab-creator and deploy them to Azure [8], which is Microsoft's cloud computing platform.

**Container Infrastructure:** To support container deployment, the comprehensive solution necessitated the employment of an Azure Kubernetes Cluster [9], which will be dynamically configured for each deployed lab.

**Custom VM Installations:** For the facilitation of custom installations, the option to use Github [10] repositories for further custom installations was established.

**Custom Docker Images:** To enable the usage of custom images and pre-existing HL Docker images from private repositories, the ability to define a custom private repository with authentication tokens was incorporated.

**VPN Access to Lab:** To provide secure access to the pentest lab via Virtual Private Network (VPN), the decision has been made to employ an OpenVPN [11] container-based solution.

**DNS Resolution:** To facilitate Domain Name System (DNS) resolution for the lab the decision was made to use Cloudflare [12] for the management of the root domain and Azure Public DNS Zone for the subdomains associated with individual labs.

**Restriction of Communication:** Benefiting from Azure's ability to define precise security rules within security groups, these capabilities have been leveraged to facilitate network segmentation and to limit communication between segments, effectively replicating real-world network conditions.



---

**Distribution of Multiflags and Credentials:** To ensure the integrity of challenge-solving and prevent the sharing of flags, a solution was established in the Terraform deployment to facilitate dynamic flag distribution and credential management. This integration aids in generating credentials and injecting the multiflags and credentials into the virtual machines and containers.

In addition to the examination of core components, a domain analysis of the application was conducted, involving the definition of a domain model and corresponding user stories. The architecture of the application was formulated using C4 architecture diagrams [13], and the Azure architecture for the lab cloud deployment was detailed, incorporating the resources and naming conventions to be utilized. Additionally, the creation of mockups for the frontend was carried out to visualize and plan the user interface effectively.

### 1.3 Results

The result is a fully functional dynamic pentest lab deployment generator that automates the setup of pentest environments. This system simplifies the process significantly, making it accessible and efficient for cybersecurity training. Lab-creators can now quickly create, customize, and deploy labs tailored to their specific needs, drastically reducing the time and effort previously required.

Key features of the system include:

- **Resource Creation:** Lab-creators can create various resources such as VMs, subnets, and containers, customizing them to fit specific testing scenarios.
- **Lab Portability:** The system allows users to save, load, and export labs, facilitating the reuse and sharing of lab configurations.
- **Snapshot Capabilities:** The state of the lab is snapshot after the initial deployment, ensuring that further deployments for the students are quick.
- **User-Friendly Interface:** The frontend provides a graphical interface that simplifies the configuration and deployment process, making it accessible even to users with limited technical expertise especially when it comes to Terraform [7].

---

## 1.4 Implications

The implementation of this generator has significant implications for creating pentest labs:

- **Efficiency:** Automation reduces the time and resources needed for lab setup, allowing cybersecurity professionals to focus more on testing and analysis.
- **Accessibility:** By simplifying the deployment process, the tool makes advanced cybersecurity testing more accessible to a broader range of users, including educators and students.
- **Integration:** The Generator is integrated into the HL environment and enriches the existing infrastructure.

## 1.5 Conclusion

In conclusion, this work has successfully developed a deployment framework that addresses the inefficiencies of traditional lab setup methods. By integrating modern technologies and adhering to best practices, the project has produced a valuable tool for creating and managing pentest labs. This project highlights the importance of innovation and automation in cybersecurity training, ensuring that experts can concentrate on developing course content without being hindered by the technical complexities of lab setup.

Reflecting on the results, the DPL deployment generator meets the initial goals set out in the project's objectives. It effectively automates the creation and management of pentest labs, significantly reducing the time and effort required for setup and maintenance. This system not only simplifies the process but also provides robust features such as resource creation, lab portability, snapshot capabilities, and a user-friendly interface which does not require knowledge about cloud deployments.

However, while this thesis has achieved significant milestones, there are areas that could benefit from further development. For instance, enhancing the backend to support more comprehensive error handling and incorporating additional user-experience features in the frontend would improve overall usability and reliability. The frontend, although functional, could be polished to offer a more intuitive and visually appealing interface.

Future developments could also explore integrating Transport Layer Security (TLS) for containers using automated certificate management solutions like Let's Encrypt [14]. Additionally, considering alternative access methods like Azure Virtual Desktop could offer more flexibility, particularly in corporate environments where VPN usage might be restricted.

# Introduction **2**

---

Hacking-Lab (HL) is a platform that provides ethical hacking for educational purposes. The overall goal is to promote awareness in the field of hacking and security. The corresponding challenges are developed in the format of a Capture The Flag scenario.

As it stands, HL leverages Terraform deployments to establish comprehensive pentest labs in Azure, enabling students to deploy labs using the HL's proprietary Deployment Manager for the corresponding challenge. However, beyond the Deployment Manager, which executes these static Terraform Deployments, there is an absence of a standardized procedure for developing these lab deployments.

Each lab, or respective its corresponding Terraform deployment, is static and manually developed to suit a specific use-case. This approach, however, involves a time-intensive process that necessitates manual interventions and several procedures, including the manual creation of snapshots for each lab. It also calls for time dedicated to debugging the Terraform deployment. Additionally, it expects the lab-creator to possess knowledge of cloud deployments, adding another layer of complexity to the process.

For understanding of today's procedure, reference will be made to the Figure 2.1. The lab designer or challenge developer manually deploys the static lab to Azure, then individually installs the necessary custom software on each Virtual Machine (VM). Once the installation is complete, the designer may incorporate static flags into the components, which remain the same for all students. Subsequently, manual snapshots of the lab are created. This is followed by the adjustment of the Terraform deployment to utilize these specific snapshots. The Terraform deployment is then incorporated into the Deployment and a `dockerfiles.tar.gz` file is generated for the purpose of adding the deployment to the challenge. Students can then initiate the deployment via the challenge and proceed to solve the lab.

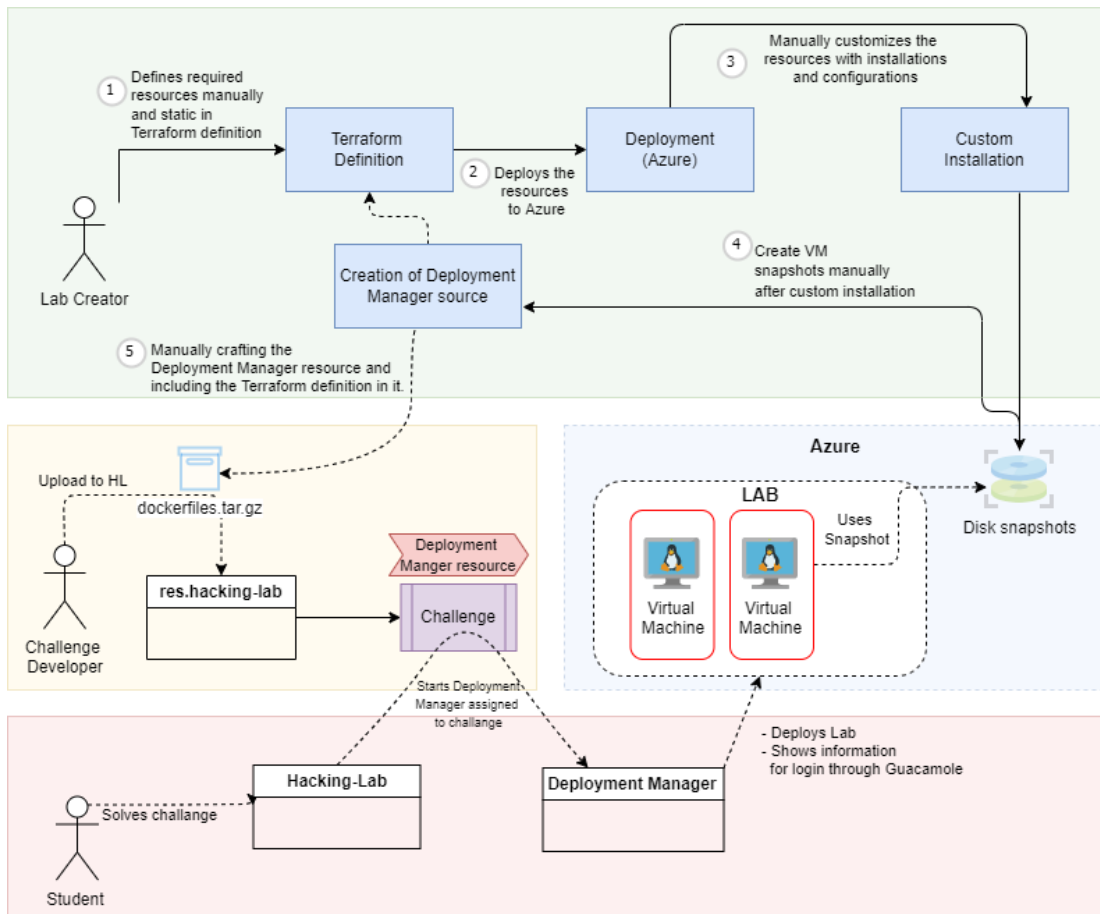


FIGURE 2.1 Manual Pentest Creation System-Context

The objective of this thesis is to develop a web application that automates the deployment and configuration of custom pentest labs, integrating them into the existing HL infrastructure. The project aims to replace the existing labor-intensive manual processes with a Visio-like tool that defines the basic infrastructure of a corporate IT network, generating configurations for deployment via Terraform [7].

The solution outlined in Figure 2.2 should simplify the creation of network setups with multiple subnets, deployment of VMs (Linux, Windows), and container services using the Visio-like tool instead of manually developing the Terraform configuration. It also needs to include Virtual Private Network (VPN) access for students, a firewall-like component to manage communication between resources, and Domain Name System (DNS) resolution for these resources. The tool should offer the ability to use existing HL Docker [15] images and allow VMs to be customized in order to create Capture The Flag (CTF) challenges. It also has to provide options for distributing dynamic flags and setting static or dynamic passwords, enhancing the educational value and uniqueness of the labs. Adjustments to the existing Deployment Manager should

facilitate the execution of the generated Terraform scripts and dynamically provide specific lab information crucial for the pentest scenarios.

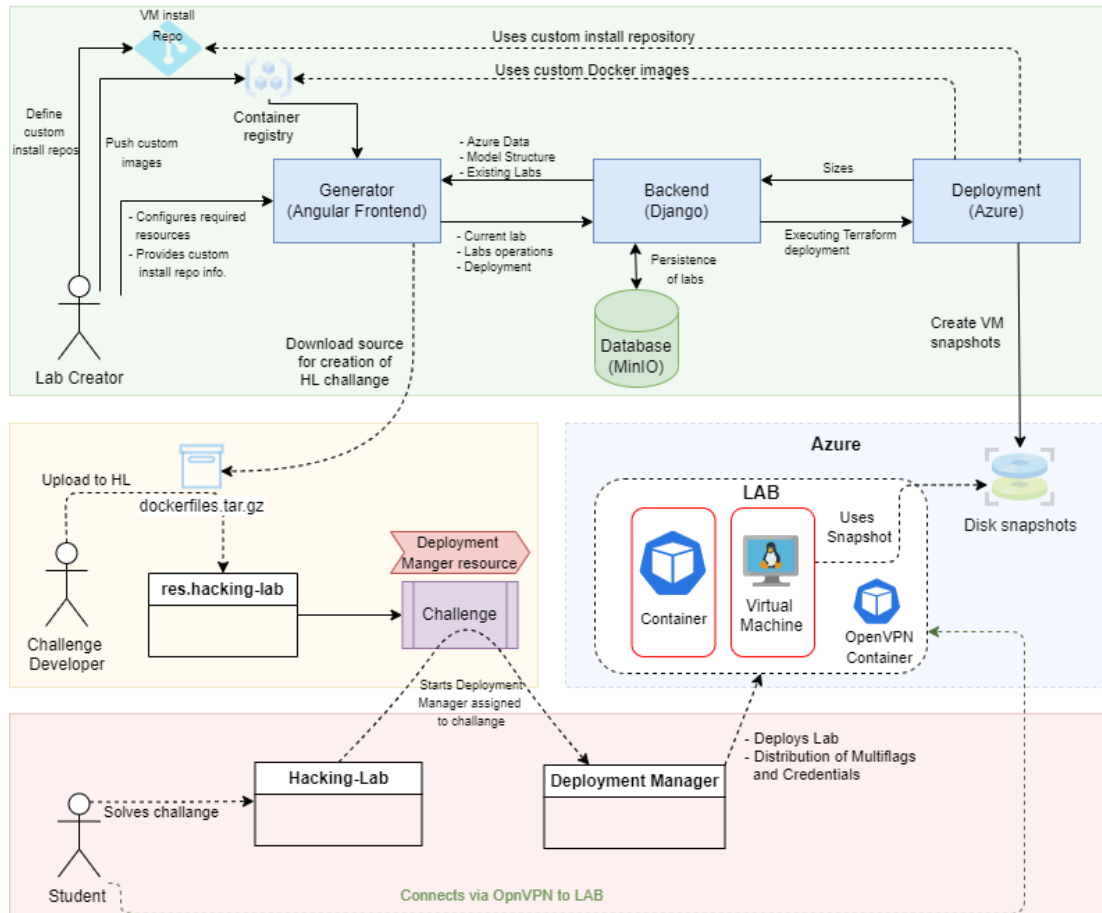


FIGURE 2.2 Dynamic Pentest Generator System-Context

The thesis has an allocation of 1080 hours in total and qualifies for twelve European Credit Transfer System (ECTS) credits for each participant.

# Requirements **3**

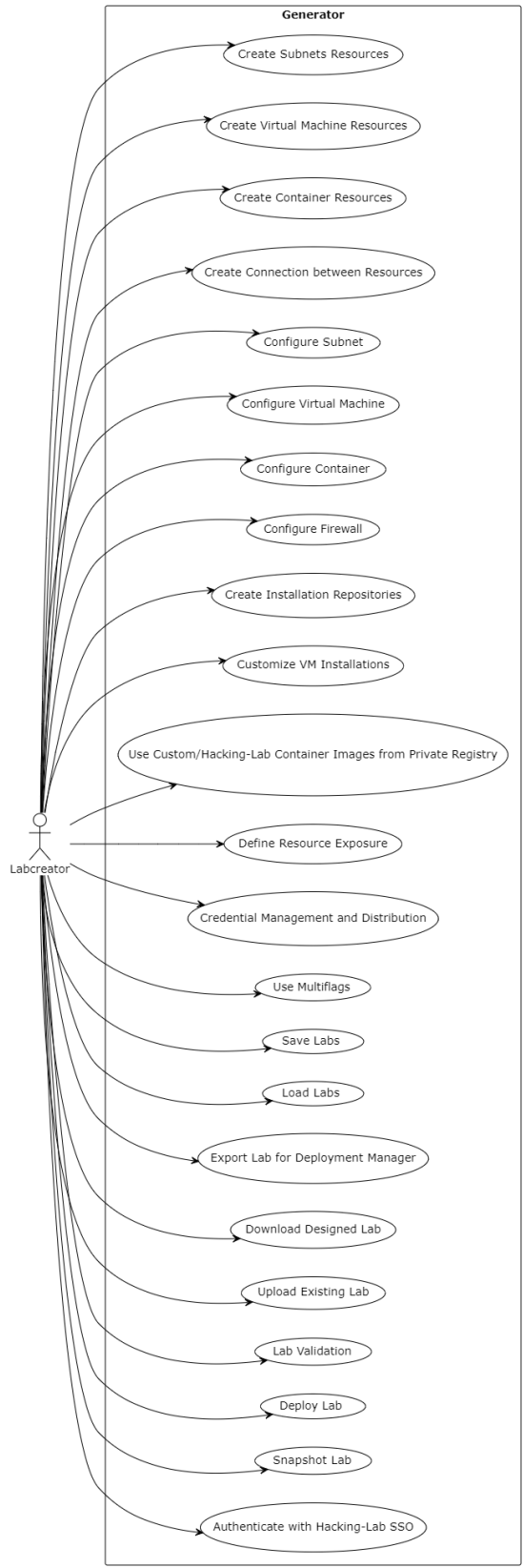
---

In this chapter, in-depth requirements are defined that provide the foundation for the development of the application. User stories, epics, and Non-Functional Requirements (NFRs) are employed to effectively define these requirements.

## **3.1 Functional Requirements**

User stories and epics are utilized to define functional requirements. Epics serve to group user stories into broader, more comprehensive requirements. Additionally, developer stories are defined to cater to non-business-related requirements.

The user stories were derived from the initial task description, which can be found in the appendix. In [Figure 3.1](#) a high-level overview of the stories is provided in the form of a use-case diagram.



**FIGURE 3.1** Use-Case Diagram

---

### 3.1.1 Roles

In the Generator domain, the role **lab-creator** refers to the central role within the application. The Generator application is used by the **lab-creator** to generate and configure custom Penetration Testing (pentest) labs.

### 3.1.2 Epics

The business requirements are defined by the following epics. Additionally, there are administrative/dev epics for general administration, documentation, deployment, and infrastructure tasks. These epics are not further described here.

#### **Core**

This epic focuses on the fundamental functionalities of the system. It includes the basic creation of defined resources without customization, forming the core foundation of the application.

#### **Lab Portability**

Lab portability is the primary focus of this epic, allowing users to save, load, and export labs for future use or sharing. It aims to enhance the flexibility and mobility of created labs.

#### **Lab Deployment**

The objective of lab deployment is to streamline the process of deploying the designed labs to Azure to test their functionality. It also focuses on the creation of lab snapshots, which capture the state of a lab after custom installation. It enables students to deploy the generated lab more quickly in order to reduce deployment time.

#### **Resource Configuration**

Resource configuration aims to provide lab-creators with the ability to customize specific attributes of lab resources. This includes configuring properties such as custom sizes, credentials, and other parameters. Additionally, it allows for communication restrictions via the implementation of firewall rules.

#### **Resource customization**

Resource customization extends beyond basic configuration by allowing users to utilize their own installation scripts stored in a own Git [16] repository. This enables the installation of specific software or the customization of resources through tailored scripts.

#### **Authentication**

This epic focuses on integrating HL Single Sign-On (SSO) to enhance security and streamline user authentication.

#### **Lab validation**

Lab validation is essential for ensuring the correctness and compatibility of created labs.



---

This epic focuses on providing checks to validate the integrity of the user-designed labs.

### **3.1.3 User Stories**

User Stories are initially drafted in a basic format with minimal details. However, during sprint planning sessions, each user story is thoroughly elaborated, adding comprehensive details and specifications for the sprint.

#### **User Story Structure**

A user story is defined with following properties:

- Name
- User Story: *As a role I want to goal*
- Epic
- Acceptance criteria
- Technical acceptance criteria

## Create Subnets Resources

<b>Name</b>	Create Subnets Resources
<b>User Story</b>	As a lab-creator I want to create network subnets in my lab in order to model a multi-subnet environment.
<b>Epic</b>	Core
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ The network resource subnet can be selected.</li> <li>▪ Subnets can be selected and added to the canvas.</li> <li>▪ A custom label can be added to the subnet.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ Subnet range is dynamically calculated in the backend.</li> <li>▪ A subnet resource is identified by a unique name.</li> <li>▪ The backend provides a prefix for the unique name.</li> <li>▪ The frontend extends the provided name prefix with an unique number.</li> <li>▪ The subnet class is implemented in the backend.</li> <li>▪ The class can dynamically be mapped to a Subnet resource in the frontend.</li> <li>▪ The subnet data structure/class is built in a way that allows for easy extension with additional attributes.</li> <li>▪ The subnet objects can be parsed by the parser service into valid tfvars variables.</li> <li>▪ The Terraform [7] definition facilitates the dynamic creation of subnets, based on the provided configuration.</li> </ul>

TABLE 3.1 User Story: Create Subnets Resources

## Create Virtual Machine Resources

<b>Name</b>	Create Virtual Machine Resources
<b>User Story</b>	As a lab-creator I want to create VMs in my lab.
<b>Epic</b>	Core
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ The resource VM can be selected.</li> <li>▪ VMs can be selected and added to the canvas.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ The VM class is implemented in the backend.</li> <li>▪ The class can dynamically be mapped to a VM resource in the frontend.</li> <li>▪ A VM resource is identified by an unique name.</li> <li>▪ The backend provides a prefix for the unique name to the frontend.</li> <li>▪ The frontend extends the provided name prefix with an unique number.</li> <li>▪ The VM data structure/class is built in a way that allows for easy extension with additional attributes.</li> <li>▪ The VM objects can be parsed by the parser service into valid tfvars variables.</li> <li>▪ The Terraform definition facilitates the dynamic creation of VMs, based on the provided configuration.</li> </ul>

TABLE 3.2 User Story: Create Virtual Machine Resources

---

## Create Container Resources

<b>Name</b>	Create Container Resources
<b>User Story</b>	As a lab-creator I want to create VMs in my lab.
<b>Epic</b>	Core
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ The resource container can be selected.</li><li>▪ The container resource can be selected and added to the canvas.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ A container resource is identified by a unique name.</li><li>▪ The backend provides a prefix for the unique name.</li><li>▪ The frontend extends the provided name prefix with a unique number.</li><li>▪ The container class is implemented in the backend.</li><li>▪ The class can be mapped to a container resource in the frontend.</li><li>▪ The container data structure/class is built in a way that allows for easy extension with additional attributes.</li><li>▪ The container objects can be parsed by the parser service into valid tfvars variables.</li><li>▪ The Terraform definition facilitates the dynamic creation of containers, based on the provided configuration.</li></ul>

TABLE 3.3 User Story: Create Container Resources

---

### Create Connection between Resources

<b>Name</b>	Create Connection between Resources
<b>User Story</b>	As a lab-creator I want to create connections between subnets and virtual machines or containers.
<b>Epic</b>	Core
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ VMs/containers can be assigned to subnets</li><li>▪ Connection between subnets and virtual machines/containers can be drawn.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ VMs/containers can only be assigned to one a single subnet</li><li>▪ The logic implemented in the frontend, configures the corresponding subnet to the virtual machine/ container when a connection is drawn</li><li>▪ Containers can only be connected to Kubernetes subnets, and virtual machines can only be connected to VM subnets.</li></ul>

TABLE 3.4 User Story: Create Connection between Resources

## Access to the Lab via VPN

<b>Name</b>	Access to the Lab via VPN
<b>User Story</b>	As a lab-creator, I want to automatically provide access to the lab through a VPN.
<b>Epic</b>	Core
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ Deployment of VPN access is automatically set up per lab.</li> <li>▪ Ability to connect to the lab using a VPN.</li> <li>▪ Each lab has a unique VPN profile with unique access keys.</li> <li>▪ The VPN connection provides compatibility with the Kookarai environment.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ Core Terraform deployment to dynamically setup VPN.</li> <li>▪ VPN profiles and access keys are generated automatically and uniquely</li> <li>▪ VPN profile is written to Terraform output</li> <li>▪ VPN profile is downloadable from Deployment Manager</li> </ul>

TABLE 3.5 User Story: Access to the Lab via VPN

## Subnet Configuration

<b>Name</b>	Subnet Configuration
<b>User Story</b>	As a lab-creator I want to be able to configure the subnet with specific properties.
<b>Epic</b>	Resource Configuration
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ A custom label can be defined for the subnet</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ A custom label attribute for the subnet class is implemented</li> </ul>

TABLE 3.6 User Story: Subnet Configuration

---

## Container Configuration

<b>Name</b>	Container Configuration
<b>User Story</b>	As a lab-creator I want to be able to configure the container deployment with specific properties.
<b>Epic</b>	Resource Configuration
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ Image can be defined</li><li>■ Ports to expose can be defined.</li><li>■ Environment variables can be defined.</li><li>■ CPU and memory requests can be defined.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ Configuration mask in fronted for image, port, CPU/memory requests and enviromental variables.</li><li>■ The container class is expanded by this properties.</li></ul>

TABLE 3.7 User Story: Container Configuration

---

## Virtual Machine Configuration

<b>Name</b>	Virtual Machine Configuration
<b>User Story</b>	As a lab-creator I want to be able to configure the VM with specific properties.
<b>Epic</b>	Resource Configuration
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Publisher, offer and SKU can be defined.</li><li>▪ Disk size can be defined.</li><li>▪ Admin user and password can be defined.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Configuration mask in frontend for publisher, offer and SKU and disksize.</li><li>▪ The available publisher, offer and SKU are directly pulled from Azure API and are displayed as dropdown.</li><li>▪ The Terraform definition facilitates the dynamic creation of VMs, based on the provided configuration.</li></ul>

**TABLE 3.8** User Story: Virtual Machine Configuration



## Firewall Configuration

<b>Name</b>	Firewall Configuration
<b>User Story</b>	As a lab-creator I want to be able to restrict or allow traffic between resources.
<b>Epic</b>	Resource Configuration
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ Central firewall in frontend with table like view to define rule-set.</li> <li>■ Ability to restrict/allow traffic from VPN to subnets, VMs and containers.</li> <li>■ Ability to restrict/allow traffic between subnets and subnets.</li> <li>■ Ability to restrict/allow traffic from VMs and containers to subnets.</li> <li>■ Ability to restrict/allow traffic from subnets to VMs and containers.</li> <li>■ Ability to restrict on protocol basis UDP/TCP</li> <li>■ Ability to restrict on port basis.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ A rule is identified by an unique name.</li> <li>■ The backend provides a prefix for the unique name.</li> <li>■ The frontend extends the provided name prefix with an unique number.</li> <li>■ The rule class is implemented in the backend.</li> <li>■ The class can be mapped to a rule resource in the frontend.</li> <li>■ The rule data structure/class is built in a way that allows for easy extension with additional attributes.</li> <li>■ The rule objects can be parsed by the parser service into valid tfvars variables.</li> <li>■ The Terraform definition facilitates the dynamic creation of security groups and security rules, based on the provided configuration to grant or restrict access.</li> </ul>

TABLE 3.9 User Story: Firewall Configuration

---

## Custom Installation Repositories

<b>Name</b>	Custom Installation Repositories
<b>User Story</b>	As a lab-creator I want to be able to have a template custom configuration repository.
<b>Epic</b>	Resource Customization
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ A template repository is available which can be cloned.</li><li>■ It should include two example scripts for the custom installation and post installation.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ A template repository for Windows and Linux is available.</li><li>■ It concludes an example install script which has error handling in place and samples on how to retrieve custom credentials applied to the virtual machine.</li><li>■ It concludes an example of post-install script with samples about multiflag retrieval.</li></ul>

TABLE 3.10 User Story: Custom Installation Repositories

---

## Customized Installations VM

<b>Name</b>	Customized Installations VM
<b>User Story</b>	As a lab-creator I want to be able to specify a Github repository for each VM which contains custom installation scripts.
<b>Epic</b>	Resource Customization
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Ability to specify Github repository and credentials to it.</li><li>▪ Ability to include install scripts which are executed before the lab is snapshotted as well as post install scripts which are executed after a student deploys his own instance of the lab.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ When running Terraform it can access and download the Github repositories by using a specified access token.</li><li>▪ The custom configuration needs to run for either Linux or Windows.</li><li>▪ Terraform executed the install script depending on the OS.</li><li>▪ The install script is executed before the creation of snapshots</li><li>▪ After snapshots are performed or a student deploys the lab, the post install script gets executed by Terraform.</li></ul>

TABLE 3.11 User Story: Customized Installations VM

### Custom/HL Container Images from Private Registry

<b>Name</b>	Custom/HL Container Images from Private Registry
<b>User Story</b>	As a lab-creator I want to be able to use custom images coming from a private registry.
<b>Epic</b>	Resource Customization
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ Custom private container registry with credentials can be defined.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ The configuration mask allows to define custom repositories with credentials.</li> <li>▪ Kubernetes secrets are automatically generated through Terraform to allow pull requests on private repositories using the supplied credentials.</li> </ul>

TABLE 3.12 User Story: Custom/HL Container Images from Private Registry

### Selective Resource Exposure

<b>Name</b>	Selective Resource Exposure
<b>User Story</b>	As a lab-creator, I want to selectively expose certain resources to students ensuring that only required information is accessible to enhance learning.
<b>Epic</b>	Resource Customization
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ Only the defined resources should be exposed to students.</li> <li>▪ An attribute to control exposure for both VMs and containers.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>▪ Configuration mask in frontend implements a checkbox on VMs and containers with an expose_to_output flag.</li> <li>▪ A filter should be implemented on the dataset exposed to the Deployment Manager in Terraform, filtering resources based on the expose_to_output flag.</li> </ul>

TABLE 3.13 User Story: Selective Resource Exposure

## Credential Management and Distribution

<b>Name</b>	Credential Management and Distribution
<b>User Story</b>	As a lab-creator, I want to define custom credentials, which are then distributed by Terraform to the corresponding virtual machine or container.
<b>Epic</b>	Resource Customization
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ Credentials are distributed to the appropriate VMs or containers.</li> <li>■ Credentials can be accessed easily when using them in custom installations.</li> <li>■ Dynamic credentials can be defined, which are dynamic and unique for each lab.</li> <li>■ It can be decided whether credentials should be exposed to the student in the Deployment Manager.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ The configuration mask allows to define custom credentials for VMs and containers</li> <li>■ The configuration masks allow to define if credential should be created dynamically.</li> <li>■ Distribution of credential in VMs by writing them to a file.</li> <li>■ Credentials are passed as environment variables to containers.</li> <li>■ Dynamic credentials are created by the Terraform deployment.</li> <li>■ The dataset is constructed in Terraform deployment with credentials to expose to student.</li> </ul>

TABLE 3.14 User Story: Credential Management and Distribution

## Multiflag Integration and Management

<b>Name</b>	Multiflag Integration and Management
<b>User Story</b>	As a lab-creator, I want to define and inject multiflags into VMs and containers via Terraform to enhance the uniqueness of the labs.
<b>Epic</b>	Resource Customization
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ Multiflags are seamlessly integrated and accessible within both VMs and container environments.</li> <li>■ The multiflag integration process is automated and aligned with the HL Framework's deployment manager.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ Multiflags are correctly read from the deployment manager's environment variables and passed to Terraform.</li> <li>■ Multiflags are distributed within Terraform to the VMs and containers.</li> </ul>

TABLE 3.15 User Story: Multiflag Integration and Management

## Save labs

<b>Name</b>	Save Labs
<b>User Story</b>	As a lab-creator I want be able to save my labs.
<b>Epic</b>	Lab Portability
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ Labs can be stored in the application for future editing capabilities.</li> </ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"> <li>■ Connection to MinIO is implemented in Backend.</li> <li>■ Each lab has a unique ID.</li> <li>■ Labs created and stored in MinIO.</li> <li>■ Labs stored in MinIO can be deleted if wanted.</li> </ul>

TABLE 3.16 User Story: Save Labs

---

## Load Labs

<b>Name</b>	Load Labs
<b>User Story</b>	As a lab-creator I want to be able to load existing labs.
<b>Epic</b>	Lab Portability
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Labs existing in the generator can be loaded and edited again.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Labs can be loaded from MinIO to edit them.</li><li>▪ Loading a lab and saving it again reuses the unique ID.</li></ul>

TABLE 3.17 User Story: Load Labs

## Export Lab for Deployment Manager

<b>Name</b>	Export Lab for Deployment Manager
<b>User Story</b>	As a lab-creator I want to export my designed lab, so I can import it into HL as resource for a challenge.
<b>Epic</b>	Lab Portability
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Option, in form of a download button, for the user to download current state of the created lab.</li><li>▪ The download automatically includes the Deployment Manager required for deploying the lab.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ After clicking on the download button, the graphical input is passed to the Generator backend where it gets parsed into Terraform config format.</li><li>▪ The downloadable file is a .tar.gz format, in which the deployment is automatically included into the Deployment Manager.</li></ul>

TABLE 3.18 Export Lab for Deployment Manager

---

**Download Designed Lab**

<b>Name</b>	Download Designed Lab
<b>User Story</b>	As a lab-creator I want to download a designed lab.
<b>Epic</b>	Lab Portability
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ Add an option, in form of a save button, to download the current state of the graphically created lab.</li><li>■ This download only includes the graphical configuration so that it can be uploaded to a generator running in a different tenant.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ The Generator provides the graphically designed lab as a file download.</li><li>■ This file can be interpreted by the Generator and can be re-opened for editing.</li><li>■ Generated file can be imported by generators running in different tenants.</li></ul>

**TABLE 3.19** User Story: Download Designed lab



---

## Upload Existing Lab

<b>Name</b>	Upload Existing Lab
<b>User Story</b>	As a lab-creator I want to upload my existing lab and edit it.
<b>Epic</b>	Lab Portability
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ Allow uploading a lab file that has been previously downloaded.</li><li>■ If not manipulated, the file opens the designed lab in the same state as it was downloaded beforehand.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ On a successful import the earlier the downloaded lab will be loaded in the same state as it was saved.</li><li>■ The data-structure that describes all the nodes, edges, configurations and connections in between, is interpreted by the Generator and displayed within the editing canvas.</li><li>■ The imported lab can then be further edited as if it was created from scratch by the user.</li><li>■ The lab can be imported to a different generator running in other tenants.</li></ul>

TABLE 3.20 User Story: Upload existing Lab

## Lab Validation

<b>Name</b>	Lab Validation
<b>User Story</b>	As a lab-creator I want to check my current lab for its validity
<b>Epic</b>	Lab validation
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ The currently opened lab can be checked for its validity.</li><li>▪ If the current lab is invalid, a feedback dialog will appear indicating the issues of the configured lab.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ The lab-creators's current lab layout is checked for Terraform compatibility.</li><li>▪ The input provided by the lab-creator is checked.</li><li>▪ If the lab was configured with resources that are not available anymore (i.E. Azure SKUs), the configuration field is marked red as the lab cannot be deployed anymore.</li><li>▪ The backend returns the results of the validity check to the frontend, which can be interpreted by the user.</li></ul>

TABLE 3.21 User Story: Lab Validation

## Deploy Lab

<b>Name</b>	Deploy Lab
<b>User Story</b>	As a lab-creator I want to be able to deploy the lab.
<b>Epic</b>	Lab Deployment
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Function to deploy current designed lab is available</li><li>▪ Deployment status is displayed and potential deployment errors can be retrieved.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Deployment Service is implemented to deploy Terraform</li><li>▪ Mechanism to destroy failed or orphaned deployments</li></ul>

TABLE 3.22 User Story: Deploy Lab

---

**Snapshot lab**

<b>Name</b>	Snapshot Lab
<b>User Story</b>	As a lab-creator I want to be able to use snapshot technology to speed up deployments.
<b>Epic</b>	Lab Deployment
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Current deployment is automatically snapshotted and snapshot version is added to current lab configuration.</li><li>▪ Snapshots are automatically used if deployed from the students.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>▪ Lab lab file is enhanced with the snapshot version</li><li>▪ Snapshot logic is implemented in Terraform</li><li>▪ Dev mode is implemented in Terraform to define what should perform the snapshot.</li><li>▪ Lab mode is implemented in Terraform to define what should use the snapshots for the creation of VMs.</li><li>▪ When redeploying the same lab from the generator, old snapshots are deleted.</li></ul>

**TABLE 3.23** User Story: Snapshot Lab

---

## Authenticate with HL SSO

<b>Name</b>	Authenticate with HL SSO
<b>User Story</b>	As a user I want to authenticate with my Hacking-Lab account.
<b>Epic</b>	Authentication
<b>Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ The lab-creator is automatically redirected to the HL authentication service if not logged in.</li><li>■ There is a possibility to log off from the application.</li><li>■ Only lab-creator with a specific role are allowed to access the application.</li></ul>
<b>Technical Acceptance Criteria</b>	<ul style="list-style-type: none"><li>■ When the frontend is accessed, the lab-creator is automatically redirected to the HL authentication service, where the lab-creator can log in with their HL credentials.</li><li>■ After successful authentication, the lab-creator is redirected back to the application.</li><li>■ Authentication validity is checked for every call to the backend.</li><li>■ The authentication details and token are stored in the session cookie in the frontend.</li><li>■ The session cookie is removed when the lab-creator presses the log-off button.</li><li>■ The authentication token is then passed to the backend with every API call.</li><li>■ Access is granted based on whether the lab-creator has a specific role assigned.</li></ul>

TABLE 3.24 User Story: Authenticate with HL SSO

---

## 3.2 Non-Functional Requirements

The International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 25010 standard is used to define NFRs for this project, with only the relevant categories being selected. The goal is to implement all NFRs classified as high priority and the majority of those rated as medium. Additionally, if resources permit, the aim is to implement NFRs categorized as low priority as well.

### 3.2.1 Functional Suitability

ID	NFR01
Description	Application should be able to perform lab deployments on Azure.
Requirement	Functional Suitability - Functional Completeness
Priority	High
Verification process	Test if lab structure was successfully deployed and is accessible.
Measures	Deployments can be accessed via the provided VPN and are complete

TABLE 3.25 Non-Functional Requirements: NFR01

### 3.2.2 Reliability

ID	NFR02
Description	Multiple users can deploy and use their labs simultaneously.
Requirement	Concurrency - Multi user capability
Priority	High
Verification process	Stress test the system with multiple lab-creators designing and deploying labs at the same time.
Measures	The app is usable while multiple lab-creators are creating and deploying labs.

TABLE 3.26 Non-Functional Requirements: NFR02

### 3.2.3 Compatibility

ID	NFR03
Description	The application should be able to work with private image repositories
Requirement	Compatibility - Interoperability
Priority	High
Verification process	Test and verify if private repositories can be used in deployments
Measures	The application implements authentication to private repositories

TABLE 3.27 Non-Functional Requirements: NFR03

ID	NFR04
Description	The generated lab has to be deployable from the adjusted HL Deployment Manager.
Requirement	Compatibility - Interoperability
Priority	High
Verification process	Test and verify if the generated lab is deployable and destroyable through the HL deployment manager.
Measures	The deployment is compatible and implements the required variables which are used in HL Deployment Manager. Specific lab output is displayed.

TABLE 3.28 Non-Functional Requirements: NFR04

### 3.2.4 Security

ID	NFR05
Description	HL's intellectual property (created labs) is protected and not publicly available.
Requirement	Security - Confidentiality
Priority	High
Verification process	Test if the Generator is only usable when authenticated with HL and has been assigned a specific role. Test if the HL Docker images used are stored on a private DockerHub, accessible only to authorized users.
Measures	Use HL SSO authentication for the Generator application. Use only private repositories for storing of HL images.

TABLE 3.29 Non-Functional Requirements: NFR05

---

### 3.2.5 Portability

ID	NFR06
Description	The system must provide a RESTful API which concludes all actions possible defined by the user stories.
Requirement	Portability - Adaptability
Priority	High
Verification process	Developer checks if all actions are provided over REST. This is best effort.
Measures	All actions defined can be triggered over REST.

TABLE 3.30 Non-Functional Requirements: NFR06

ID	NFR07
Description	Generated data is easy and intuitive to export, import safe and load.
Requirement	Portability - Data handling
Priority	Medium
Verification process	Verified with user tests focusing on usability.
Measures	Functionalities are provided that allow user to export/import and save/load labs.

TABLE 3.31 Non-Functional Requirements: NFR07

### 3.2.6 Maintainability

ID	NFR08
Description	The system has to be maintainable for the HL developers after this thesis is completed.
Requirement	Maintainability
Priority	Medium
Verification process	Regular communication with HL developers to discuss technology choices and architectural decisions. Additionally, conduct code reviews, adherence checks to coding standards, and assess documentation completeness.
Measures	Follow the general clean code principles, ensuring readability, modularity, and consistency throughout the software. Document the development process comprehensively, including design choices, architectural decisions, and codebase organization. Technologies employed should align with HL's preferences and standards according to their developers and Ivan Bütler.

TABLE 3.32 Non-Functional Requirements: NFR08

### 3.2.7 Costs

ID	NFR09
Description	The system should optimize resource usage to minimize Azure costs for the labs.
Requirement	Cost Optimization
Priority	Medium
Verification process	Check that chosen solutions were chosen in regard to the costs. Check that deployments are properly cleaned up.
Measures	Utilize Azure cost management tools to identify costs of the resource configurations. Implement scaling mechanisms to provision and de-provision Azure instances based on demand, optimizing resource utilization and reducing costs. Implement a process that guarantees the deletion of unsuccessful deployments.

TABLE 3.33 Non-Functional Requirements: NFR09



### 3.2.8 Usability

ID	NFR10
Description	The user interface should be intuitive and user-friendly, with clear instructions and error messages.
Requirement	Usability
Priority	Medium
Verification process	Hand a demo version of the application to potential users that correspond to possible end-users of the product.
Measures	Follow usability and accessibility standards in the code and design of our application. Especially enabling strict rules of ESLint enforce the accessibility standard that is thesis strives for.

TABLE 3.34 Non-Functional Requirements: NFR10

### 3.2.9 Tracking of the NFRs

NFR Nr.	Priority	Status
NFR1	High	OK
NFR2	High	OK
NFR3	Medium	OK
NFR4	High	OK
NFR5	High	OK
NFR6	High	OK
NFR7	High	OK
NFR8	Medium	OK
NFR9	Medium	OK
NFR10	Medium	Partially

TABLE 3.35 Tracking of the NFRs

# Design / Architecture 4

---

This chapter delves into the design aspects of the Dynamic Pentest Lab (DPL) system, providing a detailed overview of its design and components. The structure of the system is visualized using C4 architecture diagrams [13]. Additionally, Architectural Decision Record (ADR) [2] are used to document key architectural decisions regarding the application.

## 4.1 Domain Model

The domain model, designed in Figure 4.1, provides a rough visual representation of the core components of the DPL domain and how they incorporate with each other. This model serves as a foundational blueprint for understanding the architecture and functionality of the application.

The DPL system is designed to facilitate the creation and deployment of customized pentest labs in a cloud environment. The primary components of the domain model include several key elements:

The **lab-creator** is responsible for designing and creating penetration testing labs. This user interacts with the Generator (Frontend), an intuitive interface for configuring these labs.

The **Lab Generator** generates a Lab entity. These labs consist of various **resources**, which define network resources such as, subnets, security groups, VPN as well as VM and Containers. The **connections** define the actual relationship between computing resources and subnets belonging to a particular lab.

The lab configuration gets translated by the **parser** into Terraform deployment variables, forming the **lab deployment** together with the Terraform template.

The **deployer** component uses this **lab deployment** to create the actual lab environment in **Azure** [8]. This includes the creation of snapshots for future use.



FIGURE 4.1 DPL Domain Model

## 4.2 System Overview

The system overview in Figure 4.2 delineates the organisation and core functionalities of the system, providing insight into both its internal and external components.

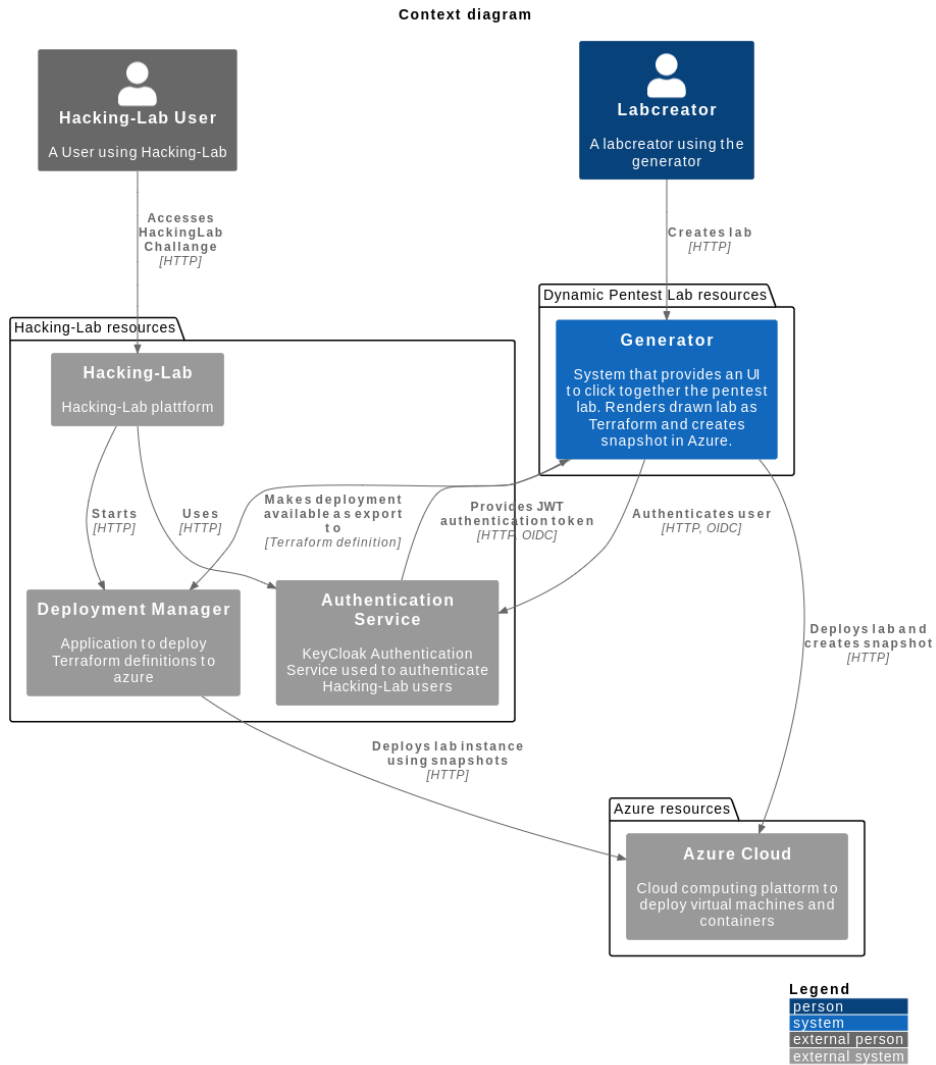


FIGURE 4.2 C4 Context Diagram

---

The context diagram categorizes components into four main resource groups: DPL resources, HL [6] resources and Azure resources:

### **DPL resources**

- **Generator Application:** This is the core system that provides the interface for creating labs. Lab-creators can configure various lab resources and settings through this User Interface (UI). Once configured, the Generator application converts the setup into Terraform [7] configurations and deploys them to Azure. This involves creating the necessary infrastructure (VMs, containers, networks) and taking snapshots for future use by the students.

### **HL resources**

- **Hacking-Lab:** The main platform where HL Users access and engage with pentest challenges. The users interact with the labs, which are deployed through the deployment manager assigned to the particular challenge.
- **Deployment Manager:** An application within HL that handles the deployment of Terraform definitions to Azure. Furthermore it provides output to the students about the resources deployed, such as DNS, credentials and IP addresses.
- **Authentication Service:** This service verifies the identities of users accessing either the Hacking-Lab platform or the generator application. It issues JWT tokens to maintain secure communication and control access effectively.

### **Azure resources**

- **Azure Cloud:** The primary cloud infrastructure used for deploying the VMs and containers that constitute the pentest labs.

## Container diagram

The container diagram, outlined in Figure 4.3, provides a more detailed view of the system's components and their interactions. This diagram extends the context diagram by breaking down the generator application into containers, showing how they interact with each other and with external systems.

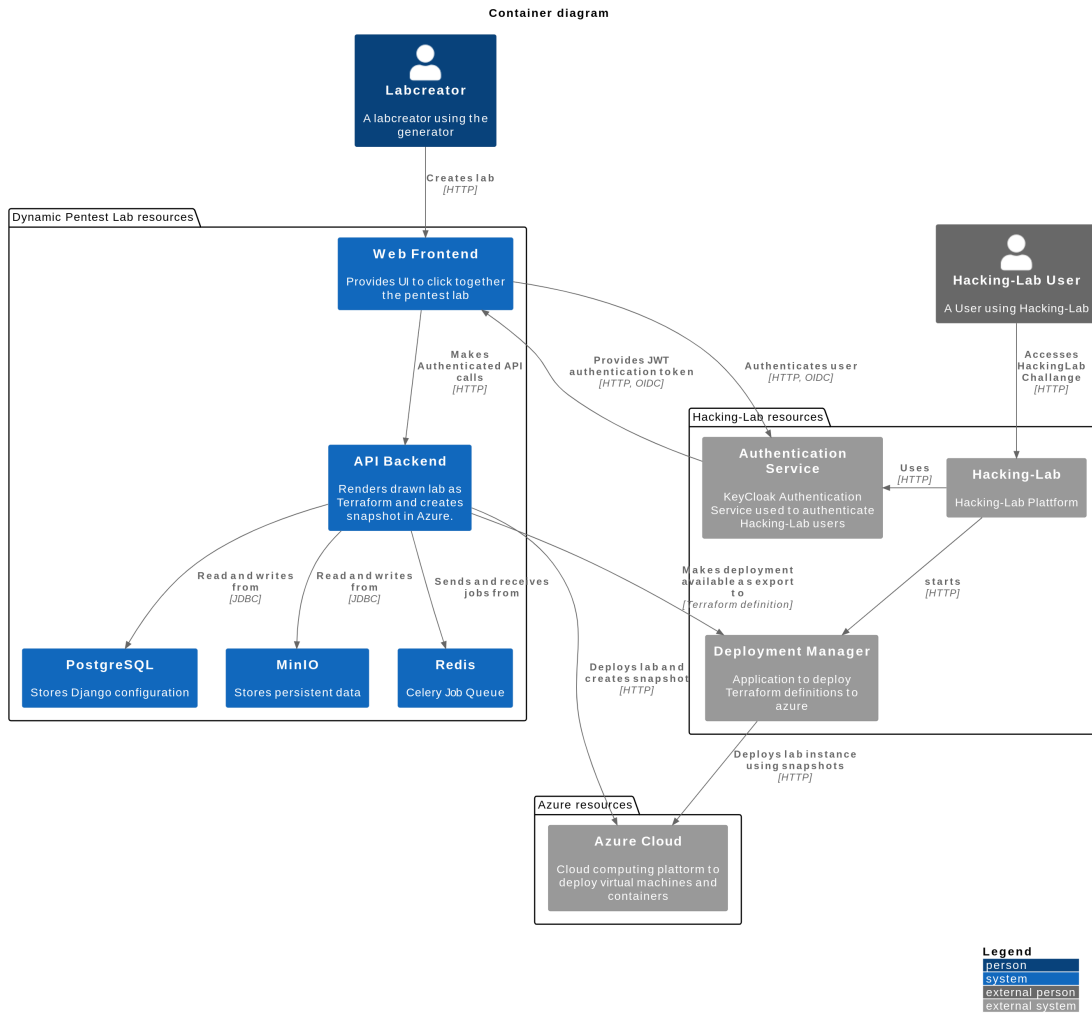


FIGURE 4.3 C4 Container Diagram

---

The container diagram shows following containers:

### **Web Frontend**

- **Frontend:** Provides the user interface for lab-creators to design pentest labs. Lab-creators use the frontend to create labs via Hypertext Transfer Protocol (HTTP). The frontend makes authenticated API calls to the backend and interacts with the authentication service for user verification.

### **API Backend**

- **Backend:** The backend is tasked with processing API calls originating from the frontend. It effectively manages job queues and is responsible for deploying labs and generating snapshots in Azure. Moreover, it facilitates the export of deployment configurations as Terraform definitions for use in the HL Deployment Manager.
- **PostgreSQL:** Stores the configuration data for Django, which powers the backend application. The backend reads from and writes to this database using Java Database Connectivity (JDBC).
- **MinIO:** Stores persistent data such as lab configurations.
- **Redis [17]:** Acts as a job queue for managing background tasks using Celery [18]. The backend sends and receives jobs from Redis.

## Component diagram

In the component diagram in Figure 4.4, the interactions between the high-level application modules are depicted.

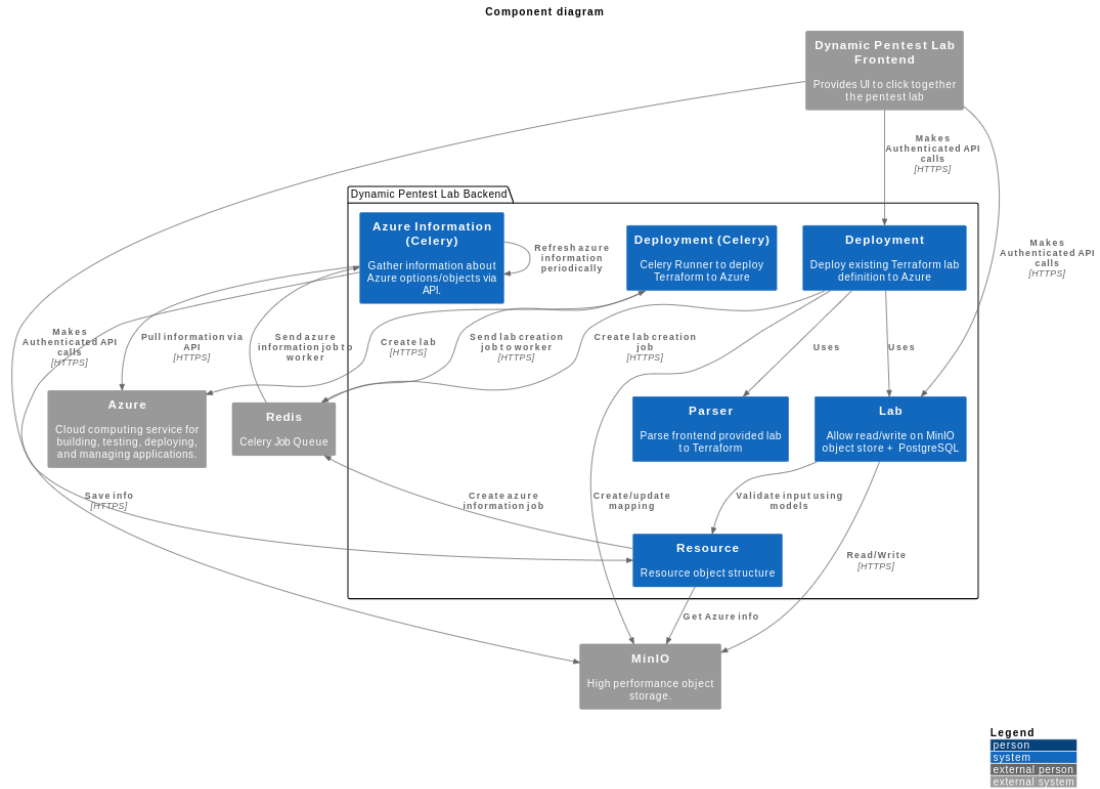


FIGURE 4.4 C4 Component Diagram



---

This components include:

### **Backend**

- **Resource:** Defines the structure of resources used in the labs. Validates input using models, interacts with MinIO [19] to get Azure information.
- **Azure Information (Celery):** Periodically gathers information about Azure options and objects via the Azure API. Saves the gathered information in MinIO and periodically refreshes Azure information. Receives jobs from Redis and interacts with Azure to pull information.
- **Parser:** Parses the lab configuration provided by the frontend into Terraform configurations. Used by the deployment component to convert lab designs into deployable Terraform definitions.
- **Deployment:** Manages the deployment of Terraform lab definitions to Azure. Receives lab creation jobs from the frontend, uses the parser to convert lab configurations, interacts with the lab component for reading/writing data, and updates mappings in MinIO. Creates lab creation jobs in Redis.
- **Deployment (Celery):** Runs Celery tasks received from Redis to deploy Terraform configurations to Azure.
- **Lab:** Provides authenticated API calls for the frontend, validates input using the resource models, and interacts with MinIO for data storage. Manages read/write operations on MinIO object storage and PostgreSQL database.

### 4.3 Azure Lab Architecture

The 4.5 shows the fundamental architecture of the lab deployment, including the different Azure Resources [20] involved, further explained below, and how they interact with each other.

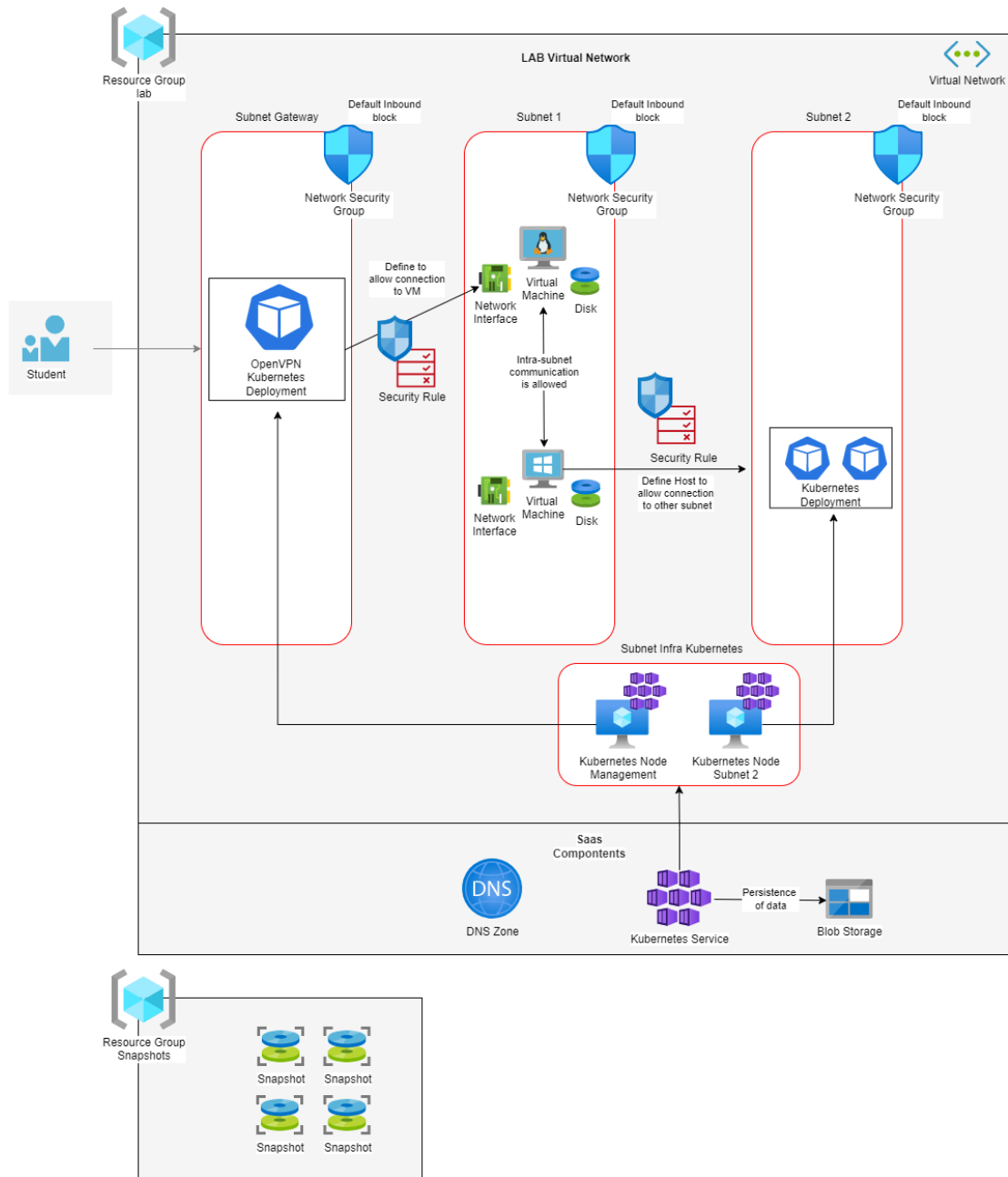


FIGURE 4.5 Azure Lab Architecture

---

### 4.3.1 Base Infrastructure

This section describes all resources required to provide the base infrastructure of the lab. These resources are automatically created for each lab.

#### **Resource group**

For each deployed lab, a unique resource group is created to encapsulate all resources used within this lab. Additionally, a resource group for the snapshots exists. This is created during the creation process through the generator, which is further described in the implementation chapter.

#### **Virtual Network**

The Virtual Network is used to isolate the whole lab. All created subnets reside in this virtual network.

#### **Azure Kubernetes Service**

The Azure Kubernetes Service serves as the Kubernetes platform to deploy the containers. The Azure Kubernetes Service is deployed by default, since the Dockovpn OpenVPN container also runs as a container.

#### **Azure Kubernetes Cluster Node Pool**

There are separate Node Pools per subnet on which the corresponding containers are deployed. Scales automatically with the number of Kubernetes subnets required.

#### **Dockovpn Kubernetes deployment**

The VPN endpoint that is required to access the lab. This is deployed as Kubernetes Deployment and runs on the management node.

**Blob Storage** The blob storage which can be used for persistent data. In this case, it is used for the persistence of container data.

#### **DNS Zone**

The DNS Zone provides DNS Resolution for the lab. The whole DNS architecture is described in more detail in the [ADR 4.7.16](#).

---

### 4.3.2 Core Resources

This section includes all core resources which can be selected in the lab generator to construct a lab.

#### **Subnet**

The subnets provide segmentation within the virtual network, enabling isolation of resources to build and simulate complex networks.

#### **Security Rule**

Security rules are used to restrict traffic between subnets and resources, such as VMs and containers.

#### **Virtual Machine**

Azure Virtual Machines (AVMs) are used to provide Windows as well as Linux machines.

#### **Kubernetes Deployment (Containers)**

Kubernetes Deployments, also called workloads in Azure Kubernetes Service (AKS), are used to facilitate container deployments.

### 4.3.3 Additional Resources

This section outlines additional Azure resources required to support the base infrastructure and core resources.

#### **Network Security Group**

Network Security Groups (NSGs) act as virtual firewalls, regulating network traffic to and from Azure resources according to specific security rules. An individual NSG will be deployed for each subnet.

#### **Network Interface**

Network interfaces serve as the network connection point for the AVM, facilitating communication with the virtual network. This essential resource is dependent on the VM and will be automatically generated along with the VM.

#### **Disk**

Disks that provide persistent storage for the VM operating system disk. This resource is dependent on the VM and will be automatically created along with the VM.

### 4.3.4 Resource Naming

The table in 4.1 illustrates the prescribed naming guidelines and limitations. In accordance with the constraints outlined in [21] for Azure resources, which include specifications regarding length and permissible conventions, a suitable naming scheme has been formulated. Resource groups, with the inclusion of a Universally Unique Identifier (UUID) and other resources necessitating distinct names, are identified by a unique alphanumeric Identifier (ID) consisting of seven characters.

Resource Type	Naming	Restrictions
Resource Group	dpl-rg-{ UUID }	<ul style="list-style-type: none"> <li>- Names must be unique within the subscription.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-)</li> </ul>
Virtual Network	dpl-vn	<ul style="list-style-type: none"> <li>- Names must be unique within the resource group.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-)</li> <li>- Max length is 64 characters.</li> </ul>
Storage Account	dplasa{ [a-z0-9]{7} }	<ul style="list-style-type: none"> <li>- Names must be globally unique.</li> <li>- Alphanumeric, lowercase.</li> <li>- Max length is 24 characters.</li> </ul>
Azure Kubernetes Service	dpl-aks	<ul style="list-style-type: none"> <li>- Names must be unique within the subscription.</li> <li>- Alphanumeric characters and hyphens (-) only.</li> <li>- Max length is 63 characters.</li> </ul>
Azure Kubernetes Cluster Node Pool	dplakcnp{ SUBNET ID }	MAX 12, no hyphens, only lowercase and numeric.
Azure Kubernetes Deployment	dpl-kd-{ [a-z0-9]{7} }	<ul style="list-style-type: none"> <li>- Alphanumeric characters and hyphens (-) only.</li> <li>- Max length is 63 characters.</li> </ul>
Subnet	dpl-s- { [a-z0-9]{7} }	<ul style="list-style-type: none"> <li>- Names must be unique within the virtual network.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-).</li> <li>- Max length is 80 characters.</li> </ul>
Virtual Machine	dpl-vm-{ [a-z0-9]{7} }	<ul style="list-style-type: none"> <li>- Names must be unique within the resource group.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-).</li> <li>- Max length is 15 characters.</li> </ul>

Network Interface	dpl-ni-{ VM ID }	<ul style="list-style-type: none"> <li>- Names must be unique within the resource group.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-).</li> <li>- Max length is 80 characters.</li> </ul>
Disk	dpl-d-{ VM ID }	<ul style="list-style-type: none"> <li>- Names must be unique within the resource group.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-).</li> <li>- Max length is 80 characters.</li> </ul>
Network Security Rule	dpl-nsr-{ [a-z0-9]{7} }	<ul style="list-style-type: none"> <li>- Names must be unique within the resource group.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-).</li> <li>- Max length is 80 characters.</li> </ul>
Network Security Group	dpl-nsg-{ RESOURCE ID }	<ul style="list-style-type: none"> <li>- Names must be unique within the resource group.</li> <li>- Alphanumeric, lowercase, and may include hyphens (-).</li> <li>- Max length is 80 characters.</li> </ul>

**TABLE 4.1** Azure Resource Naming

---

## 4.4 Backend Architecture

The backend architecture of the DPL generator is designed to support the creation, deployment, and management of penetration testing labs in a cloud environment. It leverages modern technologies and frameworks to ensure scalability, maintainability, and ease of use. The backend is primarily built using Django, a robust web framework known for its "batteries-included" approach, providing a comprehensive suite of features necessary for web development.

### 4.4.1 Core Components

The backend architecture is composed of several core components, each responsible for specific aspects of the system's functionality. These components interact with each other to provide a seamless and efficient user experience.

#### 4.4.1.1 API Backend

The API backend serves as the central point of interaction between the frontend and the backend services. It handles all API requests from the frontend, processes the data, and communicates with the other backend components to perform the required operations. Key responsibilities include:

- Rendering lab configurations into Terraform scripts for deployment.
- Managing user authentication and authorization through JSON Web Token (JWT) tokens.
- Handling Create, Read, Update and Delete (CRUD) operations for lab configurations and user data.

#### 4.4.1.2 Object Storage

MinIO is used for object storage within the backend architecture. MinIO provides high-performance, S3-compatible object storage, which is essential for storing large files such as lab configurations. This component ensures that all persistent data is securely stored and easily retrievable when needed.

---

#### **4.4.1.3 Task Queue**

Redis is utilized as a task queue manager in the backend architecture. With Celery as the task queue, Redis handles asynchronous task management, enabling the system to process tasks in the background, such as deploying labs, creating snapshots, and fetching Azure resource information. This ensures that the main application remains responsive and can handle multiple requests concurrently.

#### **4.4.2 Key Interactions and Workflows**

The backend architecture supports several key interactions and workflows essential for the operation of the DPL.

##### **4.4.2.1 Lab Creation and Configuration**

The process of creating and configuring a lab involves several steps:

1. The lab-creator uses the frontend to design and configure a pentest lab.
2. The frontend sends authenticated API requests to the backend, which are processed by the API backend.
3. The API backend interacts with the database to store the lab configuration and with MinIO for storing any large files.
4. The configuration data is then rendered into Terraform variable definition.

##### **4.4.2.2 Deployment and Snapshot Management**

Deployment and snapshot management are critical functionalities of the DPL system:

- The backend uses Terraform to deploy the configured labs to Azure. This involves creating the necessary infrastructure, such as VMs and containers, based on the rendered Terraform variable definition.
- The task queue, managed by Redis and Celery, handles the execution of these deployment and snapshot tasks in the background, ensuring efficient resource utilization and responsiveness.



---

### 4.4.3 Scalability and Maintainability

The backend architecture is designed to be scalable and maintainable:

- The use of Django provides a solid foundation for building scalable web applications, with support for modular design and reusable components.
- MinIO are chosen for their ability to handle large volumes of data and high throughput, ensuring the system can scale as the number of users and labs grows.
- Redis and Celery provide robust task management, enabling the system to process multiple tasks concurrently and efficiently. The Celery workers can be scaled by simply increasing the amount or resources of the docker container enabling vertical and horizontal scaling.

### 4.5 JavaScript Object Notation (JSON) Model

The JSON model plays a crucial role in hydrating the frontend with necessary information from the backend, ensuring seamless interaction and data flow between the two. This model is not only fundamental for data exchange but also serves as the backbone for initializing and configuring the frontend interface. Given its comprehensive structure, the JSON model is inherently complex, designed to accommodate various configurations and user interactions. The following code snippet illustrates the simplest component, `kubernettesubnet`, which has only one user-configurable field, `custom_label`:

```
1 {
2   "kubernettesubnet": {
3     "config": {
4       "type": "subnet",
5       "label_prefix": "dpl-s-",
6       "position_layer": 2,
7       "automatic_connection": [
8         "firewall"
9       ],
10      "can_connect_to": [
11        "firewall",
12        "dockercontainer"
13      ],
14      "exactly_one_instance": false,
15      "icon": "REMOVED FOR READABILITY",
16      "regex": {
17        "custom_label": "^.{0,20}$"
18      },
19      "tooltip": {
```

```
20         "custom_label": "Custom Azure label of the subnet",
21         "label": "Azure name to be of the subnet"
22     }
23 },
24     "type": "str",
25     "custom_label": "str",
26     "label": "str"
27 },
28 ...
```

LISTING 4.1 Model Data Structure

In the architecture of each resource, a "config" object is incorporated. This object is primarily utilized to initialize the frontend interface and does not need to be sent back to the backend during the saving or deploying processes of a lab. The config object contains important data, such as regular expressions for validating each user input field on the frontend, base64-encoded icons, and tooltips for the user. It also specifies configurations like whether there should be exactly one instance of a resource, the types of resources it can connect to, and whether it should automatically create connections with a specific resource type.

The full JSON Model can be found in the appendix.

## 4.6 Frontend Architecture

### 4.6.1 Conceptual Mockup

To provide a clear vision of the frontend's appearance, functionality, and structure, it was essential to establish a point of orientation before starting with any implementation. In order to capture and discuss the potential architecture and appearance, the design tool Figma[22] was utilized to create a mockup, as outlined in Figure 4.6, of the frontend. This mockup served as the foundation for the subsequent development of the frontend and was continuously expanded, adapted and finalized until there was a realistic and usable blueprint of an application that fulfills all the requirements.

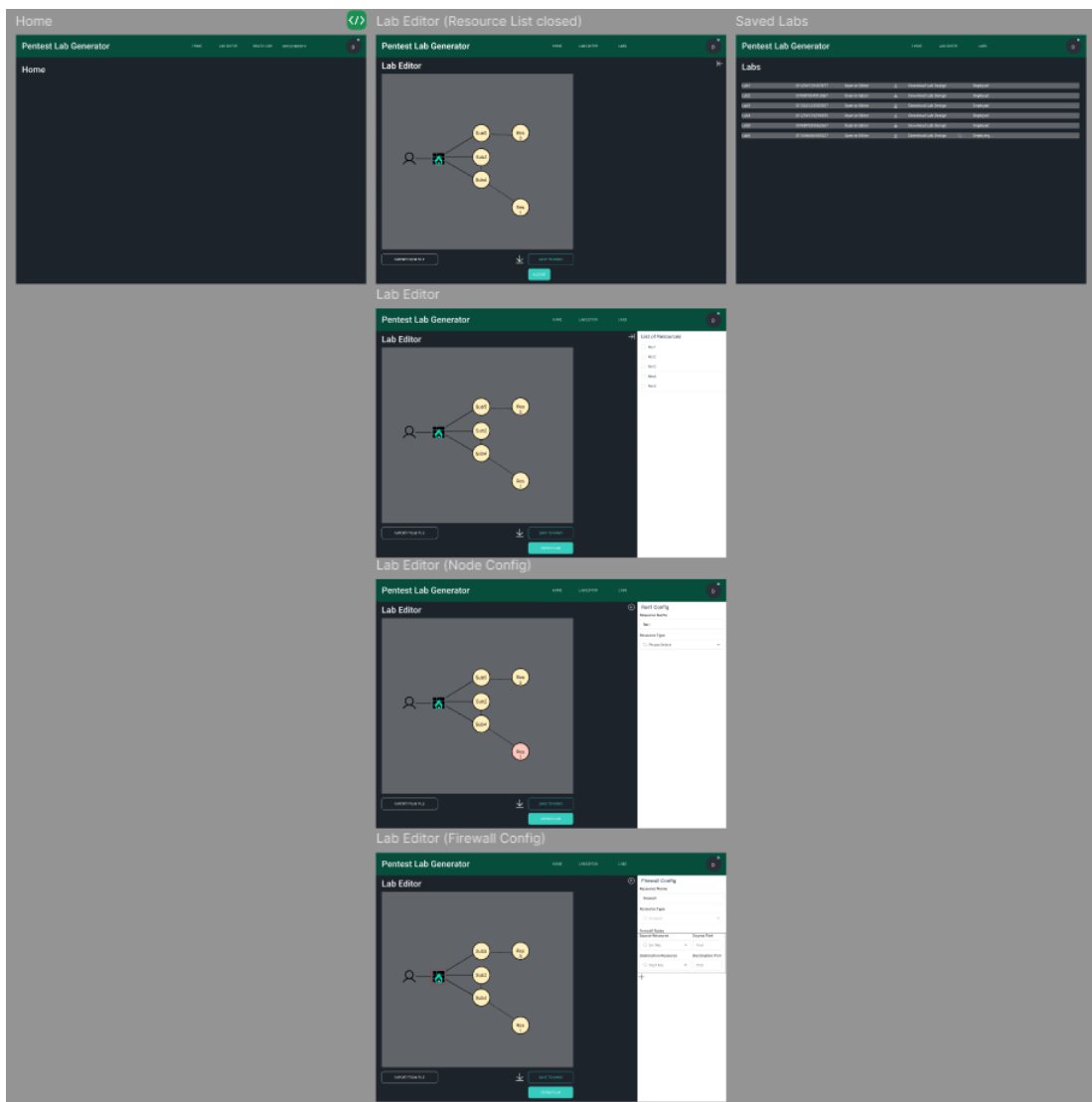


FIGURE 4.6 Frontend Figma Overview

The mockup, displayed in Figure 4.7, designs all required pages and the varied states that could transpire. In future project development stages, this mockup will function as a central single source of truth for the implementation of the frontend.

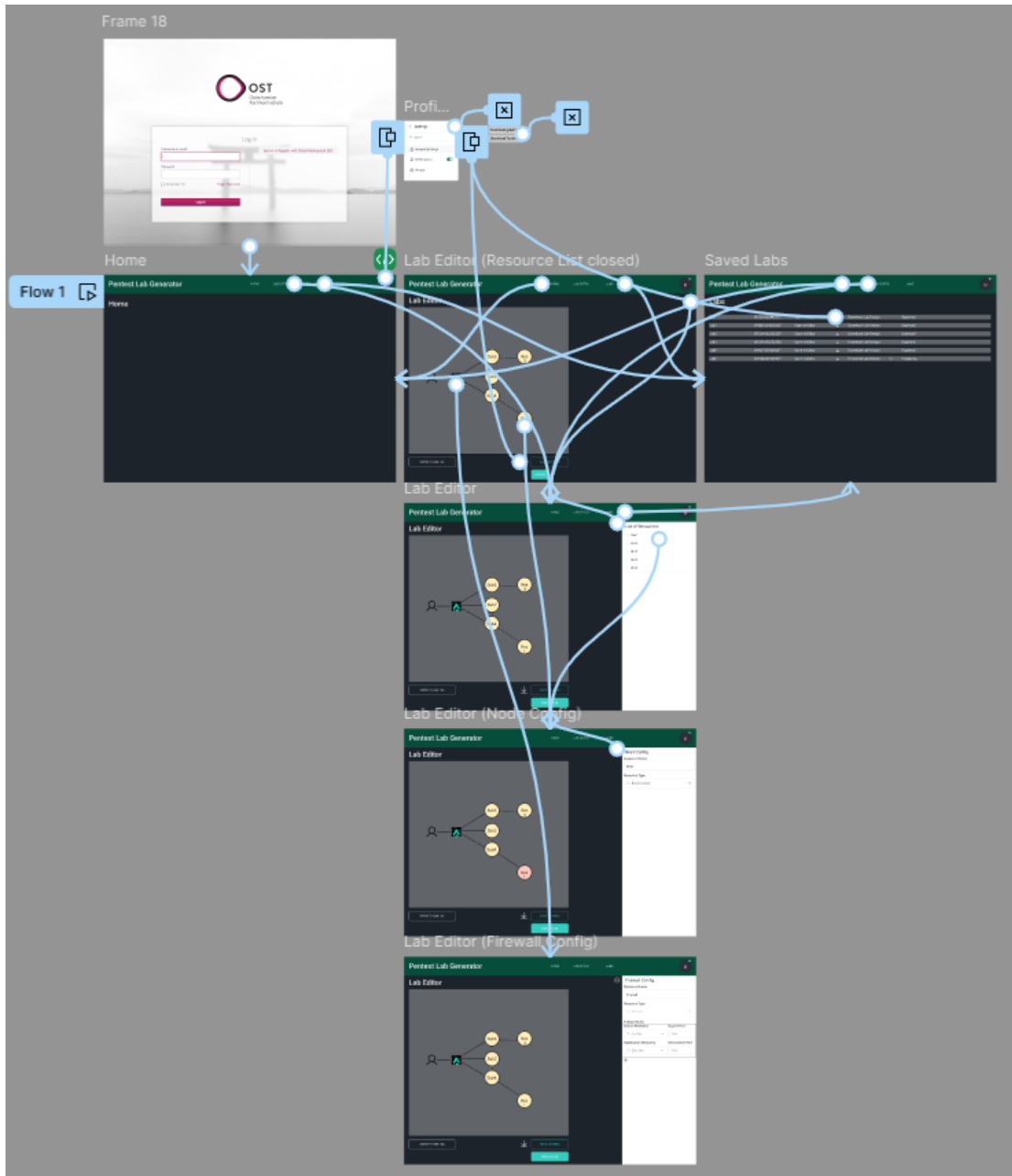


FIGURE 4.7 Frontend Figma Flow

---

To enhance understanding of the application's functionality and user interaction, Figma facilitated the embedding of elementary logic and the establishment of dynamic links between interface elements, animating the static mockups. This interactive approach allowed more detailed discussions around the application's design, paving the way for a refined and definitive product. The mockup includes all the essential components featured in the application, positioned at the approximate coordinates where they will be in the final product.

The main components that were defined in this process included:

- Authentication
- Global Navbar
- Lab Editor Canvas
- Resource Editor Sidebar
- Labs Overview

These components were initially conceptualized to provide a basic framework and subsequently refined throughout the project. This preliminary design phase was crucial for maintaining clear and consistent communication and ensuring a unified vision for both the visual and functional aspects of the application.

## 4.6.2 Final Architecture

The diagram outlined in Figure 4.8 displays the architectural structure of the frontend component of the Generator and how the different parts work together. In general, the frontend is divided into four distinct parts: pages, components, services, and styling.

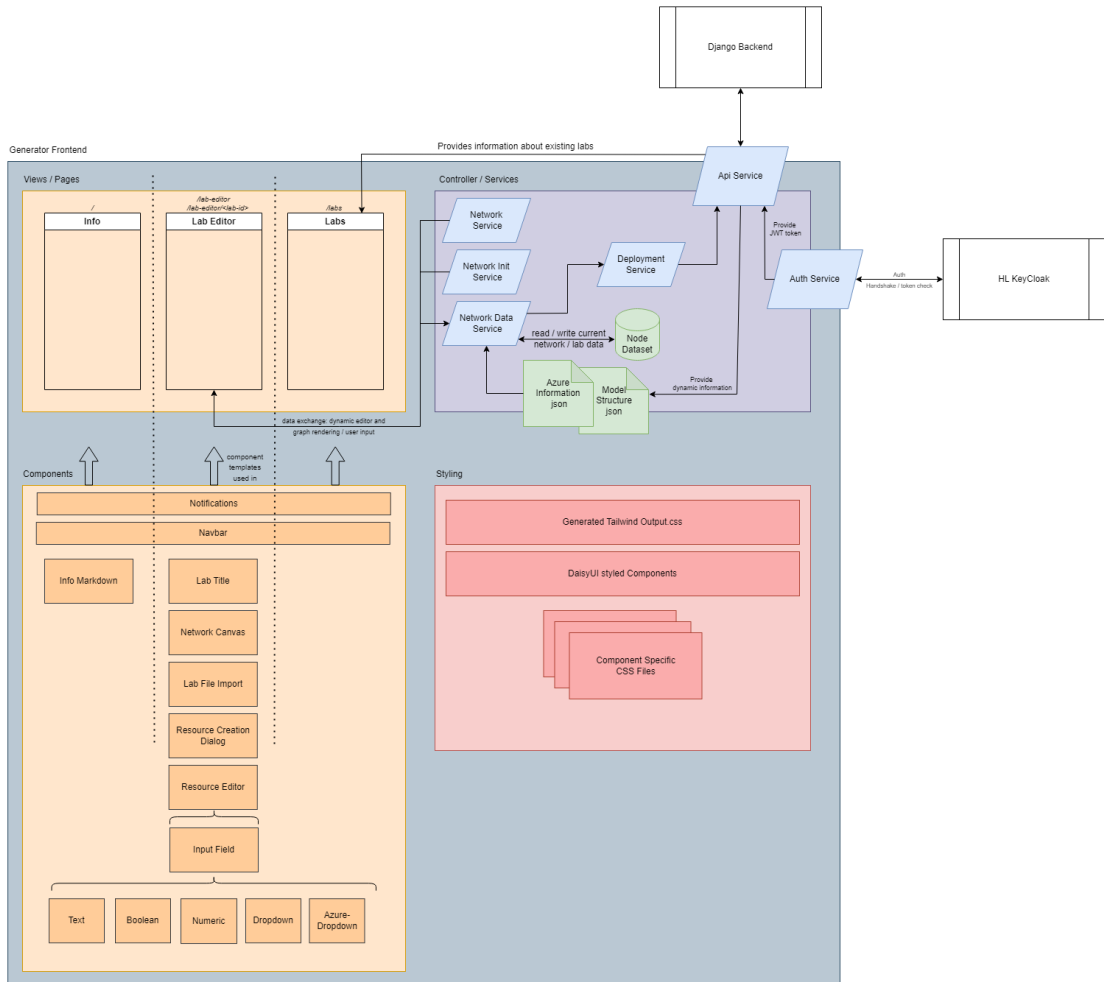


FIGURE 4.8 Frontend Architecture Diagram

---

The outermost layer of the architecture consists of pages, which include and utilize the other three component types: services, template components, and stylesheets. Each of these pages contain static elements and dynamic template components that construct the content relevant to the user. For every page there is a distinct Uniform Resource Locator (URL) used for the routing:

- `/`: The default route displays general information in markdown for the end user about the application and how it is intended to be used.
- `/lab-editor`: Opens an empty instance of the lab editor where the user can create a new lab environment.
- `/lab-editor/<lab-id>`: Opens the lab editor and renders an existing lab using the lab ID and the corresponding JSON lab structure received from the backend.
- `/labs`: Displays the labs page with a table containing all the currently saved labs in the backend's MinIO, allowing the user to interact with each lab through various functions.

To make the HTML content functional and render the information correctly, there is a script for every page and component. Also for every major functionality that reaches across single components there is a service, that holds the required logic and manages in- and outgoing data. With the capabilities of HTML templates, the frontend is dynamically built based on the backend-provided JSON model information as described in Section 4.5.

The styling is implemented in a relatively simple manner by generating a general *output.css* file, which defines the base styling guidelines of DaisyUI [23]. The styling of the different components occurs in the component-specific Cascading Style Sheets (CSS) files. This setup allows a uniform styling of the whole application while also being configurable within each component.

#### 4.6.2.1 Dynamic Rendering

The idea behind the dynamic rendering of the frontend, is to be able to easily expand the whole Generator framework with new environment resources without having to make any changes in the frontend codebase. The two main elements impacted by this dynamic rendering process are the resource editor sidebar, which is utilized during lab creation, and the labs overview table. These components adjust their display in response to the data received from the JSON Structure Model, as elaborated in Section 4.5, as well as the lab's JSON file.

With this approach, the integration of new resource types, such as Virtual Machines (VM) or containers, in forthcoming labs becomes fairly straightforward. In such a scenario, the New Resource Template, as demonstrated in Listing 4.2, can be duplicated,

---

modified accordingly to the requirements, and incorporated into the backend model structure JSON file.

```
1 {
2   "exampleResource": {
3     "config": {
4       "type": "exampleResource",
5       "label_prefix": "dpl-ex-",
6       "position_layer": 3,
7       "automatic_connection": [],
8       "can_connect_to": [
9         "dockercontainer"
10      ],
11      "exactly_one_instance": false,
12      "icon": "REMOVED FOR READABILITY",
13      "regex": {
14        "custom_label": "^.{0,20}$"
15      },
16      "tooltip": {
17        "custom_label": "Custom Azure label of the subnet",
18        "label": "Azure name to be of the subnet"
19      }
20    },
21    "type": "str",
22    "custom_label": "str",
23    "label": "str",
24    "some_number_prop": "int",
25    "some_bool_prop": "bool",
26    "some_encapsulated_props": [
27      {
28        "some_sub_text_prop": "str",
29        "some_sub_bool_prop": "bool"
30      }
31    ],
32    ...
33  }
```

LISTING 4.2 Model Data Structure for a new Resource

The actual and detailed implementation of this frontend architecture is described in the frontend's implementation part, in Section 7.1.



---

## **4.7 Architectural Decision Records**

Essential architectural decisions are recorded using ADRs, according to the ADR template, defined by Olaf Zimmermann (AppArch)[2]. ADRs, primarily, record the results of design and decision-making processes. In this documentation, attributes such as status, date and deciders for each ADR have been omitted because they are typically more relevant for larger projects with multiple participants and are not essential in this context.

### **4.7.1 ADR: 001 - Web Framework Backend**

#### **4.7.1.1 Context and Problem Statement**

The web framework for the backend is an extremely important part of the DPL. It serves as the backbone for creating, managing, and interacting with virtualized environments. The chosen web framework needs to be robust, flexible, and secure allowing to adapt authentication providers.

#### **4.7.1.2 Decision Drivers**

- Scalability to support a growing number of users and lab environments.
- Security features to protect sensitive data and lab integrity.
- Flexibility to integrate with the HL and other tools.
- Ease of development and maintenance to allow for quick iteration and deployment of new features.

#### **4.7.1.3 Considered Options**

- Django
- Flask
- Spring Boot

---

#### 4.7.1.4 Decision Outcome

After careful consideration, it was decided to adopt Django as the web framework for the DPL. Django's extensive standard library and built-in features for user authentication, session management, and security mechanisms address the critical needs for a secure and scalable web application. This decision aligns with our goals of creating a robust, secure, and user-friendly platform creating, managing and starting labs. By choosing Django, a comprehensive framework is leveraged, that supports rapid development, strong security, and scalability, which are essential for the success of the DPL.

#### 4.7.1.5 Consequences

- Good, because Django's "batteries-included" approach accelerates development by providing ready-to-use components.
- Good, because Django's built-in security features help protect against common threats like SQL injection and cross-site scripting, crucial for the security of our application.
- Good, because Django's scalability ensures that the platform can grow to accommodate more users.
- Good, because Django's vibrant community and wealth of plugins extend the framework's capabilities and provide resources for troubleshooting and development.
- Neutral, because Django's monolithic structure might limit flexibility in some cases compared to more modular frameworks like Flask.
- Bad, because Django's learning curve can be steep for developers unfamiliar with the framework or its design patterns, potentially slowing initial development efforts.

### 4.7.2 ADR: 002 - Web Framework Frontend

#### 4.7.2.1 Context and Problem Statement

#### 4.7.2.2 Decision Drivers

- Guidelines set by the HL development team suggest that, for optimal future maintainability, it is preferable to construct the code base using the same technology stack as the existing HL platform. This approach ensures continuity in software maintenance, as the responsibility for managing this software will transition to the HL development team beyond the scope of this thesis.

- 
- Scalability: The framework should offer scalability features to accommodate increased user loads and additional functionalities seamlessly.
  - Well documented: The chosen web framework should have a large community and good documentation quality. A framework with an active and supportive community results in better resources, timely updates, and a robust knowledge base, improving the learning curve of the development team.
  - Performance: The selected framework should provide efficient rendering and loading times, contributing to a responsive and smooth user experience.

#### 4.7.2.3 Considered Options

- React [24].
- Angular [4].
- Vuejs [25].

#### 4.7.2.4 Decision Outcome

The selection for the web frontend framework led to the choice of Angular. Following multiple meetings and with Ivan Bütler and the developers of the HL platform, the development team opted for Angular, primarily driven by the consideration of ensuring the long-term maintainability of the software by the HL developers.

#### 4.7.2.5 Consequences

- Good, because the software can be maintained and used in long-term perspective by HL without the extra cost of learning and implementing new technologies into their stack.
- Good, because Angular's comprehensive documentation and active community support contribute to a well-documented and supported development environment. This facilitates a smoother learning curve for the development team, enabling them to leverage resources and updates effectively.
- Good, because Angular's scalability features address the potential growth and evolving requirements of the web application. This ensures that the selected framework can seamlessly accommodate increased user loads and additional functionalities, supporting the scalability needs of the HL project.
- Good, because Angular's performance optimizations lead to efficient rendering and loading times. This positively contributes to a responsive and smooth user experience, enhancing overall satisfaction with the web application.

- 
- Neutral, because while Angular meets the maintainability and scalability criteria, it may have a steeper learning curve compared to some other frameworks. However, the decision considers the long-term benefits and aligns with the HL development team's expertise.
  - Bad, because the decision to choose Angular might result in a more rigid development structure compared to some other frameworks. However, this potential drawback is outweighed by the benefits in terms of maintainability and scalability, as well as the alignment with existing HL practices.

### **4.7.3 ADR: 003 - Frontend Graph Framework**

#### **4.7.3.1 Context and Problem Statement**

A central part of the project is the Generator Graphical User Interface (GUI) which needs to be a intuitive and simple to use application that one can use without detailed introduction. It would be highly unnecessary to build such an application from scratch and put a lot of resources into building a such a framework. Matter of fact, there are already a few frameworks that provide pre-built logic to create such an application.

#### **4.7.3.2 Decision Drivers**

- Scalability: The frontend needs to be expandable with additional objects without touching the code of the frontend. So it has to be generically designed.
- Maintainability: This tool should be easy to maintain and future follow-up projects should be able to expand our tool with new objects that can be drag and dropped into the graph canvas. In terms of the frontend this means that it needs to be implemented generically, so it can handle new objects coming from the backend without touching the code in the frontend itself.
- The framework should be easy to learn and apply for the developer team of this thesis. Therefore it is sensible to use a framework that is well documented and known by people that work on this thesis.

#### **4.7.3.3 Considered Options**

- Vis-network [5].
- React DnD [26].
- JointJS [27].
- Cytoscape.js [28]

---

#### 4.7.3.4 Decision Outcome

The final graph framework used for this thesis is the vis-network [5].

#### 4.7.3.5 Consequences

- Good, because it is well maintained and is updated on a regular basis.
- Good, because it is supported and compatible with all other technologies and frameworks used in the frontend.
- Good, because it is not very complex but still allows full customization and expansion for the requirements of this thesis.
- Bad, because it does not implement all the functionalities that are required for the full implementation of this thesis and therefore have to be manually implemented.

#### 4.7.4 ADR: 004 - Frontend CSS Framework

##### 4.7.4.1 Context and Problem Statement

One of the most important elements of an application with a GUI is a visually appealing and user friendly design and layout. In modern web applications this is implemented with a CSS framework that comes with its own styling and predefined components. It is responsible for a good user experience and a generally good optical appearance.

##### 4.7.4.2 Decision Drivers

- Boundaries by the HL development team: for future maintainability reasons, the code base ideally should be built on the same tech stack as the current HL platform, because the HL development team will be responsible to maintain this software after the time of this thesis.
- **Ease of Use:** CSS framework should prioritize ease of use, ensuring that developers can quickly and efficiently implement styling without steep learning curves. This will contribute to faster development cycles, reducing the time and effort required for styling tasks.

- 
- Another key criterion is the ability of the CSS framework to enhance the overall visual appeal of the application. The chosen framework should offer a range of modern design elements, components, and layout options to make the user interface visually engaging. Additionally, it should facilitate a seamless and intuitive user experience, enhancing the overall usability of the application.
  - An essential factor is the framework's commitment to accessibility standards. The chosen CSS framework should provide features and practices that support the development of accessible interfaces, ensuring that the application is usable by individuals with diverse abilities.

#### **4.7.4.3 Considered Options**

- Tailwind [29] + DaisyUI [30].
- React Bootstrap [31].
- Material UI [32].

#### **4.7.4.4 Decision Outcome**

After evaluating as a team, discussing the topic with the supervisor and the HL development team, the decision became clear that this project's frontend has to be based on the same tech stack as the current HL platform. Therefore the decision was made that the frontend will be built with the Tailwind framework in combination with the DaisyUI component library.

#### **4.7.4.5 Consequences**

- Good, to keep consistency within our project together with the HL platform. This is highly in favor for the maintainability by the HL development team and the user experience within the whole HL platform, so the user has a similar visual appearance of the different components.
- Good, Tailwind in combination with DaisyUI aligns with the ease of use criterion, providing a comprehensive set of utility-first styles and pre-designed components within the Angular ecosystem, facilitating a straightforward implementation process for developers.
- Good, the utilization of Tailwind with DaisyUI enhances visual appeal and usability, offering a wide array of customizable utility classes and components. This combination ensures a modern and polished appearance for the application, leading to improved user satisfaction.

- 
- Good, Tailwind with DaisyUI follows a utility-first design approach, providing responsive utility classes that seamlessly adapt to various screen sizes. This promotes a positive user experience, aligning with the project's goal of creating an accessible and user-centric application.
  - Good, Tailwind with DaisyUI places emphasis on accessibility, adhering to best practices and guidelines. This ensures that the application will be inclusive and usable by a diverse audience, aligning with the project's commitment to accessibility standards.
  - Neutral, while Tailwind with DaisyUI allows for extensive customization through utility classes, it may have a different design approach compared to component-oriented frameworks like React Bootstrap. However, the team believes that the provided customization options are sufficient for meeting the project's design requirements.
  - Bad, the decision to use Tailwind with DaisyUI may introduce a slight learning curve for team members unfamiliar with the utility-first approach. However, the overall benefits in terms of development speed and design consistency are deemed to outweigh this potential drawback.

#### **4.7.5 ADR: 005 - Repository Structure**

##### **4.7.5.1 Context and Problem Statement**

In defining the project setup, a decision arose regarding the choice between a mono- and multi-repository structure. This decision significantly influences the development process and the overall organization of the project.

##### **4.7.5.2 Decision Drivers**

- **Maintainability:** Ensuring the maintainability of the project is crucial, not only for the completion of this thesis but also for potential future follow-up projects.
- **Scalability:** Evaluating whether the project's size and complexity warrant the additional effort required for implementing a multi-repository structure.

##### **4.7.5.3 Considered Options**

- **Multi-Repository:** Opting for a distributed approach where the different components, like frontend, backend and the deployment, reside in separate repositories.
- **Mono-Repository:** Choosing a unified repository for the entire project, housing all components under a single version control system.

---

#### 4.7.5.4 Decision Outcome

After careful consideration and deliberation, the project team has decided to adopt a **Multi-Repository** structure for the following reasons.

#### 4.7.5.5 Consequences

- Good, because **Improved Isolation**: Segregating components into separate repositories allows for better isolation, enhancing code modularity and maintainability.
- Good, because **Enhanced Scalability**: The multi-repository approach provides flexibility in scaling the project as it grows, minimizing the impact of increasing complexity.
- Good, because **Clear Module Boundaries**: Clearly defining module boundaries in individual repositories contributes to a more organized and comprehensible project structure.
- Good, because **Neutral Versioning**: Each repository can have its versioning strategy, avoiding potential versioning conflicts that may arise in a mono-repository setup.
- Bad, because **Potential Overhead**: However, managing multiple repositories introduces administrative overhead, necessitating effective coordination and synchronization between teams.

#### 4.7.6 ADR: 006 - VM Customized Installation

##### 4.7.6.1 Context and Problem Statement

For further customize the the VMs deployed through the generator, the the lab-creator should be able to perform custom installations.

##### 4.7.6.2 Decision Drivers

- **Maintainability**: The custom installation options of the VMs should remain easy to update without significant time and effort.
- **Ease of Use**: The custom installation options should be user-friendly for lab-creators and managers.
- **Extendability**: Adding new packages or scripts should be straightforward with minimal complexity.



---

#### 4.7.6.3 Considered Options

- Allow package installations via default Linux package manager and Chocolatey [33] for Windows.
- Permit custom install scripts in individual Github repositories for each VM.
- Install all packages and services using the Nix [34] package manager.
- Utilize Snap [35] for installing packages and services.
- Run all packages and services in Docker on the VM.

#### 4.7.6.4 Decision Outcome

The decision was made to implement custom installations via Github repositories. This approach serves to decrease the complexity of the Generator, thereby reducing its maintenance requirements. Additionally, it offers the beneficial consequence of facilitating developers in testing their scripts with ease and providing them the liberty to implement any installation via the custom repository.

#### 4.7.6.5 Consequences

- Good, because it introduces minimal complexity to the Generator.
- Good, because it shifts the responsibility of correct package installation and configuration to the lab-creator.
- Good, because lab-creators can more easily debug package installation and configuration.
- Good, because lab-creators have flexibility in configuring and installing packages/services.
- Good, because lab-creators can reuse repositories if the VM has the same use-case.
- Neutral, custom installation scripts have to be surrounded with adequate error handling.

### 4.7.7 ADR: 007 - Public Docker Images

#### 4.7.7.1 Context and Problem Statement

To enable lab customization, the lab-creator should have the ability to deploy public docker images.

---

#### 4.7.7.2 Decision Drivers

- **Maintainability:** The addition of custom Docker images should not require excessive time and effort to keep up to date. Options should be designed to remain available in the future.
- **Ease of Use:** Docker containers should be easily integratable into a new lab for someone creating and managing labs.
- **Extendability:** Adding new Docker images should be a straightforward and accessible process.

#### 4.7.7.3 Considered Options

- Use Docker Hub Registry.
- Use Azure Docker Registry.
- Create Docker images from source code.

#### 4.7.7.4 Decision Outcome

The Docker Hub Registry is selected as the preferred option due to its public accessibility, providing a user-friendly, self-service-oriented solution for lab-creators to use their own images.

#### 4.7.7.5 Consequences

- Good, since Docker Hub Registry is available to everybody without access considerations.
- Good, as the Docker image environment is isolated from the actual application, requiring minimal maintenance.

#### 4.7.7.6 More Information

Considering that the HL registry is adapted as described in [4.7.8](#), the internal HL development of the image, and consequently, the intellectual property, remains private.

---

## **4.7.8 ADR: 008 - Private HL Docker Images**

### **4.7.8.1 Context and Problem Statement**

The HL has a collection of docker images which should be accessible to the lab-creator. This approach is necessary because hosting vulnerable Docker images on the public Docker registry, where they could be accessed by anyone, would enable students to easily bypass challenges by inspecting the images.

### **4.7.8.2 Decision Drivers**

- **Integration:** Should be simple to integrate and coexist with the current HL infrastructure.
- **Complexity:** Should not introduce a high level of complexity and dependencies.
- **Costs:** The solution should not incur significant additional costs.
- **Security:** The intellectual property of HL should be securely maintained.

### **4.7.8.3 Considered Options**

- Push all HL images to Docker Hub into a private repository.
- Expose the HL registry to the internet and connect using credentials.
- Build own Docker Registry in Azure and allow access using S2S VPN tunnel
- Mirror HL registry to Azure Docker Registry.

### **4.7.8.4 Decision Outcome**

Togheter with Ivan Bütler, the supervisor, it was decided that pushing the private images from the HL to Docker Hub as private registry, is the optimal solution as this solution is the least complex and integrates the best into the current workflow.

---

#### 4.7.8.5 Consequences

- Good, because it provides easy accessibility to HL images for lab-creators.
- Good, because it allows for a straightforward integration with the existing HL infrastructure.
- Good, because there are no significant additional costs associated with this solution.
- Good, because the intellectual property of HL is secured through the authentication on the private registry.

#### 4.7.9 ADR: 009 - Restrict Communication between Resources

##### 4.7.9.1 Context and Problem Statement

Within a pentest lab, network segmentation and limited communication between segments are essential for imitating real-world networks. Given Azure's capabilities of defining precise security rules within security groups to limit interaction, a decision must be made regarding the extent and complexity of these restrictions.

##### 4.7.9.2 Decision Drivers

- **Real-world based:** The solution should reflect current communication restrictions encountered in actual corporate networks.
- **Simplicity:** While providing advanced functionalities, the solution should not introduce unnecessary complexity or overhead in setting up the restrictions.
- **Adoption:** The feature should be straightforward to implement within the generator.

##### 4.7.9.3 Considered Options

- Restricting communication through security groups and rules between subnets exclusively.
- Implementing communication constraints through security groups and rules on individual resources, thereby deploying a Host-based firewall.
- Enforcing communication limitations through security groups and rules on a per-port and source/destination basis via the parent subnet, hence establishing a conventional network firewall.

---

#### **4.7.9.4 Decision Result**

The choice to limit communication based on per-port and source/destination parameters provides a substantial level of control, enabling the emulation of complex real-world networks using network firewalls. Rules can be established to permit communication between resources, as well as between resources and subnets, or the reverse. The security rules are then applied at the subnet level, accommodating both containers and VMs, as there is no feasibility to apply security groups directly to containers. This restriction concerning the application to containers, invalidates the approach of applying security groups and rules directly to individual resources. Restricting solely between subnets would not have replicated a state-of-the-art network adequately.

#### **4.7.9.5 Consequences**

- Good, because it allows for a fine-grained control of communication.
- Good, as it ensures user-friendliness within the generator, facilitating the creation of rules akin to a firewall.
- Good, because sophisticated real-world networks can be emulated.
- Good, because the solution supports containers as well as VMs.

#### **4.7.9.6 More Information**

By default, each subnet should be isolated, and only outbound connectivity to the internet should be possible in order to install software. The security groups should be applied at the end of the deployment to ensure that custom installations can communicate freely between the resources for installation purposes.

---

## **4.7.10 ADR: 010 - Container Deployment**

### **4.7.10.1 Context and Problem Statement**

The lab generator requires support for container deployment. Considering the decision-making factors, the objective is to discern an optimal deployment solution that strikes a balance among cost, deployment duration, compatibility, and manageability.

### **4.7.10.2 Decision Drivers**

- Inter-compatibility with HL
- Network isolation
- Costs for deployment
- Deployment time
- Stability, Manageability error handling of deployments

### **4.7.10.3 Considered Options**

- Azure Container Instances (ACIs) [36]
- AKS [9]
- Dedicated AVM with Docker installed

### **4.7.10.4 Decision Outcome**

After evaluating the available options, the decision was made to utilize AKS for deploying containers. This decision is further outlined in the comprehensive evaluation in section [4.7.10.6](#).

### **4.7.10.5 Consequences**

- Good, because existing HL images can be reused.
- Good, because costs are in the average range.
- Good, because deployments can easily be managed with Terraform.
- Neutral, because separate subnets have to be used for containers.

---

#### 4.7.10.6 More Information

In order to evaluate the best possible solution to run containers in the lab, each of the following requirements is assigned a weight. Furthermore, the key points of the evaluated solutions are explained.

##### **Intercompatibility**

It is vital that existing HL images of vulnerable services can be reused and deployed in pentest labs. This includes, for example, support for the S6 layer [37], which is utilized in all HL images, as well as the usage of private repositories. Given that this is a high requirement, it is weighted with a priority of 5 out of 5. All of the options support the usage of private repositories such as Docker Hub. The only considered options supporting the S6 overlay are AKS and the self-installed Docker host on an AVM. However, ACI unfortunately does not support the S6 Overlay since it utilizes Shared Process Namespace. [38].

##### **Isolation**

The deployed containers must be capable of being placed in separate networks to simulate segmented networks with multiple subnets. In ACI, this can be achieved by placing a node of the cluster in the corresponding subnet. In ACI, the container can be assigned to the corresponding subnet. However, it is important to note that placing the containers together with other AVM is not supported, as subnets cannot be used for normal AVMs if delegated to a service such as AKS or ACI. Placing containers together with other AVMs would only be possible with the option of self-installing Docker on an AVM. Considering that containers in real-world scenarios typically run apart from VMs in separate subnets managed by, for example, Kubernetes utilizing networks with different network policies, this requirement is not of significant importance. Therefore, the requirement is weighted with a priority of 2 out of 5.

##### **Costs**

The solution should be cost-optimized to avoid significant expenses since multiple students will utilize and deploy multiple instances of the lab, cost optimization is a significant factor, weighted with a priority of 4 out of 5. For this calculation, a deployment with 6 containers over 3 subnets for 24 hours was assumed. In AKS, it is necessary to have a dedicated node for each separate subnet. In this example, it results in 3 regular nodes plus 1 management node. The nodes were calculated with 2 cores and 4 GB RAM each. For ACI, 6 containers were calculated with resources of 1 GB RAM and 1 CPU each, which is the minimum for the calculator and thus results in a higher price. For the option of an AVM with Docker installed for each container, a separate AVM is required, resulting in 6 AVMs. These AVMs are also calculated with 2 cores and 4 GB RAM each, considering the overhead for running the Docker host itself. The Figure 4.9 shows the estimated price for each evaluated solution.













Your Estimate				
 Azure Container Instances 	6 Container group(s) x 86,400 ...	 	Upfront: \$0.00	Monthly: \$7.45
 Virtual Machines 	6 A2 v2 (2 Cores, 4 GB RAM) x ...	 	Upfront: \$0.00	Monthly: \$12.40
 Azure Kubernetes Service (... 	Standard; Cluster managemen...	 	Upfront: \$0.00	Monthly: \$8.27

FIGURE 4.9 Pricing Azure Container Deployment

### Deployment Time

Deployment time must be as low as possible since this is the time the students have to wait till they can use their lab. Therefore, the requirement is weighted with a priority of 4 out of 5. AKS deployments, including containers, generally take around 6.5 minutes. Similarly, AVM Docker hosts could see faster deployment times by utilizing a predefined image and snapshots. However, ACIs offer the fastest option, as they eliminate the need for deploying infrastructure altogether, directly spinning up the container.

### Stability, Manageability Error Handling of Deployments

The deployment of infrastructure along with the containers running on it demands an efficient and manageable solution. Terraform definitions present a robust solution for AKS and ACI deployments, ensuring consistency and ease of management. While extending this approach to Docker hosts is conceivable, it would entail exposing each VM to the internet for manageability, potentially compromising security and introducing complexity into the deployment process. Although installation is necessary for the AVM to act as Docker host, creating a deployment image could streamline the process and improve manageability. However, this approach also requires maintenance. Such complexity may impede error handling and overall manageability when compared to cloud-native solutions like AKS and ACI. Therefore, the requirement is weighted with a priority of 4 out of 5.



In following Table 4.2 the rating is concluded based on the categories above. The rating and the weight are numbers from 1 to 5.

	Azure Container Instances	Azure Kubernetes Service	Azure Vm with docker installed
Intercompatibility (weight: 5)	2	5	5
Costs (weight: 4)	5	4	3
Isolation (weight: 2)	4	4	5
Deployment Time (weight: 4)	5	3	3
Stability and Manageability (weight: 4)	2	4	3
<b>Score</b>	66	77	71

**TABLE 4.2** Container Deployment Evaluation

---

## **4.7.11 ADR: 011 - Persistence of Labs**

### **4.7.11.1 Context and Problem Statement**

The lab-creator will have the capability to export and save the labs they've created to their device. This enables them to reupload the file and modify the lab's configuration. While this method is functional, it's not not allow efficient sharing among all lab-creators, as each would need to individually distribute their files. Therefore, a solution that facilitates easy use and sharing of existing lab creations must be identified.

### **4.7.11.2 Decision Drivers**

- Shareability: The ease with which files can be shared among lab-creators.
- Maintainability: The effort required to manage and update lab configurations and content.
- Ease of Use: The simplicity of the process for lab-creators to share and utilize labs

### **4.7.11.3 Considered Options**

- MinIO: An object storage server that provides high performance and easy access to stored labs.
- Cloud Storage Services: Could include services like Amazon S3 or Google Cloud Storage, known for their scalability and global accessibility.
- Peer-to-Peer Sharing Networks: Technologies that allow direct sharing among users, bypassing centralized storage.

### **4.7.11.4 Decision Outcome**

The choice to employ MinIO was made in conjunction with Ivan Bütler, primarily due to the presence of an existing instance that Hacking-Lab AG is already utilizing.

---

#### 4.7.11.5 Consequences

- Good, because MinIO's use can be integrated in the existing HL infrastructure.
- Good, because it offers high performance, ensuring that lab files are accessible with minimal delay, enhancing the user experience.
- Good, because MinIO supports scalability, accommodating the growing number of labs and users without significant modifications.
- Neutral, because reliance on a single storage solution could pose risks in terms of data sovereignty, vendor lock-in, or potential downtime impacting access to lab files.

#### 4.7.12 ADR: 012 - Terraform Templating

##### 4.7.12.1 Context and Problem Statement

To dynamically generate Terraform lab definitions, it is crucial to establish an appropriate templating solution.

##### 4.7.12.2 Decision Drivers

Key factors influencing the decision include:

- Ability to create complex deployments with dependencies
- Error handling capabilities
- Ensuring consistent and reproducible deployments

##### 4.7.12.3 Considered Options

Several options were considered:

- Terraform
- JINJA2
- Django

---

#### 4.7.12.4 Decision Outcome

The decision was made to leverage internal Terraform expressions for dynamically generating resources. This approach requires only the provision of Terraform variables. With Terraform variables, which may also include arrays or lists, Terraform expressions and evaluations can be utilized to define the generation logic.

#### 4.7.12.5 Consequences

- Good, Terraform can handle dependencies directly.
- Good, all logic can be implemented directly within Terraform.
- Neutral, there may be a learning curve associated with mastering Terraform's syntax and best practices.

### 4.7.13 ADR: 013 - Student Access to the Lab

#### 4.7.13.1 Context and Problem Statement

To grant students access to the lab, an appropriate access method needs to be selected. The task assignment states that this should be facilitated through a VPN connection.

#### 4.7.13.2 Decision Drivers

- The access method should be user-friendly.
- The access method should be compatible with the Kookarai environment.
- The access method should require unique authentication for each lab.
- The access method should be deployable automatically via Terraform definition.
- The access method should be deployable in a short period of time.

#### 4.7.13.3 Considered Options

- Azure Virtual Network Gateway (AZNG) utilizing a Point-to-Site (P2S) connection [39].
- Self-Managed OpenVPN instance using the Dockovpn container to establish a P2S VPN [40].
- Wireguard as container instance [41]

---

#### **4.7.13.4 Decision Outcome**

The choice was made for a self-managed OpenVPN instance due to its widespread and established use, adaptability, cost efficiency, and quick deployment capabilities.

#### **4.7.13.5 Consequences**

- Good, OpenVPN profiles can be automatically generated and distributed to students.
- Good, access is authenticated via generated OpenVPN profiles, ensuring unique credentials per lab.
- Good, the self-managed Docker instance deploys in under 5 minutes.
- Neutral, maintenance of the Dockovpn container is handled by a private developers.

#### **4.7.13.6 More Information**

For evaluation, two Proof of Concept (PoC) were developed. While the AZNG could have been suitable due to its Azure-based nature and Infrastructure as a Service (IaaS) availability, it suffers from lengthy deployment times (15-20 minutes) and inability to retrieve OpenVPN profiles directly from the API. Therefore, the Dockovpn container was chosen, offering rapid deployment and direct retrieval of OpenVPN profiles through Azure Storage Containers, making it the most user-friendly choice for lab users.

The option to use Wireguard was considered, however, the choice ultimately fell on OpenVPN due to its wider integration and longer-standing presence in the field.

### **4.7.14 ADR: 014 - Authentication Frontend**

#### **4.7.14.1 Context and Problem Statement**

Provide Authentication in every state of the generator application from front- to backend via the HL Authentication service KeyCloak [42].

---

#### 4.7.14.2 Decision Drivers

- Library has to be compatible with the existing HL Authentication service via KeyCloak and Angular.
- The library has to be well maintained with regular commits for bug fixes and improvements.
- There should be sufficient documentation and community support.
- It's preferable when the library is used in various larger projects as a proof of concept and as a template.

#### 4.7.14.3 Considered Options

- angular-oauth2-oidc (frontend side) [43].
- Auth0 [44].

#### 4.7.14.4 Decision Outcome

Within the development team of this thesis in collaboration with the HL developer Samuel Phillip a final solution for the authentication has been worked out. Consequently, the decision was taken to utilize the angular-oauth2-oidc library for the front-end, as it offers optimal compatibility with the KeyCloak service.

#### 4.7.14.5 Consequences

- Good, because angular-oauth2-oidc is well supported and maintained with regular contributions.
- Good, because angular-oauth2-oidc is used in various other projects and kept up-to-date by the community.
- Good, because the learning curve is not too steep and therefore does not increase the work-load of the developers by a large amount.
- Good, because it supports and provides all the functionalities and features that are required for our application.

---

#### 4.7.14.6 More Information

Since all HL sub-applications utilize the HL KeyCloak service, it was evident that this application would also authenticate through the same KeyCloak. It would use a distinct realm, designated as "generator" (<https://auth.demo-dc.hacking-lab.com/>).

#### 4.7.15 ADR: 015 - Authentication Backend

##### 4.7.15.1 Context and Problem Statement

The primary focus is to verify the authenticity of the JWT tokens transmitted from the frontend to the backend. These JWT tokens are subsequently received by the HL KeyCloak service.

##### 4.7.15.2 Decision Drivers

- The used library should be well maintained
- The implementation should not need more than a JSON Web Keys (JWKs) Uniform Resource Identifier (URI)
- The implementation should be fast and efficient

##### 4.7.15.3 Considered Options

- PyJWT [45].
- Authlib [46].
- Django-OAuth [47].

##### 4.7.15.4 Decision Outcome

Authlib was chosen because is the easiest to implement, maintain and is well supported and developed by the community.

---

#### 4.7.15.5 Consequences

- Good, because Authlib is well supported and maintained with regular contributions.
- Good, because Authlib is used in various other projects and kept up-to-date by the community.
- Good, because Authlib integrates well into the django workflow.
- Good, because the implementation is only dependant on the JWKs URI.
- Neutral, because Authlib is a rather big package which means it uses more space but also has features which could be used in other places.

#### 4.7.16 ADR: 016 - DNS Resolution

##### 4.7.16.1 Context and Problem Statement

In order to access resources using DNS rather than just IP addresses, a strategy must be developed to ensure proper DNS resolution for the labs. This solution needs to be scalable to accommodate the continuous deployment of additional labs.

##### 4.7.16.2 Decision Drivers

- Scalability: Support for an increasing number of labs, each necessitating a unique subdomain.
- Cost-effectiveness: Utilization of free service tiers to minimize expenditure.
- Maintainability: Simplification of DNS record management and removal through Terraform.
- Security: Avoidance of unnecessary disclosure of the internal network structure.
- Complexity: Avoid introducing excessive complexity when configuring DNS resolution for end-users.

##### 4.7.16.3 Considered Options

- DNS management using a Azure private DNS Zone [48]
- Using an Azure public DNS Zone [49]
- Use Cloudflare [12] DNS for the root domain and Azure public DNS zone for the specific subdomains related to the lab.



---

#### 4.7.16.4 Decision Outcome

The decision to employ Cloudflare for the management of the root domain and Azure Public DNS Zone for the subdomains associated with individual labs is made. This choice is strategically advantageous due to Cloudflare's offering of free services for up to 1000 DNS entries, enabling the simultaneous deployment of 250 labs, with each lab utilizing four nameserver entries in the root domain. This setup suffices for current scaling requirements. Additionally, opting against deploying the DNS root zone in Azure mitigates unnecessary costs, as Azure would require continuous operation, thereby incurring ongoing expenses. The use of Terraform for DNS record management and removal further simplifies the administrative process and enhances maintainability with automatic provisioning and deprovisioning of DNS entries using the Cloudflare [50] and Azure [51] Terraform providers. To simplify the configuration of DNS resolution for end-users, the DNS records directly resolve to the internal IP addresses of resources. However, this poses a risk of potential DNS zone enumeration [52], which could inadvertently reveal the internal structure of the lab and assist participants in identifying available resources. To mitigate this risk, the introduction of randomized alphanumeric strings for resource names (format 'dpl-vm-7 alphanumeric chars') was adopted in the naming concept as defined in Table 4.1 to reduce the feasibility of DNS zone enumeration.

#### 4.7.16.5 Consequences

- Good, cost efficiency by utilizing the free tier of Cloudflare, avoiding additional costs associated DNS management of the root zone.
- Good, scalability is enhanced with the capability to adjust the number of nameservers per lab or increase the number of labs, as long as the total does not exceed the 1000 DNS entry limit provided by Cloudflare.
- Good, Maintainability through the automation of DNS record management and removal with Terraform, enabling systematic updates and configurations across multiple labs with no manual intervention.
- Neutral, the dual-provider setup introduces slightly more complexity in DNS management because Cloudflare is used additionally.

# Technologies 5

---

This chapter provides an overview of the various technologies and tools employed in this project.

## Frontend

- **TypeScript:** Superset of JavaScript which adds static typing. [53].
- **Angular:** Web application framework by Google [4].
- **Tailwind CSS:** An open-source CSS framework [29]
- **DaisyUI:** Tailwind CSS component library [30].
- **Typescript-eslint:** TypeScript support for ESLint linting tool [54].

## Backend

- **Cookiecutter Django:** Framework for jumpstarting production-ready Django projects quickly. [3].
- **Celery:** A distributed task queue system used for handling asynchronous tasks and scheduling. [18].
- **Redis:** An in-memory data structure store used as a message broker for Celery. [17].
- **Ruff:** Extremely fast Python linter and code formatter [55].

## Database

- **MinIO:** S3 object storage [19].
- **PostgreSQL:** Open-source relational database system [56].

---

## Deployment Infrastructure

- **Kubernetes:** Container orchestration platform [57].
- **Cloudflare:** Content delivery network and internet security services [12].
- **Azure:** Cloud computing service [8].
- **Terraform:** Infrastructure as code tool [7].
- **TFLint:** A Pluggable Terraform Linter [58].
- **OpenVPN:** Secure open-source software for VPN [11].

## Tools

- **Docker:** Platform for delivering software in containers [15].
- **Git:** Distributed version control system [16].
- **Visual Studio Code:** Source-code editor [59].
- **PyCharm:** Python Integrated Development Environment (IDE) for software development [60].
- **WebStorm:** Frontend Javascript and Typescript IDE for web application development [61].
- **Postman:** API platform [62].
- **Insomnia:** API platform [63].
- **LaTeX:** Document preparation system [64].
- **Overleaf:** Collaboration tool for LaTeX [65].
- **Qodana:** Static Code Analysis Tool [66].
- **Zotero:** Reference management software [67].
- **Pre-commit:** Package manager for pre-commit hooks [68].

## Collaborative Tools

- **Jira:** Bug tracking and project management [69].
- **Confluence:** Collaboration software [70].
- **Miro:** Collaborative online whiteboarding platform [71].
- **Teams:** Virtual meetings and chatting tool [72].

# Quality measures 6

This chapter outlines the quality measures in place, including topics such as static code analysis, dependency monitoring, testing strategy, and Git workflow.

## 6.1 Git Workflow

In this section, the branching and merge request policies are described.

### Branching

Branches should be created from `develop`, and changes should be pushed to them. The `main` and `develop` branches should only be altered through merge requests, as shown in Figure 6.1. Branches should follow these naming conventions: `feature/NEW-FEATURE-NAME` for features, and `bugfix/BUGFIX-NAME` for bug fixes.

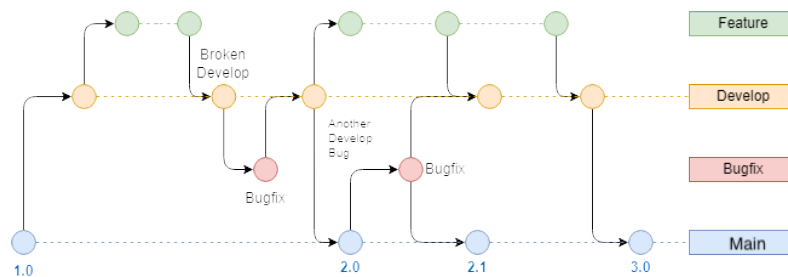


FIGURE 6.1 Git Branching Workflow

### Merge requests

When it comes to merge requests, larger changes require review by another developer, while small configuration changes can be merged into `develop` without the need for a reviewer. Additionally, merging from `develop` to `main` should only occur when the `develop` branch is stable and manually tested by multiple developers.

---

## Four Eyes Principle

A four-eyes principle is also implemented, necessitating code reviews before each merge into the main branch. These code reviews are conducted within GitLab. This ensures that no slip-ups or major logical flaws make it to the main branch, which should always be in a functional and stable state.

## 6.2 Code Quality

### Pre-Commit

To ensure code quality and consistency throughout the development process, pre-commit hooks were utilized to implement automated checks on both the backend and frontend repositories. Pre-commit hooks are scripts that run syntax, style validation, and custom tests before a commit is completed in a version control system like Git. The package pre-commit was used to manage and maintain these hooks, thereby enforcing coding standards and identifying potential issues early on.

### Linters

The linting engine djLint played a crucial role in managing the Django templates in this project. Its powerful linting and formatting capabilities ensured that templates were not only error-free but also adhered to a consistent style and structure. This was particularly beneficial in a collaborative environment, as it minimized the cognitive load on developers when navigating and understanding the project's template code.

On the other hand, Ruff was employed as the primary linting tool for the Python codebase. As a fast and extensible linter, Ruff offered comprehensive coverage of common Python coding errors and stylistic issues, making it an invaluable asset for maintaining code quality.

The backend repository has pre-commit hooks for both linters to ensure only sufficiently linted code was committed and pushed.

For the development of the frontend, TypeScript was utilized in conjunction with ESLint (typescript-eslint) for code linting and static analysis, ensuring clean and maintainable code throughout the development process.

Additionally, Tflint was utilized for the deployment alongside Terraform. Tflint not only offers linting for Terraform itself but also for specific providers, including the provider from Azure.

---

## 6.3 Static Code Analysis

JetBrains Qodana [66], a static code analysis tool, was used to ensure high code quality and maintainability. Furthermore, Qodana has been integrated into our Continuous Integration (CI)/Continuous Delivery (CD) pipeline. This integration facilitates ongoing monitoring and enhancement of our codebase, guaranteeing compliance with coding standards and prompt identification of prospective concerns.

## 6.4 CI/CD Pipeline

The CI/CD pipelines automate the process of integrating code changes, running tests, and deploying the application. These pipelines ensure that both the backend and frontend components of the application are consistently tested and built, providing a reliable and efficient development workflow.

### 6.4.1 Backend

The backend pipeline is defined in the `.gitlab-ci.yml` file and consists of three main stages: lint, test, and build. The pipeline is configured with specific variables to facilitate the setup of the PostgreSQL database and Docker environment.

#### 6.4.1.1 Stages and Jobs

##### Lint Stage:

- **precommit:** Uses a Python 3.11 image to run pre-commit hooks, ensuring code quality and style are maintained before further stages. The hooks are installed and executed, showing diffs on failure to provide immediate feedback.
- **qodana:** Executes Qodana, a JetBrains tool, to perform static code analysis. This job allows failures and stores the results as artifacts for later review.

##### Test Stage:

- **pytest:** Utilizes a Docker-in-Docker (DinD) setup to run the application's test suite. The environment is prepared by building and starting necessary services (Django, Celery, PostgreSQL, Redis, MinIO) defined in the 'local.yml' Docker Compose file. Tests are run with coverage reports generated and stored as artifacts.

##### Build Stage:

- 
- **build\_django\_image:** This job logs into the Docker registry and builds the production Django Docker image. The image is tagged and pushed to the registry, making it available for deployment.

#### 6.4.1.2 Variables

- PostgreSQL and Docker-related variables are set to configure the environment for testing and building the backend application.

### 6.4.2 Frontend

The frontend pipeline, also defined in a `.gitlab-ci.yml` file, similarly includes stages for linting, testing, and building. It uses Node.js and Docker to manage the build and deployment processes.

#### 6.4.2.1 Stages and Jobs

##### Lint Stage:

- **precommit:** Runs pre-commit hooks using a Python 3.11 image, similar to the backend. Additionally, it installs Node.js and its dependencies, ensuring code style and quality before proceeding.
- **qodana:** Performs static code analysis using Qodana, with results stored as artifacts for later review.
- **code\_quality:** Uses a Node.js image to run linting scripts, ensuring the frontend code adheres to defined quality standards.

##### Test Stage:

- **test:** Uses a Cypress image for running frontend tests. This job installs dependencies and executes the test suite, generating coverage reports stored as artifacts. It captures any failures to provide feedback for developers.

##### Build Stage:

- **build:** This job builds the frontend Docker image using a Docker-in-Docker setup. The image is tagged and restricted to run only on the main branch, ensuring that only stable code is built and deployed.

#### 6.4.2.2 Variables

- Environment variables for Docker image names and Node.js version are set to ensure consistent builds across different stages.

---

## 6.5 Testing Strategy

To ensure the robustness of the application, various testing techniques were employed, including unit testing, integration testing and end-to-end testing. Each technique was applied to different parts of the framework, such as the backend, frontend, and deployment.

### 6.5.1 Unit Tests

Unit testing is a fundamental practice in software development that ensures individual components function correctly. In this project, unit tests were essential for maintaining the reliability and integrity of both the backend and frontend.

#### 6.5.1.1 Backend

Pytest was chosen as the testing framework for the backend due to its simplicity and powerful features, including its ease of use, which is characterized by simple syntax and easy setup. It also provides fixtures that allow for reusable test setup, has an extensive plugin ecosystem to extend functionality, and offers great compatibility as it integrates well with Django.

##### Test objects

Comprehensive tests were written for each Django application, ensuring the functionality of related features. Tests were executed locally and automated via the CI/CD pipeline, as detailed in Section 6.4. An overview of the executed tests is provided in the appendix.

##### Test Coverage

Achieving over 80% test coverage with more than 110 tests for a codebase exceeding 5000 lines of source code was a significant milestone. This high coverage indicates that a substantial portion of the codebase is tested, increasing confidence in the application's stability and reliability. Tools such as `coverage.py` were used to measure and report on test coverage, identifying untested parts of the codebase for improvement. The results of `coverage.py` for the main branch are displayed in the repository using a badge.

While a higher code coverage could have been achieved the results for more testing diminish at high levels when compared to the effort. The test-report can be found in the appendix.



---

### 6.5.1.2 Frontend

Karma was chosen as the testing framework for the frontend due to its effective integration with Angular and its robust features, including ease of use with simple setup and configuration, the ability to run tests in real browsers, seamless integration with CI tools, and support for various plugins and frameworks.

#### Test objects

Tests were developed for each Angular component to verify the corresponding features. These tests were carried out locally and automated via the CI/CD pipeline, as elaborated in Section 6.4. A comprehensive summary of the conducted tests can be found in the appendix.

#### Test Coverage

Achieving 36% test coverage with 34 tests indicates that there is room for improvement. Tools such as Istanbul were used to measure and report on test coverage, helping to identify untested parts of the codebase. The results of Istanbul for the main branch are displayed in the repository using a badge. The test-report can be found in the appendix.

### 6.5.1.3 Terraform Deployment

For the Terraform Deployment, a semi-automated approach was taken. This approach involves testing the individual parts of the deployment through actual deployment of the configuration and subsequent verification of the generated output and success of the deployment.

The unit testing phase focuses on verifying individual Terraform resources in isolation before full deployment. This is achieved by applying configurations in isolation, where each resource is applied to validate its correctness. This step ensures that individual components are configured correctly and function as expected. Additionally, output verification is performed to check that the outputs generated by the applied configurations match the expected values. For certain core components the according test-reports can be found in the appendix. Implementing unit testing in such a manner guarantees the reliability of the fundamental building blocks of the infrastructure. This step is crucial because it provides a robust foundation for subsequent phases of integration and end-to-end testing, where the deployment undergoes a more thorough examination.

## 6.5.2 Integration Testing

Integration testing ensures that different modules and services of the application, including the frontend, backend, and deployment, work together correctly. This type of testing verifies that interactions between these components function as expected in a

---

combined environment. The primary objective of integration testing is to identify issues that may arise when integrating various parts of the application. This process helps verify that modules and services communicate correctly, data flows accurately work between components, and combined functionalities meet the specified requirements.

**Big-Bang Integration Testing:**

Big-bang integration testing was performed to some extent. This approach involves integrating all components or modules simultaneously, after which everything is tested as a whole. While this method can quickly show how all components work together, it can be challenging to isolate the source of any detected issues.

For the setup, all modules of the application, including backend services such as Django, Celery, and PostgreSQL, as well as the frontend, were integrated into a single environment. A complete system setup was deployed using Docker Compose, ensuring that all services were running and able to interact with each other.

During test execution, comprehensive test scenarios were carried out to verify the interactions between different components. These scenarios included tests for user authentication, data retrieval and submission, and GUI interactions for creating, uploading, downloading, and deploying labs.

The results indicated several were logged and assigned to the relevant developers for debugging and resolution.

Despite the challenges, Big-Bang integration testing provided a quick overview of the overall system integration. The process highlighted critical integration points that required more focused testing in subsequent iterations. By performing Big-Bang integration testing, key integration issues were identified early in the development cycle.

---

### 6.5.3 End-to-end testing

End-to-end tests were manually performed together with the supervisor, Ivan Büttler, in the demo tenant of HL. These tests aimed to validate the complete workflow of the DPL generator framework from the user interface to the final deployment of labs in the Azure environment. The following key scenarios were tested:

- **User Authentication:** Verified that users could successfully log in using their HL credentials and access the lab creation interface.
- **Lab Creation:** Ensured that users could create labs by adding and configuring various resources such as VMs, containers, and subnets.
- **Resource Configuration:** Checked that custom configurations for resources (e.g., custom installation scripts for VMs, multiflags and credentials) were correctly applied.
- **Terraform Deployment:** Confirmed that the generated Terraform configurations were valid and successfully deployed the labs in Azure.
- **VM Access:** Verified that the OpenVPN container provided secure access to the deployed labs.
- **VPN Resolution:** Ensured that VPN settings for the labs were correctly configured using Cloudflare and Azure VPN zones.
- **Snapshot:** Tested the snapshot functionality by starting the Deployment Manager in HL and then restoring the snapshot

Any issues encountered during testing were promptly addressed and resolved, ensuring a final product of good quality.

# Implementation **7**

---

## 7.1 Frontend

### 7.1.1 General

The frontend of the application was developed using Angular [4], TypeScript [53], and DaisyUI [30]. The focus lied on establishing general best practices and workflows tailored to this specific tech stack. This stack was selected by the HL developers to ensure maintainability beyond the duration of this thesis.

The overall concept and design of the frontend was evaluated and defined using the mockup created in Figma according to Section 4.6.1. This mockup served as a tool to present and discuss various design approaches with both the developer team and the supervisor, ensuring a unified vision for the frontend development. It provided a single source of truth for building components according to a plan. Additionally, the mockup served as a reference for tracking workflows and workload, allowing the team to compare the current state of the application to the envisioned final product.

#### 7.1.1.1 Model-View-Control Pattern

For the clarity and maintainability of the code base, it was decided to enforce a separation of purpose and divide the code into components and services, according to their function. The folder structure represents the architecture of the application, which helps future developers to understand the code and work on it.

This application implements the MVC pattern, which is a design pattern used to separate concerns and improve the modularity of the application. The components of the MVC pattern in the application are as follows:

- **Model:** The Model is responsible for managing the data and business logic of the application. In this implementation, the Model is represented by services that implement data fetching, state management, and business logic. These services are utilized across multiple components, ensuring a centralized and consistent

---

data handling mechanism. The model is realized through most of the services within the file structure of the frontend, which handle communication with the backend, format data that is received and sent, and manage the overall state of the page and its content.

- **View:** The View is the visual representation of the data. In this application, the view consists of the HyperText Markup Language (HTML) page and templating component files in the folder structure. These files define how data is presented to the user and ensure a clear, dynamic and user friendly interface.
- **Controller:** The Controller acts as an intermediary between the model and the view. It processes user inputs, manipulates the Domain Object Model (DOM), and coordinates interactions between the model and the view. In the Generator frontend, the controller role is fulfilled by TypeScript files associated with each component and HTML page. These files handle user interactions and ensure that the view is updated in response to changes in the model.

By applying the Model View Control (MVC) pattern, a well-structured and maintainable codebase was achieved. This separation of concerns not only makes the code easier to understand and manage, but also simplify the testing and future extension of the application.

#### **7.1.1.2 Vis-Network Library**

Integrating the central JSON structure for resource data with the vis-network library posed a significant challenge in frontend development. This process required a thorough understanding of the vis-network library, particularly its primary components: nodes and edges.

The vis-network library takes a dataset of nodes and edges to render a graph on the canvas. For the implementation of this work, the dataset was extended to include all properties necessary for the corresponding resources. The network disregards any unknown properties that are only relevant for deployment, using the remaining properties to render the nodes and edges.

#### **7.1.2 Maintainability**

A main focus of the frontend was set to the maintainability and modularity of the application. The goal was to make extending the application, with further resources that can be added, as simple and intuitive as possible.

As for the implementation of that mentioned goal, an architecture was worked out revolving around JSON files. Generally, these JSON files contain fundamental structures of the data about deployments and the lab setting. These JSON files act as a centralized and single source of truth for the deployment and especially for the frontend. More

---

information about the exact structure and content of those JSON files can be found in the Section [7.2](#).

For the frontend specifically, these JSON files function as a source of hydration to dynamically provide the content that should be displayed for the user.

#### **7.1.2.1 Model Data Structure JSON**

This centralized JSON file, mentioned above, is utilized in various parts of the frontend, primarily on the Lab Editor Page outlined in Section [7.1.4.2](#). This file is essential for hydrating nodes, also known as resources, with the correct data structure and information. More specifically, the model structure JSON provides ordered and encapsulated key-value pairs for all properties associated with each resource.

This architectural design has a significant advantage: the lab resource editor page contains minimal static logic, allowing the component to be built dynamically. Consequently, if any changes are required or new resources are to be added, this can be easily achieved by simply updating the JSON file in the backend with the new resource and its properties. As a result, maintaining the application requires minimal effort and only a basic understanding of the frontend implementation.

The only requirement for a new resource is that it must be deployable in Azure and compatible with the deployment logic in Terraform. Otherwise future developers are completely free with adding new resources and additional logic to the frontend.

Implementing this dynamic hydration in the frontend required significant initial development time and resources due to the complexity of rendering data dynamically based on the resource structure and input types. However, this investment has paid off, as once the base case for a resource was implemented, it became functional for all defined resources and any future additions. Future developers can now easily add new resources to the existing JSON file and utilize all available input field types.

Therefore, the frontend is not only a static graphical editor but rather a versatile toolbox that enables developers to easily create their own version of this cloud network-defining application.

---

### 7.1.3 Development Setup

For the local development of the frontend it was required to set up multiple components for the whole application, early in the project.

#### 7.1.3.1 Backend

For the hydration in the frontend to properly work, it is required to have the complete backend running. As part of the workflow it is therefore required to start all the Docker containers that the backend builds including: django, the Celery worker with redis and the MinIO database. These components are further described in the backend implementation Section 7.2.

#### 7.1.3.2 Frontend

To efficiently develop the frontend a traditional local web server is hosted and allows instant recompiling after any changes have been made.

#### 7.1.3.3 Nginx

After implementing Hacking-Lab authentication via KeyCloak, it became necessary to route traffic over Hypertext Transfer Protocol Secure (HTTPS) instead of HTTP. This change was required to mitigate Cross-Origin Resource Sharing (CORS) errors when redirecting between the Hacking-Lab authentication page and our application. This was achieved with an Nginx[73] that can be run locally or with a Docker[15] container

Additionally, a local DNS entry was created to map the localhost address to the URL <https://generator.demo.hacking-lab.com/>.

---

## 7.1.4 Pages

The Dynamic Pentest Lab Generator consists of three main pages, which form the core of this thesis from the end user's perspective. The lab-creator primarily interacts with the content displayed on these pages to create a new lab structure.

### 7.1.4.1 Info Page

The info page, displayed in Figure 7.1, is the main and default route for this web application. It contains a rendered markdown file that provides a brief introduction, clarifications, and explanations about the application. This page aims to inform the user about the general workflow and offers a short explanation with examples of the various functionalities of the application.

The info page simply renders a markdown file that is filed in the assets folder within the project folder structure. To enable the rendering of markdown within HTML the ngx-markdown[74] library was used.

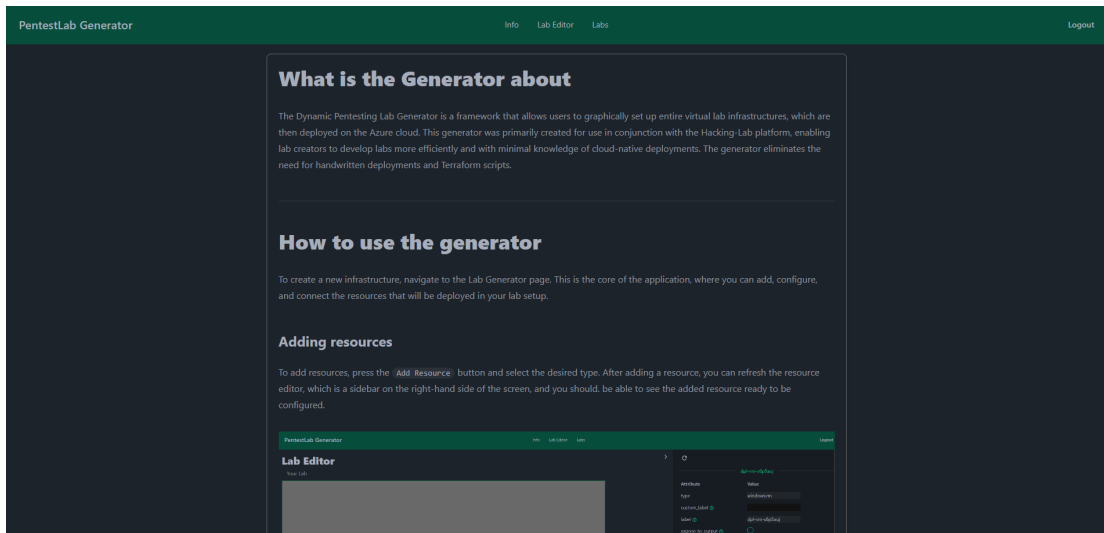


FIGURE 7.1 Info Page



### 7.1.4.2 Lab Editor Page

The editor page, outlined in Figure 7.2, forms the core page in the frontend of this thesis. It contains the core frontend logic and allows the end user to create labs.

It contains the following components which then together build the lab editor:

- Network Canvas: This component lays directly within the lab editor page and consists of components from the vis-network library.
- Lab File Import Component
- Resource Creation Dialog Component
- Resource Editor Component

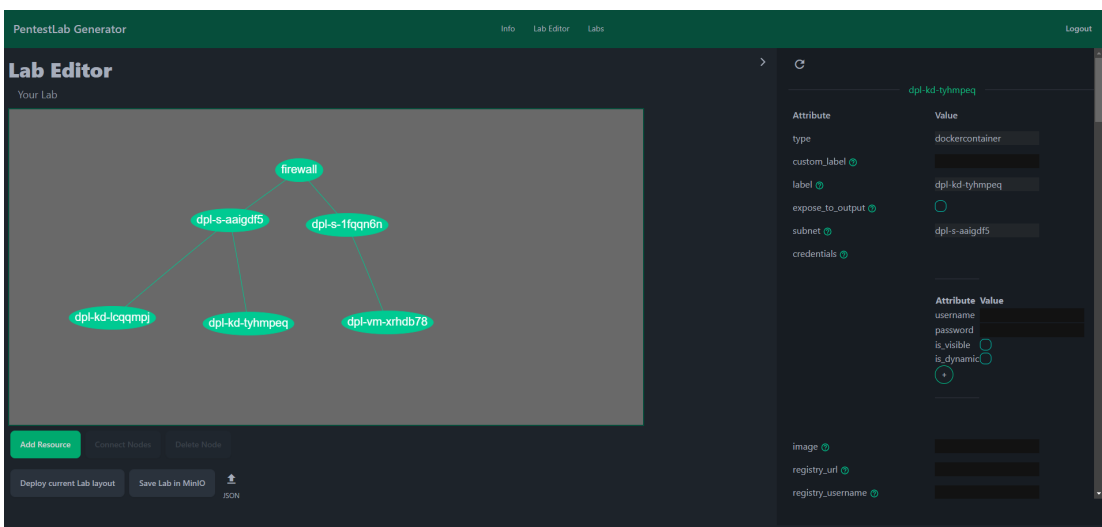
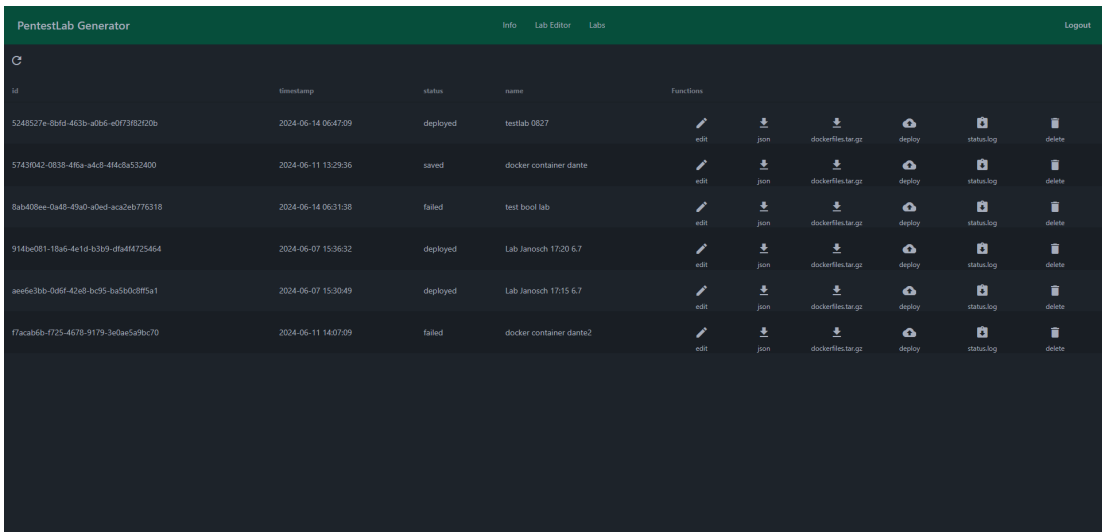


FIGURE 7.2 Lab Editor Page

### 7.1.4.3 Labs Page

The labs page, shown in Figure 7.3, is the other important half of the Generator application. This page serves as an overview of all the saved, which are provided through the GET /lab API call to the backend. The labs page mainly consists of a dynamically generated table, which reads the received data about the existing labs. The HTML loops over the objects within the labs JSON and creates a row for every object and with a second loop iterates over every key value pair property of that object. With every property a column is created and filled with the value of that property. Additionally to each row, the standard function buttons get appended which concludes edit, download as JSON, download dockerfiles.tar.gz, deploy and delete.



The screenshot shows the 'Labs' page of the PentestLab Generator. The page has a dark green header with the title 'PentestLab Generator' and navigation links for 'Info', 'Lab Editor', 'Labs', and 'Logout'. Below the header is a table with the following columns: 'id', 'timestamp', 'status', 'name', and 'Functions'. The 'Functions' column contains icons for 'edit', 'json', 'dockerfiles.tar.gz', 'deploy', 'status.log', and 'delete'. The table lists six labs with their respective IDs, timestamps, statuses, and names.

id	timestamp	status	name	Functions
5248527e-66fd-463b-a0b6-e0773f82920b	2024-06-14 06:47:09	deployed	testlab 0827	edit, json, dockerfiles.tar.gz, deploy, status.log, delete
5743f042-0838-4f6a-a4d8-4468a532400	2024-06-11 13:29:36	saved	docker container dante	edit, json, dockerfiles.tar.gz, deploy, status.log, delete
8ab408ee-0a48-49a0-a0ed-aca2eb776318	2024-06-14 06:31:38	failed	test bool lab	edit, json, dockerfiles.tar.gz, deploy, status.log, delete
914be081-18a6-4e1d-b3f9-dfa4f725464	2024-06-07 15:36:32	deployed	Lab Janosch 17:20 6.7	edit, json, dockerfiles.tar.gz, deploy, status.log, delete
aeef63bb-096f-42e8-bc95-ba5b0c8f5a1	2024-06-07 15:30:49	deployed	Lab Janosch 17:15 6.7	edit, json, dockerfiles.tar.gz, deploy, status.log, delete
f7aca66b-f725-4678-9179-3e0ae5a9bc70	2024-06-11 14:07:09	failed	docker container dante2	edit, json, dockerfiles.tar.gz, deploy, status.log, delete

FIGURE 7.3 Labs Page

---

## 7.1.5 Components

To organize the code into components, the code is grouped based on functionality and location within the codebase. The components in this project form a template that can be inserted anywhere within the existing codebase. This approach helps mitigate code duplication, enhances readability, and improves the maintainability of the application. The following criteria are used to determine whether a piece of code should be used as a component:

- The code can run independently, thus forming a valid component.
- The component is utilized multiple times in different parts of the application.

### 7.1.5.1 Navbar

This is a traditional navigation bar at the top of the page which contains all the different sub-routes of the application. It is used and displayed on every page. The nav-bar is rendered globally through the parent component of the application, the `app.component`. With that the nav-bar is a static element available on every page.

### 7.1.5.2 Notification Component

Similar to the nav-bar component in Section [7.1.5.1](#), the notification component is globally rendered in the `app-component` but is not always visible to the user. It provides feedback through success messages, error alerts or informational messages.

The visibility of the notification is controlled by the `isVisible` input property. When set to `true`, so for example when calling the `showNotification` function in an error, the notification appears on the screen. The component also accepts `title`, `message`, and `type` as input properties:

- **title**: The title of the notification.
- **message**: The main content of the notification.
- **type**: Defines the notification type (e.g., success or error), which determines the background color of the notification.

The notification component uses the Notification Service, further described in [7.1.6.2](#), to manage its state.

Users can manually dismiss the notification using a close button, enhancing user control and accessibility. The component is styled with Tailwind CSS to ensure a consistent and responsive design across different devices. DaisyUI on the other hand did not provide a component that was suiting this use-case as intended.

---

In summary, the notification component is essential for providing user feedback within the application, utilizing Angular's input properties and services for flexible and maintainable notifications.

### 7.1.5.3 Lab Title

The lab-title component is only used in the lab-editor page and is not used in multiple parts of the application, but for better code readability it was also separated into a separate component. It represents the title of a lab which can be edited and is sent to the backend with the rest of the lab structure. This component reacts on changes of its value and updates the value via the NetworkData Service, outlined in Section 7.1.6.5, where the rest of the current lab structure information is saved.

### 7.1.5.4 Resource Creation Dialog

The resource creation dialog component is used in the lab editor page and is a crucial element for the creation and designing a lab environment.

This is the first component that dynamically gets built depending on the resources information provided by the backend, outlined in Section 7.2.4.1, which is received via an API call in the API Service, outlined in Section 7.1.6.7.

After successfully receiving the model data from the backend-call the HTML then creates a button by iterating through every resource object in the model JSON and reads its key. These keys then represent the options that the user has, to add different resources to his lab environment.

```
1 <button
2   *ngFor="let resourceType of Object.keys(modelStructureJson)"
3   (click)="selectResourceType(resourceType)"
4   class="btn m-1"
5 >
6     {{ resourceType }}
7 </button>
```

LISTING 7.1 Dynamic Button Creation through JSON File

### 7.1.5.5 Resource Editor

The resource editor is a core component of the dynamically built frontend and is visually represented as a sidebar within the Lab Editor Page. It implements complex logic to build an editor form capable of representing various resource types with each having different properties and types of input fields that can be configured.

---

The primary concept of this editor is to utilize specific input fields, or building blocks, defined by the model structure JSON file provided by the backend. Users can then add multiple resources of different or identical types to their lab structure.

For instance, considering the basic case of a text input field for some property that should be configurable with text input:

```
1 <app-text-input
2     *ngIf="shouldDisplayInputField(node, property, 'text')"
3     [name]=property
4     [value]="node[property]"
5     (valueChange)="editNode(node, property, $event)"
6     >
7 </app-text-input>
```

LISTING 7.2 Resource Editor Text Input Example

In this context, two nested for-loops render the resource editor dynamically:

- The outer loop iterates over all nodes (or resources) currently in the lab structure dataset, which contains all added resources.
- The inner loop iterates through each property of the current node (resource) from the outer loop.

During each inner loop iteration, the properties mapped to the model structure from the backend, determine which Input Field Component should be rendered for each property. The Input Field Component is further explained in Section 7.1.5.6.

Examining the HTML instance of the Text Input Component Example in Listing 7.2, the following directives and properties are present:

- **Structural Directive:** Used for each input type to check if the current property should be rendered with this component. It takes the current resource or node to provide the necessary information for rendering the specific component. The property and the string text are used in the function to determine if the property type matches text.
- **Name Input Value:** Provides the component with the name of the current property and is used for edge cases to identify properties in the model structure that are necessary elsewhere but do not need rendering in the resource editor.
- **Value Input:** Crucial for loading existing labs, this input takes any value already set on the current property in the dataset and sets it as the property's value when the component is rendered.
- **Value Change Output:** This output serves as an event handler. When a change is detected in an input field, it bubbles up from the input component to the resource editor component. Information about the resource (node), the current property, and the edited property value are passed as arguments to the *editNode*

---

function, which then uses the Network Data Service to update the dataset. Saving this information updates the network graph, which is essential for saving or deploying the current lab structure.

Taking a look at a more complex input component featuring an Azure Dropdown Component outlined in Listing 7.3, which is a special case that requires a backend-call to receive information about the currently available Azure resources, which again come in via the API service. The dropdown component is built so it can be reused normally for future resources as all the other components, but it catches the special case of an Azure dropdown, that functions slightly differently. For more detailed information about the Azure information JSON, refer to the implementation documentation of the backend in Section 7.2.4.1.

```
1 <ng-container
2   *ngIf="shouldDisplayInputField(node, property, 'azure_offer_dropdown')"
3 >
4   <app-dropdown
5     [disabled]="!getChosenVmDropdownValue(node, 'publisher')"
6     [items]="getChosenVmDropdownValue(node, 'publisher') ?
7       azureData[getVmType(node.type)][ 'publisher' ]
8       [getChosenVmDropdownValue(node, 'publisher')][property] : ''
9     [(value)]="azureDropdownTypes.includes(node[property]) ? '' :
10      node[property]"
11     (valueChange)="editNode(node, property, $event)"
12   >
13   </app-dropdown>
14 </ng-container>
```

LISTING 7.3 Azure Offer Dropdown Component

Going into the Listing 7.3 above more in detail, one can see that current property of the iteration step through all the properties of a resource gets checked with the `shouldDisplayInputField` function, if it meets the condition to render the dropdown component, same as for every other property. If this check succeeds the dropdown component gets rendered with the following input and output values:

- **Input [disabled]:** The used function checks a different value, in this case the publisher dropdown value, for the current node (resource) and returns if it is set. If this function returns false the current dropdown gets disabled. This allows the developer to enforce serial selection of Azure dropdowns, depending on the previous dropdowns to have a value selected.
- **Input [items]:** The items input value serves the dropdown to provide it with the options for the user to select. The conditial only serves the purpose to prevent the code from trying to load the items when the previous dropdown, or some other value, is not set yet. If the items are to be loaded, the code looks at the Azure data JSON and returns a list of all matching values, in this case all the offers.

- 
- **input [value]:** This input value is necessary for the loading of existing labs and setting the value of the property that is defined by the loaded lab structure.
  - **output (valueChange):** This final output value is required to register the event of the user changing anything in the currently checked property and writing it into the lab structure that is currently active.

#### 7.1.5.6 Resource Editor Input Fields

The user input fields are a core component of the resource editor, these are dynamically filled into the resource editor template depending on the previously mentioned JSON structure. There are different types of input fields that are called by the value of the according property key that are looped through.

- **Text Input:** This is the base case of the input fields called by the value `str` by the JSON model structure.
- **Numeric Input:** The numeric input is a classic numeric only user input that is called with the value `int` in the JSON model structure.
- **Boolean Input:** This input field type is represented as a classic checkbox that sets the values `true` and `false`. It is inserted in the input template when the value `boolean` is called.
- **Dropdown** The dropdown input field is a simple dropdown component that takes a list of elements that can be display as a scrollable list of items. It allows the user to chose one of the listed elements and select this as the chosen value for the resource property configuration.

There are several special cases for the dropdown, that can be triggered with special keywords in the value that represents the input type of the Lab Structure JSON file. These edge cases of the dropdown component are implemented for the Azure dropdowns for the different VMs and do not take a direct item list but instead pull the newest available Azure information from the backend directly and use these as their items.

Additionally there is a search function implemented for the dropdown item list, which is especially important for the Azure dropdowns, because the list of certain properties can be quite long.

- **Encapsulated Input:** The encapsulated input component is one of the more special and interesting components of the frontend part of the DPL. It is used for resources like the firewall or the docker container resource that have properties within a property. An example being the credentials property that allows the user to add multiple users with credentials that get stored within the credentials property and therefore had to be implemented as a one-step recursion of the

---

classic input fields. Each encapsulated input field takes the arguments of classic input fields like mentioned above.

#### **7.1.5.7 Lab Import**

The lab import component is a simple button that allows the user to upload a JSON file, which is used in the Lab Editor Page directly, to import an existing lab structure via file.



---

## 7.1.6 Services

As for the services, they are created for central functionalities that are required for multiple components and contain functions shared by those components. The services are thematically separated and give the code more structure, so not all the central functions and helper function are clustered in one single file. This makes maintaining and understanding the application easier when searching for a specific function.

### 7.1.6.1 Authentication Service

Like the name tells, this service covers all the functionalities regarding the authentication for every API-call, the initial login, logout and for the validity check of the session token gained by the HL KeyCloak.

### 7.1.6.2 Notification Service

This service provides methods to show and hide notifications and uses BehaviorSubject to track the state of the notification's visibility, title, message, and type. When showNotification() is called, the service updates the relevant subjects, and the component subscribes to these changes to update its display accordingly. Due to the component's global implementation, this service can be called from everywhere in the application and is displayed wherever an error or success message is required.

```
1 <div class="w-full">
2   <app-notification
3     class="notification"
4     [isVisible]="(notificationVisible | async) || false"
5     [title]="(notificationTitle | async) || 'default error'"
6     [message]="(notificationMessage | async) || 'default error message'">
7   </app-notification>
8   <app-nav-bar></app-nav-bar>
9   <router-outlet></router-outlet>
10 </div>
```

LISTING 7.4 Notification Component

Examining the HTML code above it is visible that the notification component is always present and can just be enabled and disabled whenever it is necessary in the application with the mentioned showNotification() function.

---

#### **7.1.6.3 Deployment Service**

This service outsources helper-functions that would clutter the code which trigger deployments and it mitigates unnecessary code duplication, improving code quality and readability.

These helper functions simply format the, by the user created, lab structure and serve it in a data structure which is usable for the backend. The most important function being the `formatJsonForDeployment()` that merges the set Lab Title and the dataset with all the resources into a final JSON that then gets sent to the backend via the API Service.

#### **7.1.6.4 Network Service**

The network service mainly contains the configuration and options of the canvas and graph that is given as a function to render the `vis-network[5]` graph.

#### **7.1.6.5 Network Data Service**

This service contains all the functions that handle the central data logic concerning the `vis-network[5]` library in combination with our lab data structure. This service maps the received data from the backend into a dataset that can be visualized with `vis-network[5]`.

#### **7.1.6.6 Network Init Service**

The network init service functions as a primary initialization of a `vis-network` graph and renders an empty graph with an empty node and edge dataset.

#### **7.1.6.7 API Service**

The API Service is the core service that functions as a bridge between the frontend and backend. It contains all the API calls required to hydrate the frontend with necessary data and to post updates to the backend. This service ensures seamless communication and data exchange between the two layers of the application.

The API Service includes the following key functionalities that also represent the core operations of the application that can be used by the lab editor:

- 
- **Fetch Model Structure:** Retrieves the model structure from the backend to ensure the frontend has the correct data structure and information needed for rendering resources dynamically.
  - **Fetch Azure Data:** Obtains specific data related to Azure resources, ensuring that the frontend can properly display and interact with Azure-related components.
  - **Deploy Lab Structure:** Sends the current lab structure data to the backend for deployment. This method supports both POST and PUT requests, depending on whether an existing lab ID is provided.
  - **Save Lab Structure:** Saves the lab structure data to the backend. Similar to deployment, this can involve creating a new lab (POST) or updating an existing one (PUT).
  - **Get Labs Data:** Fetches a list of all labs from the backend, allowing the frontend to display available labs to the user. This API call receives all the data that is required to build the content of the Labs page.
  - **Download Lab JSON:** Downloads the JSON representation of a specific lab, allowing users to save or review the lab configuration offline.
  - **Download Lab Tar GZ:** Downloads a tar.gz archive of the lab's Dockerfiles, useful for deployments and backups.
  - **Download Lab Status Log:** Retrieves and downloads the deployment status log for a specific lab, providing insights into the deployment process and any issues that occurred.
  - **Load Existing Lab:** Loads the data of an existing lab from the backend, facilitating the editing or redeployment of an existing lab configuration.
  - **Deploy From Minio:** Deploys a lab directly from Minio storage, leveraging the existing lab data fetched from the backend.
  - **Delete Lab:** Deletes an existing lab from the backend, removing its configuration and associated data.

#### 7.1.6.8 URL Service

The URL Service is primarily used for handling operations related to an existing lab that already has an ID. Its core functionality is to extract the lab ID from the URL and return it to any function within the application that requires it.

Additionally, the URL Service includes functionality for managing the lab ID in the URL for a related use-case. When a lab has previously been opened either through the "edit" button on the labs page or directly via a URL containing the lab ID, there is a function to remove the lab ID from the current URL. This prevents the previously

---

opened lab from being reloaded when the user tries to import a lab structure directly from a JSON file, in the current state of the page.

#### **7.1.6.9 Deployment Service**

This service contains all functionalities related to deploying or saving labs within the frontend. The core component of this service is the formatting function for labs. This function takes the dataset of resources and connections (nodes and edges) temporarily stored in the frontend while the user is editing the current lab. It then merges this data with the lab name to create a correctly formatted JSON, which the backend can further process.

#### **7.1.7 Styling**

As for the styling the authors of this thesis were asked to use the same framework as the HL developers used in their products. So like the Angular[4] framework it was a requirement by the HL to use Tailwind[29] and DaisyUI[23].

For the implementation, Tailwind[29] with the defined theme is introduced to the application via the `angular.json` file and within the `tailwind.config.js` the more specific DaisyUI theme is defined. The developers then generate an `output.css`, based on the mentioned `tailwind.config.js` file, that sets a base styling over all HTML files in the application.

For more specific styling of the different components and pages of the application, each component and page comes with its own CSS file, which overwrites the base CSS if there are any conflicts.

---

## 7.2 Backend

### 7.2.1 General

The backend serves as a crucial intermediary between the user-facing frontend and the underlying infrastructure management performed by Terraform. Its primary responsibilities include providing API endpoints for the frontend, initiating Terraform deployments, and generating the Deployment Manager in the form of a docker-files.tar.gz. This tarball can be utilized by lab-creators within the HL environment.

To enhance development efficiency and maintainability, the backend architecture follows a clear separation of concerns. The frontend is designed to handle minimal logic, relying extensively on API calls to the backend for complex operations and data manipulations. This approach ensures that changes in the backend data structure or logic do not necessitate significant modifications to the frontend, unless a major, breaking change occurs. Consequently, this design facilitates easier updates and scalability, as the backend can evolve independently of the frontend.

### 7.2.2 Basic Setup and Structure

The backend is built using Python, Django, and Celery, interfacing with MinIO, PostgreSQL, and Redis for data and job management. This technology stack was chosen in collaboration with the developers at HL to ensure easy maintainability beyond the duration of this thesis.

Key components of the backend include:

- **Python and Django:** Provide the core framework for the backend application.
- **Celery:** Manages asynchronous task execution and scheduling.
- **MinIO:** Handles object storage for lab configurations and related data.
- **PostgreSQL:** Manages relational database operations.
- **Redis:** Acts as a message broker for Celery and provides fast data access.

### 7.2.3 Software

This section lists the external software needed for the backend to function effectively.

---

### **7.2.3.1 MinIO**

MinIO is an open-source, high-performance object storage system that is fully compatible with Amazon S3 APIs. It excels in large-scale data storage. MinIO is known for its simplicity and scalability and supports essential features to ensure data integrity and durability.

In this project, MinIO is used to store lab configurations, snapshots, information gathered from Azure, and other smaller datasets in JSON format.

### **7.2.3.2 Celery & Redis**

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation but supports scheduling as well. The execution units, called tasks, are executed concurrently on one or more worker nodes using multiprocessing, eventlet, or gevent. Celery requires a message broker to handle messages between the main application and Celery workers. Redis is commonly used for this purpose due to its high performance and low latency. For this reason and good integration with Celery it was decided to use Redis for this project as well.

For this project, Celery tasks are used to run Terraform deployments asynchronously. Celery beat is used to run certain functions periodically, similar to cron jobs. Celery is the only component of the backend that needs to initiate communication with external connections, and therefore, it is configured to communicate over a proxy set by the environment variable `HTTPS_PROXY`.

### **7.2.3.3 Postgres**

PostgreSQL is a powerful, open-source object-relational database system. It has a strong reputation for reliability, feature robustness, and performance. PostgreSQL supports advanced data types and performance optimization features, making it the standard choice for handling various internal tasks within Django.

In this project, PostgreSQL is used to manage relational data, providing robust and efficient data storage and retrieval capabilities that are essential for the backend's operations. Its advanced features ensure that the backend can handle complex queries and large datasets efficiently.

### **7.2.4 Django Applications**

A Django project is essentially a collection of various configurations and apps that together make a complete web application. In Django, an "app" is a self-contained

---

package that encapsulates a specific functionality or a set of related features. These apps are designed to be reusable, potentially in different projects, which promotes modularity and reduces code duplication. In the following sections the apps of our project will be described.

#### **7.2.4.1 Resources**

This app ensures that the backend offers two distinct types of resources through its API: the model data structure and the information about various OS versions supported by Azure that is periodically retrieved.

##### **Model Data Structure**

One of the most important and time-intensive parts of the project was the definition and creation of a model which is both easily maintainable, can be easily extended and has the necessary logic to deliver the needed information to the frontend. It was decided not to use JSON Schema to structure JSON that is sent over API as our implementation of it couldn't be both clean and tailored to this very specific use case. It is already described in the JSON model section [4.5](#) in the architecture.

##### **Azure Resources**

To display the supported types of VMs on the frontend, the backend uses Celery's scheduling capabilities to periodically fetch this information from Azure.

The implementation collects all available offers, their corresponding SKUs, and all existing versions of each SKU from specified publishers. The publishers are categorized into Windows and Linux types. By default, the Windows publishers include `MicrosoftWindowsServer` and `MicrosoftWindowsDesktop`, while the Linux publishers are `Canonical`, `RedHat`, and `SUSE`. These defaults can be customized by setting the environment variables `LINUX_PUBLISHERS` and `WINDOWS_PUBLISHERS`.

The gathered information is formatted into a JSON structure and saved in MinIO. This setup ensures that any function requiring information about Azure VM types can access it quickly and efficiently from MinIO.

##### **Implementation Details**

The Celery task is scheduled to run periodically to fetch the latest VM information from Azure. Additionally, it is triggered when there is no existing Azure information in MinIO and a request is received. Here's a high-level overview of the process:

1. **Fetch Offers:** The task fetches all current offers available from the specified publishers.

- 
2. **Retrieve SKUs and Versions:** For each offer, it retrieves the corresponding SKUs and all available versions of each SKU.
  3. **Categorize by Publisher Type:** The offers are categorized into Windows and Linux based on the publisher.
  4. **Save to MinIO:** The collected data is serialized into JSON and stored in MinIO, making it accessible for other backend functions that need this information.

This periodic update ensures that the frontend always has access to the most current information about Azure VM offerings, facilitating accurate and up-to-date resource selection.

#### 7.2.4.2 Lab

The Lab module manages the configurations and statuses of various labs within the system. This module interacts with MinIO to store, retrieve, and update lab data, ensuring that all lab configurations are persistently stored and accessible as needed.

#### Functionalities

1. **Saving New Labs** When a new lab configuration is created, it is stored in MinIO. This ensures that the configuration is saved persistently and can be accessed or modified later. The new lab is initially given a status of "saved".
2. **Overwriting Existing Labs** The module allows for the updating of existing lab configurations. When an update is made, the existing configuration in MinIO is overwritten with the new data. This functionality ensures that lab configurations can be modified and kept up to date.
3. **Retrieving Lab Configurations** Users can retrieve the configuration of a specific lab. This functionality fetches the lab data from MinIO and returns it to the user, allowing them to view the current settings and details of the lab.
4. **Listing All Labs** The module can list all lab configurations stored in MinIO, along with their statuses. This provides a comprehensive overview of all the labs in the system, including their current states.
5. **Lab Status Management** Each lab has a status that indicates its current state. The possible statuses are:
  - **Saved:** The lab configuration is saved but not yet deployed.
  - **Deploying:** The lab is in the process of being deployed.
  - **Deployed:** The lab has been successfully deployed and is operational.



- 
- **Failed:** The deployment of the lab has failed.

## Detailed Process Flow

### 1. Creating a New Lab

- A new lab configuration is submitted and validated.
- The validated configuration is saved to MinIO with a status of "saved".
- The lab configuration can include various details such as resources, settings, and deployment parameters.

### 2. Updating a Lab

- An existing lab configuration is fetched from MinIO.
- The configuration is updated with new data.
- The updated configuration is saved back to MinIO, overwriting the previous version.

### 3. Fetching a Lab Configuration

- The lab's UUID is used to locate and retrieve the configuration from MinIO.
- The retrieved configuration is returned to the user, providing all the details of the lab.

### 4. Listing Labs

- The system queries MinIO to retrieve a list of all stored lab configurations.
- Each lab's status is also fetched and included in the list.
- The list of labs, along with their statuses, is returned to the user.

### 5. Checking Lab Status

- The system checks the current status of a lab using its UUID.
- The status is determined based on the deployment process and any updates made to the lab.
- The status is returned to the user, indicating whether the lab is saved, deploying, deployed, or failed.

By providing these core functionalities, the Lab module ensures efficient management of lab configurations and statuses, supporting the overall operation and maintenance of labs within the system.

---

### 7.2.4.3 Parser

The parser module is a critical component of the backend, responsible for transforming the JSON data structures shared between the frontend and backend into the format required by Terraform. This transformation is necessary because a single data structure cannot be used directly by both the frontend and Terraform due to their differing requirements.

#### Functionalities

1. **JSON Transformation** The parser module converts JSON data received from the frontend into a structure that Terraform can use. This involves mapping and transforming the data to ensure compatibility with Terraform's configuration requirements.
2. **Handling Complex Configurations** Some Terraform configurations require fixed structures that are challenging to represent dynamically in the frontend. The parser module addresses this by handling these complex configurations and ensuring they are correctly formatted for Terraform.
3. **Validation and Error Handling** During the transformation process, the parser module validates the data to ensure it meets the required specifications for Terraform. If any errors are encountered, they are logged and appropriate error messages are returned to guide users in correcting the issues.
4. **Integration with Terraform** The transformed JSON data is used to generate the necessary Terraform configuration files. These files are then utilized in the deployment process to provision the required infrastructure.

#### Detailed Process Flow

1. **Receiving JSON from Frontend** The parser module receives JSON data from the frontend, which contains the configuration details for the lab or deployment.
2. **Transforming Data Structure** The received JSON data is parsed and transformed into a format that Terraform can understand. This involves:
  - Mapping frontend data fields to corresponding Terraform configuration fields.
  - Converting dynamic structures into the fixed formats required by Terraform.
  - Ensuring all necessary configuration parameters are included and correctly formatted.
3. **Validating Transformed Data** The transformed data is validated to ensure it adheres to Terraform's configuration requirements. This step checks for:

- 
- Missing or incorrect fields.
  - Data types and value ranges.
  - Structural integrity and consistency.
4. **Handling Errors** If any validation errors are detected, the parser module logs the errors and returns detailed error messages to the frontend. This allows users to understand what went wrong and make the necessary corrections.
  5. **Generating Terraform Configuration Files** Once the data is successfully transformed and validated, the parser module generates the required Terraform configuration files. These files include:
    - Main configuration files defining the infrastructure.
    - Variable files specifying the parameters and values.
    - Any additional configuration files needed for specific resources.
  6. **Storing and Using Configuration Files** The generated Terraform configuration files are stored and made available for the deployment process. These files are then used by Terraform to provision the infrastructure as specified in the configurations.

By handling the complex task of transforming and validating JSON data, the parser module ensures that the backend can effectively use the configurations provided by the frontend. This seamless integration between the frontend and Terraform is essential for accurate and efficient infrastructure provisioning.

#### **7.2.4.4 Deployment**

The deployment module is responsible for handling the entire lifecycle of lab deployments. It manages tasks such as setting up necessary files and configurations, executing Terraform commands to deploy resources, and cleaning up resources once the deployment is completed. This module ensures that the deployment process is efficient, reliable, and maintains the integrity of the deployed resources.

This module utilizes Celery for asynchronous task management, enabling it to handle multiple deployment tasks concurrently. This setup ensures that the deployment process does not block other operations in the backend and can scale with demand. Temporary directories are created for each deployment to isolate the environment and avoid conflicts between different deployments. It also enables the user to download the `dockerfiles.tar.gz` and to check the deployment status.

---

## Task Chain Overview

The task chain is designed to manage the deployment lifecycle of a lab, ensuring that resources are provisioned, managed, and cleaned up correctly. It includes tasks for renaming the lab status in MinIO, running Terraform commands, and handling cleanup operations. Additionally, it includes error handling to manage failures gracefully.

```
1 task_chain = chain(  
2     rename_lab_in_bucket.si(AWS_STORAGE_BUCKET_NAME, lab_uuid, "deploying"),  
3     cleanup_task,  
4     init_task,  
5     apply_task,  
6     remove_resources_task,  
7     destroy_task,  
8     rename_lab_in_bucket.si(AWS_STORAGE_BUCKET_NAME, lab_uuid, "deployed"),  
9 ).on_error(  
10     chain(  
11         rename_lab_in_bucket.si(AWS_STORAGE_BUCKET_NAME, lab_uuid, "failed"),  
12         force_destroy_task,  
13     ),  
14 )
```

LISTING 7.5 Celery Task Chain

## Detailed Explanation of the Task Chain

The task chain is constructed using Celery's `chain` and `on_error` methods to sequence tasks and define error handling procedures. Here is a step-by-step breakdown of the task chain:

1. **Rename Lab to "deploying"**: The first task renames the lab status in MinIO to "deploying". This indicates that the deployment process has started and resources are being provisioned.
2. **Cleanup Task**: This task runs a cleanup operation to ensure that any residual resources from previous deployments with the same lab ID are removed. This helps to avoid conflicts and ensures a clean environment for the new deployment.
3. **Terraform Initialization**: The task chain runs `terraform init` to initialize the Terraform configuration. This step sets up the necessary Terraform files and prepares the environment for applying the configuration.
4. **Terraform Apply**: The next task in the chain is `terraform apply`, which provisions the cloud resources as defined in the Terraform configuration. This step creates the infrastructure required for the lab.

- 
5. **Resource Cleanup:** After the resources are successfully provisioned, the task chain runs a cleanup task to remove unnecessary Terraform state files. This ensures that the subsequent terraform destroy command does not remove essential snapshots but cleans up other resources.
  6. **Terraform Destruction:** The task chain includes a terraform destroy command to clean up the resources that are no longer needed. This step helps in managing resource lifecycle and cost.
  7. **Rename Lab to "deployed":** If all the previous tasks are successful, the final task renames the lab status in MinIO to "deployed". This indicates that the lab has been successfully provisioned and is ready for use.

### Error Handling

In case any task fails during the deployment process, the task chain defines an error handling procedure using `on_error`:

1. **Rename Lab to "failed":** If an error occurs, the lab status in MinIO is renamed to "failed". This indicates that the deployment process encountered an issue and was not completed successfully.
2. **Force Destroy Task:** The error handling chain includes a force destroy task that uses the Azure command-line tool to delete all resources associated with the lab. This ensures that no residual resources are left behind, which could incur unnecessary costs or cause conflicts in future deployments.

By structuring the deployment process in this way, the task chain ensures a robust and reliable deployment process that handles both success and failure scenarios effectively.

### Deployment Status

The deployment status can be checked during the deployment process and for a specified period after deployment, defined by the environment variable `REMOVE_DEPLOYMENTS_OLDER_THAN_HOURS`, which defaults to 168 hours. This status check helps in monitoring the progress and outcome of the deployment. With this real-time logs can be queried to provide insights into the deployment process.

### Deployment Download

During the deployment process and for a defined period afterwards (set by the environment variable `REMOVE_DEPLOYMENTS_OLDER_THAN_HOURS`), the deployment can be downloaded as a `dockerfiles.tar.gz` file. This tarball contains all necessary files and configurations, which can be easily imported into the HL resource editor without any modifications.

---

#### 7.2.4.5 Authentication

The authentication module secures access to the backend services by validating JWT and ensuring that users possess the required roles to access specific functionalities. It acts as a middleware in Django, intercepting requests to verify tokens and user roles before allowing access to protected resources.

The authentication mechanism is implemented using a custom Django authentication class that extends `BaseAuthentication` from the Django REST framework. It leverages the `authlib` library to handle JWT validation and role checking.

#### Detailed Explanation

The authentication process involves several key steps:

1. **Token Extraction:** The authentication middleware extracts the JWT from the Authorization header of the incoming request. If the header is missing or the token is improperly formatted, an `AuthenticationFailed` exception is raised with an appropriate message.
2. **JWT Validation:** The extracted token is validated against a set of JWK fetched from a specified URI (`JWKS_URI`), which can be defined using environment variables. The token's claims are decoded and validated to ensure the token's integrity and authenticity. If the validation fails, an `AuthenticationFailed` exception is raised.
3. **Role Verification:** Once the token is validated, the claims are checked to verify that the user has the required role (`EDITOR_ROLE`). This ensures that only users with the appropriate permissions can access the requested resources. If the user does not possess the required role, an `AuthenticationFailed` exception is raised.
4. **User Authentication:** If the token is valid and the user has the required role, the user is authenticated, and the claims are attached to the request object. This allows subsequent middleware and view functions to access user information and proceed with the request.

---

## 7.3 Deployment with Terraform

This section covers the deployment of resources required for the lab. It encompasses a general part which includes the structure and development configuration, along with parsing techniques. Further more the specific deployments are further described. The deployment is realized based on the defined lab architecture in 4.3 as well as on the corresponding ADRs 4.7. Corresponding prototypes were implemented to evaluate and define the ADRs. All Files referenced in the following sections can be found in the deployment repository Dynamic Pentest Lab Framework/deployment attached in the appendix.

### 7.3.1 Terraform Structure

The structure of the Terraform deployment can be observed in Listing 7.6. In addition to the default file structure recommended by Terraform-Best-Practises [75], it has been further organized into four parts. This four parts are grouped as following:

**Part 00:** This section encompasses all the essential resources needed for the foundational infrastructure, including setting up basic elements like virtual networks, DNS resolution, and deploying VPN access to the lab environment.

**Part 01:** This segment covers the setup of network and traffic restrictions established by the lab-creator within the generator tool.

**Part 02:** This part entails the configuration of containers and VM as specified by the lab-creator in the generator. Additionally, it manages the creation of snapshots for the VM.

**Part 03:** This section involves any special customizations implemented by the lab-creator, such as unique installations through repositories or post-deployment scripts.

```
1 00_infra.tf
2 00_infra_aks.tf
3 00_infra_container_dockovpn.tf
4 00_infra_dns.tf
5 00_infra_networking_subnet.tf
6 00_infra_security_networkRestriction.tf
7
8 01_networking_subnet.tf
9 01_security_networkRestriction.tf
10
11 02_service_container.tf
12 02_service_vm.tf
13 02_service_vm_snapshot.tf
14
15 03_service_customization_vm_customInstall.tf
16 03_service_customization_vm_postInstall.tf
```

```
17
18 data.tf
19 outputs.tf
20 provider.tf
21 tags.tf
22 terraform.tfvars.json
23 vars.tf
```

LISTING 7.6 Terraform Structure

### 7.3.2 Templating Methodologies

Terraform manages the entire parsing and templating process. Various expressions like loops, conditionals, and dynamic blocks are employed to create a comprehensive templating structure with the capability to incorporate specific logic. The part coming from the generator is the variable definition in the `terraform.tfvars.json` file. This is the central where all resources are listed and as well provides configuration of the lab itself.

#### Deployment Mode

It is important to distinguish whether the lab is deployed from the generator through a lab-creator or deployed with the deployment manager from a student. This differentiation is crucial because certain resources, such as Snapshots, may not need to be deployed in both scenarios. To address this, the variable `deployment_mode` was introduced, which is either set to `dev` when deploying through the generator or `lab` when deployed from the deployment manager.

#### Variable Definition

The associated lab is solely deployed according to the variable definition provided in `terraform.tfvars.json` by the backend, a process that necessitates the formulation of suitable data structures. An exemplary instance is the definition of the VM, as indicated in Listing 7.7. This definition is comprised of a map of objects, allowing iteration and therefore dynamic generation of resources.

```
1 variable "vm" {
2   type = map(object({
3     custom_label      = string
4     expose_to_output  = bool
5     os_type           = string
6     size              = string
7     admin_username    = string
8     admin_password    = string
9     admin_is_visible  = bool
10    publisher         = string
11    offer              = string
12    sku                = string
13    version            = string
```



```

14     os_disk_size_gb      = number
15     subnet              = string
16     dpl_install_settings = object({
17         repository_name    = string
18         repository_url     = string
19         repository_branch  = string
20         authorization_token = string
21     })
22     credentials          = list(object({
23         username          = string
24         password          = string
25         is_visible       = bool
26         type              = string
27     }))
28     )))
29     default = {}
30 }

```

LISTING 7.7 Virtual machine definition

In addition to defining resources, configurations for other components are also provided, such as the base settings, which include elements like the `deployment_mode`, or DNS-related configurations. These are organized and grouped by corresponding properties.

```

1  variable "generator_settings" {
2      type = object({
3          deployment_mode = string
4          snapshot_version = string
5          lab_id          = string
6      })
7  }
8
9  variable "dns_settings" {
10     type = object({
11         domain            = string
12         cloudflare_api_token = string
13     })
14 }

```

LISTING 7.8 Configuration Definition

---

## Loops

Loops were employed to dynamically generate resources based on variable definitions. This technique is exemplified in the dynamic creation of VMs, as shown in Listing 7.9. It makes use of the variable definition data structure introduced in Listing 7.7.

```
1 resource "azurerm_virtual_machine" "dpl_vm" {
2   for_each = var.vm
3
4   name           = each.key
5   resource_group_name = azurerm_resource_group.dpl_rg.name
6   location       = var.base_settings.location
7   ...
8 }
```

LISTING 7.9 Dynamic Generation utilizing Loops

## Conditionals

The variable definition supplied, along with additional configuration parameters, set the foundation for a sophisticated and dynamic deployment logic to be implemented. For this different conditional where introduced.

In Listing 7.10, a conditional statement is used, as an example, to initiate snapshots solely in lab mode. This condition ensures the iteration and subsequent creation of the `azurerm_snapshot` resource only when operating in lab mode.

```
1 resource "azurerm_snapshot" "dpl_ss_osdisk" {
2   for_each = {
3     for key, value in var.vm :
4       key => value if var.generator_settings.deployment_mode == "dev"
5   }
6   ...
7 }
8 }
```

LISTING 7.10 Conditional for each

Listing 7.11 illustrates another type of conditional, utilizing a dynamic block configuration. The full block is applied based on the conditional `var.generator_settings.deployment_mode == "lab"`. This is particularly used to ensure the disk is created from the snapshot when in lab mode.

```
1 resource "azurerm_virtual_machine" "dpl_vm" {
2   ...
3   dynamic "storage_os_disk" {
4     for_each = var.generator_settings.deployment_mode == "lab" ? [1] : []
5
6     content {
7       name = "dpl-d-os-${each.key}"
8       caching = "ReadWrite"
9       create_option = "Attach"

```

```

10     managed_disk_type = "Standard_LRS"
11     managed_disk_id   = azurerm_managed_disk.dpl_md[each.key].id
12     os_type           = each.value["os_type"]
13     disk_size_gb     = each.value["os_disk_size_gb"]
14   }
15 }
16 ...
17 }

```

LISTING 7.11 Dynamic Block

### 7.3.3 Data Management

The file `data.tf` establishes multiple datasets and data manipulation structures through the utilization of Terraform `terraform_data` resources. These data resources are integral to our infrastructure as code practices, as they allow us to fetch and compute data dynamically during the Terraform run. This approach is particularly beneficial in our deployment processes, providing necessary support structures to achieve our objectives alongside templating methodologies. The definition highlights the principal components without encompassing all the supporting data structures employed during deployment. These are subsequently detailed in the sections where they are used.

#### 7.3.3.1 Dynamic Credentials

The dataset of credentials, illustrated in Listing 7.12, incorporates crucial functionalities for the generation of dynamic passwords. It retrieves all the credentials defined in any VM or container deployment and produces random passwords using the `uuid()` function if the type is set to `dynamic`. The reason for executing the randomization process within the Terraform framework is crucial to understand. The deployment manager solely runs this terraform with the `terraform.tfvars` definition that the generator creates. Given that these dynamic passwords must be unique for each lab user, they need to be generated at the deployment's runtime and cannot be predefined; otherwise, they would be identical for all users.

```

1 resource "terraform_data" "dataset_credentials" {
2   input = { for key, resource in merge(var.vm, var.aks_deployments) :
3     key => [ for credential in resource.credentials :
4       {
5         username   = credential.username
6         password   = credential.type == "static" ? credential.password : uuid()
7         is_visible = credential.is_visible
8         type       = credential.type
9       }
10  ]

```

```
11 }
12 }
```

LISTING 7.12 Creation of dynamic Credentials

### 7.3.4 Resource Information

The importance of the resource dataset outlined in Listing 7.13 lies in providing essential information to the deployment manager, and consequently, to the lab user. Upon deployment of the VMs and containers, the Internet Protocol (IP) address information is retrieved. Consequently, this dataset is formulated, which, along with the IP address, constructs the DNS name based on the naming convention, and retrieves the dynamic credential dataset, previously discussed, for the specific resource. It is important to understand that only resources with the `expose_to_output` property set to true are incorporated into this dataset. Furthermore also only the credentials are included which have the `is_visible` property set to true. This approach ensures that only the information about the resources, as defined by the lab-creator, is provided to the student.

```
1 resource "terraform_data" "dataset_resources_connection_information" {
2   input = { for resource in concat(terraform_data.vms.output, terraform_data.pods.
3     ↪ output) :
4     resource.name => {
5       ip      = resource.ip
6       subnet  = resource.subnet
7       user    = can(resource.user) ? resource.user : "-"
8       password = can(resource.password) ? resource.password : "-"
9       custom_label = can(resource.custom_label) ? resource.custom_label : "-"
10      dns_name = "${resource.name}.${random_uuid.resource_group_uuid.result}.${var
11        ↪ .dns_settings.domain}"
12      credentials = can(terraform_data.dataset_credentials_to_expose.output["${
13        ↪ resource.name}"]) ? terraform_data.dataset_credentials_to_expose.
14        ↪ output["${resource.name}"] : null
15    } if resource.expose_to_output == true
16  }
17 }
```

LISTING 7.13 Resource Information Dataset

---

## 7.3.5 Baseline Deployment

### 7.3.5.1 Provider Configuration

For the deployment, three providers are utilized: the `azurerm` for the Azure deployments, the `kubernetes` provider for managing AKS resource deployments, and the `cloudflare` provider for managing the root domain DNS. These are outlined in the `File provider.tf`. A dynamic configuration of these providers is achieved based on the variables defined in the `terraform.tfvars.json` File, thereby allowing for a dynamic customization of the provider configuration. The `kubernetes` provider, as shown in Listing 7.14, retrieves information directly from the deployed AKS Cluster for self-configuration.

```
1 provider "kubernetes" {
2   host                = azurerm_kubernetes_cluster.dpl_aks.kube_config[0].host
3   username            = azurerm_kubernetes_cluster.dpl_aks.kube_config[0].
    ↪ username
4   password            = azurerm_kubernetes_cluster.dpl_aks.kube_config[0].
    ↪ password
5   client_certificate  = base64decode(azurerm_kubernetes_cluster.dpl_aks.
    ↪ kube_config[0].client_certificate)
6   client_key          = base64decode(azurerm_kubernetes_cluster.dpl_aks.
    ↪ kube_config[0].client_key)
7   cluster_ca_certificate = base64decode(azurerm_kubernetes_cluster.dpl_aks.
    ↪ kube_config[0].cluster_ca_certificate)
8 }
```

LISTING 7.14 Kubernetes Provider

### 7.3.5.2 Base Infrastructure Deployment

The deployment of the basic infrastructure resources required for the foundation of the lab, defined in the file `00_infra.tf`, includes the deployment of a Virtual Network and a Resource Group where all resources are grouped into. The Virtual Network enables Azure resources to communicate with each other and provides a unique isolated network for the lab. The resource group name includes a randomly generated UUID since it needs to be unique within the Azure Subscription. This is achieved through utilizing the `random_uuid` Terraform resource. As seen in Listing 7.15, when deploying in dev mode, the resource group gets named by the provided UUID generated from the generator in order to map the deployment into the specific lab in the generator.

```
1 resource "azurerm_resource_group" "dpl_rg" {
2   name                = var.generator_settings.deployment_mode == "dev" ? "dpl-rg-
    ↪ ${var.generator_settings.lab_id}-${var.generator_settings.
    ↪ snapshot_version}" : "dpl-rg-${random_uuid.resource_group_uuid.result}"
3   location            = var.base_settings.location
```

```

4   tags
5     = local.tags
6   }

```

LISTING 7.15 Resource Group Name

### 7.3.5.3 Kubernetes Cluster Deployment

Following the deployment of the base infrastructure, a AKS, as outlined in the file `00_infra_aks.tf`, is deployed to facilitate the deployment of containers. A detailed evaluation of this container deployment solution is provided in ADR [4.7.10](#). The AKS cluster plays a crucial role in the foundational infrastructure as it also hosts the container for VPN access.

A fundamental aspect of the cluster configuration is the employed network plugin. The primary requirement for this plugin is to ensure bidirectional communication between the VM and the pod. Therefore, the Azure network plugin was selected [\[76\]](#). Additionally, the Azure network plugin offers compatibility with Windows-based nodes, facilitating potential future implementations without any major issues.

As indicated in the architectural design, a separate Node is deployed for each container network. This setup is achieved through the dynamic definition of the node pools, as illustrated in [Listing 7.16](#). As observed in the conditional statement within the `for_each` definition, a node pool is deployed for each subnet identified as a Kubernetes subnet. The naming convention corresponds to the subnet identifier. Additionally, the `pod_subnet_id` dynamically gets assigned to the specific subnet, indicating that the containers operate within this subnet. A subnet label is also appended to the `node_labels`. This is required for the deployment and assignment of containers, designed to run in that specific subnet, to this particular node pool. This operation is facilitated by node selectors [\[77\]](#), a concept further expounded in the container deployment [Section 7.3.9](#). Moreover, the `enable_auto_scaling` feature was activated, offering automatic scaling and hence ensuring scalability for the dynamic workload in terms of quantity and load of the containers to be run in a lab. The scaling was capped at three nodes per subnet, which appeared to be an appropriate initial value to handle the average load and prevent excessive costs. This value can be modified according to requirements.

```

1 resource "azurerm_kubernetes_cluster_node_pool" "dpl_akcnp" {
2   for_each = {
3     for key, value in var.subnets :
4       key => value if value.kubernetes == true
5   }
6 }
7
8 name = "akcnp${split("-", "${each.key}")[2]}"
9 ...

```

```

10 pod_subnet_id      = azurerm_subnet.dpl_s[each.key].id
11 node_labels       = { subnet = "${each.key}" }
12 enable_auto_scaling = true
13 min_count         = 1
14 max_count         = 3
15
16 tags = local.tags
17 }

```

**LISTING 7.16** Dynamic Creation of Node Pools

### 7.3.5.4 Networking and Security Restriction

To provide connectivity for the base infrastructure, mainly the AKS Cluster, infrastructure subnets for the AKS nodes and infrastructure pods are deployed. These are defined in the file `00_infra_networking_subnet.tf`. To enhance the security and limit the accessibility to these infrastructure subnets within the lab, customized security rules have been outlined in the file `00_infra_security_networkRestriction.tf`. These rules minimize communication to the bare essentials and only permit interactions that are crucial for the lab's operation. The allowed rules specify communication from the internet to the dockovpn container to enable the VPN connection for the students and DNS resolution to the internal DNS service of Kubernetes. Any other connections to the node subnet and infra pod subnet are strictly forbidden. Through this approach, potential manipulation or attempts to do so are prevented.

### 7.3.5.5 VPN Access

During the development and evaluation of prototypes, the VPN access technology was defined, as outlined in ADR 4.7.13. As a result, the decision was made to implement the dockovpn container to enable VPN access for the students.

The deployment of the VPN access is outlined in File `00_infra_container_dockovpn.tf`. It encompasses a service definition for the dockovpn pod, which is defined as type load balancer and therefore automatically gets exposed to the internet via the AKS managed load balancer. Further more persistent storage was established through the use of a Azure Storage Share. The deployment of persistent storage involves multiple parts, including setting up a Storage Account where an Azure Storage Share is created, defining a Kubernetes persistent volume that references this specific Storage Share, and ultimately, establishing a Volume Claim that supplies the container with storage. Although persistent storage would not have been strictly necessary for the lab under normal circumstances, as the container is not typically removed, it is crucial for the dynamic retrieval of the generated OpenVPN profile, which gets created during start of the container. Utilizing the storage share to access the generated profile via Azure storage API, was deemed the most effective approach. However, other strategies were

also considered. One such approach involved opening and exposing the SSH port during the deployment process to retrieve the generated profile. But this approach would have introduced additional complexity, such as needing to manage exposure solely during the deployment phase, making adjustments to the container image to support the ssh access, and addressing potential security concerns when exposing the port. An alternative approach considered was employing the container's built-in option to expose the profile on port 8080. However, this method also raised security issues and necessitated additional restrictions to ensure only the deployment could retrieve the profile on the port. Moreover, the design only allowed for a single download of the profile, which posed a problem as the load balancer's health checks on that endpoint would prevent further downloads. Once again, modifications to the container image would have been required to resolve this issue.

The deployment of Dockovpn was set up in accordance with the available configuration options described in the Dockovpn documentation [78]. The required options for the project's use-case, was the definition of the environmental variable `HOST_ADDR`. This variable defines the VPNs external address, which is configured in the OpenVPN profile. The address is equivalent to the public IP of dockovpn's service definition, and is dynamically retrieved as demonstrated in Listing 7.17. Subsequently the `depends_on` was defined to ensure the load balancer service is established and the external IP address is known before deploying the Dockovpn container.

```
1 resource "kubernetes_deployment" "dpl_kd_dockovpn" {
2   ...
3   env {
4     name      = "HOST_ADDR"
5     value     = data.kubernetes_resource.data_dpl_ks_dockovpn_ip.object.status.
               ↪ loadBalancer.ingress[0].ip
6   }
7   ...
8   depends_on = [ data.kubernetes_resource.data_dpl_ks_dockovpn_ip ]
9 }
```

LISTING 7.17 Dockovpn Deployment ENV

Moreover, the container is required to function within a security context that holds `NET_ADMIN` capabilities. Although this prerequisite is not detailed in Dockovpn's documentation, it's inferred that, due to the inherent characteristics of a VPN server, it must be able to perform routing adjustments in the routing table to work properly. Upon examination of Dockovpn's startup script `scripts/start.sh`, it becomes apparent that the `iptables` are configured to permit forwarding traffic from the VPN. Also, a `post_start` command was defined, which executes the `openvpn` generation script `./genclient.sh n ovpnprofile`. The script generates a VPN profile named `ovpnprofile`. While an OpenVPN profile is also created by default, it is assigned a random name, making it impractical for later dynamic retrieval of the OpenVPN profile due to the unpredictability of the name.



---

The basic server configuration is limited to a few environmental variables by default. Given that the standard setup of the OpenVPN server initiates a Full Tunnel, thus directing all client traffic through this tunnel, a solution has been implemented using `kubernetes_config_map` to inject a custom configuration to the `dockovpn` container. With the config map, it's possible to override the default `server.conf` and `client.ovpn`, thereby allowing settings for routing behavior to be configured. The route installed is dynamically configured, depending on the `vnet_address_space` selected for the lab. This modification permits the customization of the OpenVPN server configuration to facilitate split tunneling, which results in only the traffic intended for the lab being routed through the tunnel. This also provides flexibility for customization in the future.

### 7.3.5.6 DNS Resolution

To facilitate DNS resolution for the lab the decision was made in ADR 4.7.16 to use Cloudflare for the management of the root domain and Azure Public VPN Zone for the subdomains associated with individual labs. For this project the domain `dynpentestlab.ch` was registered with Cloudflare. The deployment of the VPN infrastructure is outlined in File `00_infra_dns.tf`. This deployment includes the establishment of nameserver records for the root domain, outlined in Listing 7.18, which are directed towards the nameservers of the Azure subdomain zone. The resource definition has been designed to dynamically configure all four nameservers. However, this can be tailored as per requirements should scalability become an issue due to the limitation of the 1000 free records at Cloudflare, as further elaborated in ADR 4.7.16.

```
1 resource "cloudflare_record" "dpl_dns_entry_ns" {
2   ...
3   count = 4 #add 4 NS servers
4   value  = element(to_list(azurerms_dns_zone.dpl_dns.name_servers), count.index)
5   type   = "NS"
6   depends_on = [ azurerms_dns_zone.dpl_dns ]
7 }
```

LISTING 7.18 Root DNS Zone NS Records

Furthermore, it includes the generation of the lab's subdomain, along with the establishment of VPN records for all resources associated with the lab, as delineated in Listing 7.19. The IP address information for each resource is obtained from the central dataset `dataset_resource_information_internal`, offering a straightforward method to access the necessary information. The dataset is further described in .... This procedure guarantees that all resources receive a VPN record dynamically.

```
1 resource "azurerms_dns_a_record" "dpl_dns_record" {
2   for_each = merge(var.vm, var.aks_deployments)
3   ...
```

```

4 records          = ["${terraform_data.dataset_resource_information_internal.
    ↪ output["${each.key}"].ip}"]
5 depends_on = [ terraform_data.dataset_resource_information_internal ]
6 }

```

LISTING 7.19 Creation of DNS A-Records

Through the implemented design, which encompasses the core dataset and its application in generating DNS records, the replacement of DNS providers can be easily facilitated, should the need or desire for alternative providers arise.

### 7.3.6 Virtual Machine Deployment

The deployment for VMs is outlined in File `02_service_vm.tf`. It handles the creation of Windows as well as Linux based VMs. To achieve this, dynamic block configurations and conditional expressions as described in the section template methodologies 7.3.2 were utilized.

For the deployment of VMs the Terraform resource `azurerm_virtual_machine` was used. The Terraform resource `azurerm_virtual_machine` was selected for the deployment of the VM. This selection was made over `azurerm_linux_virtual_machine` and `azurerm_windows_virtual_machine` resources, despite the impending deprecation of `azurerm_virtual_machine` in the 3.x releases.

The key reason for this choice was the need to incorporate the snapshot technology as detailed in Section 7.3.7. A significant issue arises because the new resource definitions do not permit the attachment of existing OS disks, an essential aspect of the snapshot technology. This issue, already reported, remains unresolved [79].

According to the official `azurerm` resource documentation [80], a note suggests using the older resource definition `azurerm_virtual_machine` or to use images instead. However, the image approach presents a challenge.

It only supports creating the OS disk from generalized images, which would result in the loss of computer-specific information such as the computer name and Client Machine ID (CMID) [81]. This becomes problematic in custom installations, for instance, an Active Directory installation. This issue is also reported [82] and remains unresolved.

Generally speaking, conflicts often emerge between the old and new resource definitions, primarily because the newer version lacks certain features present in the older one. Conversely, the old resource no longer receives updates or and also lacks certain features. The root cause of these issues appears to be inherent limitations in the existing Terraform protocol itself, as noted in [83]. It is hoped that these problems will can be solved with the integration of Terraform Protocol v6.

An additional issue arose requiring the implementation of a workaround, which utilized a `local-exec` provisioner as illustrated in Listing 7.20. This was to solve a bug

---

encountered when applying the `azurerms_virtual_machine_run_command` immediately after the deployment of the VM. This particular command, used in section 7.3.8 for custom installations, tends to cause complications when applying directly following the VM's deployment. This provisioner pauses for 20 seconds prior to proceeding, which effectively resolves the issue. This well-documented bug [84] remains unresolved but seems to only occur with specific agent installations on Linux based distributions. The goal was to ensure seamless support for a broad spectrum of images without needing to create multiple fixes for different images. Despite its unconventional nature, this approach appeared to be the most viable solution under the circumstances. Had it been possible to utilize the `azurerms_linux_virtual_machine` resource, which allows the definition of the `provision_vm_agent` property, not only for provisioning but also for checking and ensuring the agent's active status, this issue could have been addressed without resorting to the workaround. This pertains to the problem that the existing resource `azurerms_virtual_machine` lacks the inclusion of more recent features.

```
1 resource "azurerms_virtual_machine" "dpl_vm" {
2   ...
3   provisioner "local-exec" {
4     command = "sleep 20"
5   }
6   ...
7 }
```

LISTING 7.20 Virtual Machine Sleep Provisioner

### 7.3.7 Virtual Machine Snapshot Technology

For the creation and utilization of snapshots the Deployment Mode detailed in Section 7.3.2 is integral. When deployment occurs in dev mode, a snapshot is generated as detailed in Listing 7.21. Each snapshot, including the VM's name, is appended with a timestamp, referred to as the `snapshot_version`. These snapshots are stored in a distinct snapshot resource group. The creation of a separate resource group is necessitated by the need to identify and maintain these snapshots, as they are used by other labs during their deployment processes. The `create_option` indicates that the disk specified in `source_uri` is being snapshotted. Additionally, the creation of the snapshot depends on the successful completion of the custom installation, as it should be encapsulated in the snapshot.

```
1 resource "azurerms_snapshot" "dpl_ss_osdisk" {
2   for_each = {
3     for key, value in var.vm :
4       key => value if var.generator_settings.deployment_mode == "dev"
5   }
6 }
```

```

7   name                = "dpl-ss-osdisk-${each.key}-${terraform_data.
      ↪ snapshot_version[0].output}"
8   location             = var.base_settings.location
9   resource_group_name = azurerm_resource_group.dpl_rg_snapshots[0].name
10
11  create_option        = "Copy"
12  source_uri           = data.azure_rm_managed_disk.dpl_md_data[each.key].id
13
14  depends_on = [ azurerm_virtual_machine_run_command.dpl_vme_repo_install ]
15 }

```

LISTING 7.21 Creation of virtual Machine Snapshot

During the deployment in lab mode, no snapshot is generated. Instead, OS disks are created based on the `azurerm_managed_disk` definition presented in Listing 7.22 that enables the usage of snapshots. The `create_option` specifies the copy instruction, indicating that the disk should be created or duplicated from that given snapshot. To provide the `source_resource_id` to the `azurerm_managed_disk` definition, the `dpl_ss_osdisk_data` definition is used, which contains all pertinent snapshot information.

```

1  "azurerm_snapshot" "dpl_ss_osdisk_data" {
2    for_each = {
3      for key, value in var.vm :
4        key => value if var.generator_settings.deployment_mode == "lab"
5    }
6
7    name                = "dpl-ss-osdisk-${each.key}-${var.generator_settings.
      ↪ snapshot_version}"
8    resource_group_name = "dpl-rg-snapshots-${var.generator_settings.lab_id}-${var.
      ↪ generator_settings.snapshot_version}"
9  }
10
11 resource "azurerm_managed_disk" "dpl_md" {
12   for_each = {
13     for key, value in var.vm :
14       key => value if var.generator_settings.deployment_mode == "lab"
15   }
16
17   name                = "dpl-d-os-${each.key}"
18   resource_group_name = azurerm_resource_group.dpl_rg.name
19   location             = var.base_settings.location
20   storage_account_type = "Standard_LRS"
21   create_option        = "Copy"
22   source_resource_id   = data.azure_rm_snapshot.dpl_ss_osdisk_data[each.key].id
23   os_type              = each.value["os_type"]
24 }

```

LISTING 7.22 Usage of virtual Machine Snapshots

---

## 7.3.8 Virtual Machine Custom Installations

### 7.3.8.1 Installation Repositories

As established in ADR 4.7.6, Github repositories are to be employed to facilitate custom installations for VMs. For this purpose, the example repositories `install-template-linux` and `install-template-windows` have been created for Linux and Windows. These provide a fundamental structural guide for custom installation repositories, along with examples of credential access, the use of multi flags, and appropriate error handling in line with the procedures laid out in Section 7.3.8.3.

Custom installation repositories consist of two essential scripts that serve as entry points. Although these two scripts are mandatory, the lab-creator has the liberty to define additional scripts and subsequently invoke them from the primary scripts.

**Install script:** This script facilitates the custom installation designed by the lab-creator. The example repository delineates a method to retrieve custom credentials, leveraging only built-in tools to ensure maximum compatibility without necessitating the installation of JSON parsers like `jq` on Linux. These credentials can then be potentially employed in the creator's custom installation to set up a login or include a user in the OS.

**Post-install script:** This script manages the custom logic to be executed after the VM snapshot is created or when lab is deployed by the student. The post-install script could potentially be used to distribute dynamic credentials or multi-flags. These elements are unique to each lab and should not be incorporated in the custom installation routine, as they would be captured in the snapshot.

### 7.3.8.2 Custom Installations

This section describes the resources defined in the Terraform deployment, which make use of the custom repositories and initiate the installation and post installation scripts, as described in the previous section. Furthermore some additional logic to distribute multi-flags and custom credentials is implemented.

The Terraform resource, `azurerm_virtual_machine_run_command` was utilized to execute the logic of using these repositories and trigger the custom installation. This resource the execution of commands on a specific VM. For every VM, a resource is dynamically created to manage the custom installation exclusive to that machine. Dynamic block configurations and conditional expressions are used to manage the commands for both Windows and Linux.

Conversely, an assessment was made of the `azurerm_virtual_machine_extension`, which similarly allows the execution of customScripts. However, the constraint of this resource is its restriction to a single assignment of a machine extension of the

---

customScript type per VM. Given the necessity of using two resources per VM to trigger both custom and post-installation scripts, this was not a feasible option.

The custom installation is facilitated through the definition detailed within File 03\_service\_customization\_vm\_customInstall.tf, in which following commands are executed:

**Distribution of Credentials:** To distribute the credentials defined for the particular VM, the corresponding credential definition for the VM is retrieved and written to a Textfile text file on the operating system. This can then be used by the lab-creator to further use these credentials in his custom installation script, as mentioned before.

**Download of installation repository:** Based on the repository information provided for the VM, curl is used together with the provided authorization token to download the repository. Curl was chosen over git clone since not each distribution comes with git preinstalled. With the aim to have a generalized approach which works for the most distributions this was the best solution.

**Extraction and Execution:** Following the download the repository archive gets extracted and the custom installation script install-script gets executed.

An example of the custom Installation procedure for Linux is outlined in Listing 7.23.

```
1 resource "azurerm_virtual_machine_run_command" "dpl_vme_repo_install" {
2   ...
3   dynamic "source" {
4     for_each = each.value["os_type"] == "Linux" ? [1] : []
5     content {
6       script = <<-EOT
7         sudo mkdir /dpl_tmp
8         echo '${base64encode(jsonencode(terraform_data.dataset_credentials.output[
9           ↪ each.key]))}' | base64 -d > '/dpl_tmp/credentials.txt'
10        curl -L -o '/dpl_tmp/${each.value.dpl_install_settings.repository_name}.
11          ↪ tar.gz' '${each.value.dpl_install_settings.repository_url}' -H '
12          ↪ Authorization: Bearer ${each.value.dpl_install_settings.
13          ↪ authorization_token}' -sS
14        tar -xzf '/dpl_tmp/${each.value.dpl_install_settings.repository_name}.tar.
15          ↪ gz' -C '/dpl_tmp/'
16        chmod +x '/dpl_tmp/${each.value.dpl_install_settings.repository_name}-${{
17          ↪ each.value.dpl_install_settings.repository_branch}/install-script.
18          ↪ sh'
19        sudo '/dpl_tmp/${each.value.dpl_install_settings.repository_name}-${{each.
20          ↪ value.dpl_install_settings.repository_branch}/install-script.sh'
21      EOT
22    }
23  }
24  }
25  ...
26 }
```

---

### LISTING 7.23 Custom installation Linux

The post-install script also gets triggered through the same Terraform resource and is further outlined in File `03_service_customization_vm_postInstall.tf`. The resource is handled in the same way as the custom installation resource. The resource definition executes the following commands:

**Distribution of Multiflags:** Analogous to the distribution of the credentials, the multiflags are written to a text file on the operating system. This flags can subsequently be employed as part of a post-snapshot routine to disseminate unique user flags, thereby finalizing the design for the CTF challenge.

**Execution of post installs script:** The `post-install-script` gets executed.

#### 7.3.8.3 Output and Debugging for Lab-Creator

In addition to the ability to test and develop the custom installation script on an identical distribution in advance, command execution provides installation feedback. This feedback is made available through the output, which can be easily accessed from the generator log. To facilitate this, a `terraform_data` resource, defined in the `data.tf` file, is employed to retrieve and parse the `azurerm_virtual_machine_run_command` output from all VMs.

As shown in Listing 7.24, the installation output contains two components: the output itself and the `error_message`. The output comprises all content written to the console, in this instance, the output originates from the example repository code, which prints various credentials. Conversely, the `error_message` captures any error messages generated by the script.

```
1 vm_custom_install_output = {
2   ...
3   "dpl-vm-abcd444" = {
4     "end_time" = "2024-06-02T17:08:03+00:00"
5     "error_message" = <<-EOT
6     /dpl_tmp/dpl-test-install-linux-main/install-script.sh: line 29: get-foo:
7       ↪ command not found
8
9     EOT
10    "output" = <<-EOT
11    Starting with installation script
12    b8ff5566-862b-b047-58c1-8159f8da2ef5
13    custompassword2
14    customuser1
15    customuser2
16    custompassword2
17    Finished with installation script
```

```

17
18     EOT
19     "start_time" = "2024-06-02T17:08:01+00:00"
20 }
21 }
22 ...
23 }

```

LISTING 7.24 Example Install Output

Outputting the `error_message` is crucial because even if some commands in the script are unsuccessful, as long as the final command is successful with an exit code of 0, the entire `azurerms_virtual_machine_run_command` is deemed successful. This necessitates knowing the actual `error_message` output to ascertain if the operations were executed as intended.

For instance, the `error_message` in listing 7.24 reveals that a specific command is nonexistent. However, it still culminates successfully because the final command (echoing "Finished with installation script") is successful and exits with code 0. This behavior applies for Windows as well.

In contrast, in Listing 7.25, an unsupported parameter causes `curl` to fail. Subsequent commands, which are reliant on the file archive that `curl` should download, also fail, resulting in the final command exiting with code 2. In such instances, the execution of the `azurerms_virtual_machine_run_command` is unsuccessful, causing the entire lab deployment to fail.

```

1 ...
2 Error: running the command: polling failed: the Azure API returned the following
3 Status: "VMExtensionProvisioningError"
4 Code: ""
5 Message: "VM has reported a failure when processing extension 'dpl-mrc-repo-
   ↳ install-dpl-vm-abcd222' (publisher 'Microsoft.CPlat.Core' and type '
   ↳ RunCommandHandlerLinux'). Error message: '{"executionState\":\"Failed
   ↳ \",\"executionMessage\":\"Execution failed: failed to execute command:
   ↳ command terminated with exit status=2\", \"output\": \"\", \"error\": \"curl:
   ↳ option --no-progress-meter: is unknown\\ncurl: try 'curl --help' or 'curl
   ↳ --manual' for more information\\n\", \"exitCode\":2, \"startTime
   ↳ \": \"2024-06-02T13:49:44Z\", \"endTime\": \"2024-06-02T13:49:44Z\"}'."
6 Activity Id: ""
7 ...
8
9 }

```

LISTING 7.25 Example failing Install Output

To ensure the entire deployment fails when an error arises, the utilization of `try-catch` should be considered for Windows deployments. As illustrated in Listing 7.26, the



---

execution of an unrecognized command is captured, and an exit code 1 is produced from the catch block, consequently causing the whole deployment to fail.

```
1 ...
2 try {
3   $foo = get-foo
4 }
5 catch{
6   write-error "An error occurred"
7   exit 1
8 }
9 ...
10 }
```

LISTING 7.26 Error handling Powershell

To ensure robust error handling in Linux Bash scripts, a similar approach to Windows PowerShell is implemented. The example in Listing 7.27 illustrates a Bash script configured to terminate on any error. The trap command specifies a function to execute when an error is detected. The catch function then logs an error message and exits with a status of 1, thereby ensuring the entire deployment process halts immediately upon encountering an issue.

```
1 ...
2 trap 'catch' ERR
3
4 catch() {
5   echo "An error occurred"
6   exit 1
7 }
8 ...
9 }
```

LISTING 7.27 Error handling Bash

Both of the error handling strategies are outlined in the example repositories `install-template-linux` and `install-template-windows`.

### 7.3.9 Container Deployment

This section describes container deployment utilizing AKS. Additionally, it explains how to use custom Images from private repositories, along with the distribution of flags and credentials.

---

### 7.3.9.1 Base Definition

The deployment of containers involves the creation of a deployment through the Terraform resource `kubernetes_deployment`, as detailed in File `02_service_container.tf`. For improved Resource Management and Organization, each deployment initiates the creation of a separate namespace. This approach enables Scoped Configurations such as `configMaps` and `secrets`, which are assigned to this namespace and only available within it. Network Policies could potentially be applied on a namespace scope, however, this is not utilized due to the adoption of a different strategy as described in Section 7.3.10.

Moreover, ports and environmental variables are dynamically populated and set based on the configuration set in the `terraform.tfvars` using dynamic block configurations. To ensure the minimum necessary resources (Random-Access Memory (RAM), CPU) for the containers, resource requests were employed to ascertain the corresponding values defined by the lab-creator.

In order to ensure that containers operate on the designated nodepool residing within the specified container subnet, a `node_selector` is assigned to the container, as illustrated in Listing 7.28. This node selector guarantees that the container only operates on the specified node, which is crucial for facilitating the network architecture in assigning the container to particular subnets.

```
1 resource "kubernetes_deployment" "dpl_kd" {
2   ...
3     spec {
4       node_selector = {
5         "kubernetes.io/os" = "linux"
6         "subnet"           = each.value.subnet
7       }
8     ...
9   }
```

LISTING 7.28 Deployment Node Selector

Furthermore a delay when spinning up the containers had to be introduced, using the `min_ready_seconds` parameter, which was set to 15 seconds. This adjustment was essential as occasionally, the data resource referred to in Section 7.3.4 would attempt to retrieve the container's IP Address as soon as they were deployed. In a few instances, despite the completion of the deployment, the pod IP for some containers remained unavailable. As a result, to enhance the stability of the lab deployment, this modification was instituted.

An alternative approach to avoid the `min_ready_seconds` configuration was assessed using the Terraform resource `kubernetes_manifest`. This resource allows the definition of deployments as Kubernetes YAML Ain't Markup Language (YAML) manifest and provides the configuration of wait blocks, as illustrated in Listing 7.29, which

necessitates the availability of certain fields in the correct format, utilising regex. However, due to a previously reported limitation [85], the `kubernetes_manifest` resource mandates API access during the Terraform plan operation. As the Kubernetes cluster and its API configuration are dynamically created during runtime, facilitating API access is currently unviable. This would necessitate two separate deployments: the initial deployment to establish the Kubernetes infrastructure, followed by a second deployment to launch the resources.

```
1 resource "kubernetes_manifest" "dpl_kd" {
2   for_each = var.aks_deployments
3   ...
4   wait {
5     fields = {
6       "status.podIP" = "^(\\d+(\\.|$)){4}"
7     }
8   }
9   ...
10 }
```

LISTING 7.29 Kubernetes Manifest wait Configuration

### 7.3.9.2 Distribution of Flags and Credentials

The method of distributing flags and credentials for containers is accomplished by setting them as environmental variables. For this to occur, custom data resources are defined to extract the multi-flag or credential data and convert it into a valid ENV variable. This process is illustrated by the example of the Multi-flags in Listing 7.30.

```
1 resource "terraform_data" "multiflags_as_env" {
2   input = { "MULTIFLAGS" : jsonencode(var.multiflags)}
3 }
```

LISTING 7.30 Multiflag env Parsing for Container

The custom data definitions are subsequently utilized in conjunction with standard environment variables during assignment to the container, as demonstrated in Listing 7.31.

```
1 resource "kubernetes_deployment" "dpl_kd" {
2   ...
3   dynamic "env" {
4     for_each = merge(each.value.env, terraform_data.multiflags_as_env.
5       ↪ output, terraform_data.credentials_as_env.output[each.key])
6     content {
7       name   = env.key
8       value  = env.value
9     }
10  }
```

```
10 ...
11 }
```

LISTING 7.31 Env Assignment Container

### 7.3.9.3 Custom Docker Images

To provide the possibility for custom images and pre-existing HL images from private repositories, the lab-creator has the capability to define its custom repository along with suitable authentication tokens. These enables the pulling of the corresponding images from either public or private repositories. If a custom repository is defined for a container deployment, a secret of the `dockerconfigjson` type is dynamically created to facilitate the authentication process for pulling from the corresponding repository, as detailed in Listing 7.32.

```
1 resource "kubernetes_secret" "dpl_ksec_dockerpull" {
2   ...
3   type = "kubernetes.io/dockerconfigjson"
4   data = {
5     ".dockerconfigjson" = jsonencode({
6       auths = {
7         "${each.value.image_pull_secrets.registry_url}" = {
8           username = "${each.value.image_pull_secrets.username}"
9           password = "${each.value.image_pull_secrets.authorization_token}"
10          auth = base64encode("${each.value.image_pull_secrets.username}:${each.
11             ↵ value.image_pull_secrets.authorization_token}")
12        }
13      }
14    })
15  }
16  ...
17 }
```

LISTING 7.32 Docker Pull Secrets

### 7.3.10 Network Restrictions

The emulation of intricate real-world networks via network firewalls, as delineated in ADR 4.7.9, has been implemented. This implementation is outlined in File `01_security_networkRestriction.tf`.

Since the application of these restrictions happens to the parent subnet, a unique security group for each defined subnet is dynamically generated. This group forms the foundation for the subsequent application of security rules.

Given that traffic within the same v-net is permitted by default, it is necessary to establish a rule for each subnet's security group to allow all inter-subnet traffic. This has been accomplished by constructing a rule that blocks all inbound communication and applying it to the security group of the subnets. As this rule also blocks all intra-subnet traffic, another rule is enforced to permit this. This forms the basis for the implementation of custom rule rules that allow certain forms of communication, an essential element presented in Listing 7.33.

```
1 resource "azurerm_network_security_rule" "dpl_nsr" {
2   for_each = var.network_security_rules
3
4   name                = "${each.key}"
5   priority            = each.value.priority
6   direction           = "Inbound"
7   access              = each.value.access
8   protocol            = each.value.protocol
9
10  source_port_range   = length(each.value.source_ports) <= 1 ? each.value.
11    ↪ source_ports[0] : null
12  source_port_ranges  = length(each.value.source_ports) > 1 ? each.value.
13    ↪ source_ports : null
14  source_address_prefix = terraform_data.
15    ↪ dataset_resource_information_internal.output[each.value.source].ip
16
17  destination_port_range = length(each.value.destination_ports) <= 1 ?
18    ↪ each.value.destination_ports[0] : null
19  destination_port_ranges = length(each.value.destination_ports) > 1 ?
20    ↪ each.value.destination_ports : null
21  destination_address_prefix = terraform_data.
22    ↪ dataset_resource_information_internal.output[each.value.destination].ip
23
24  resource_group_name      = azurerm_resource_group.dpl_rg.name
25  network_security_group_name = azurerm_network_security_group.dpl_nsg[
26    ↪ terraform_data.dataset_resource_information_internal.output[each.value.
27    ↪ destination].subnet].name
28 }

```

LISTING 7.33 Network Security Rules

The rule is always applied to the security group of the destination resource's subnet, as demonstrated by the dynamic assignment of the property `network_security_group_name`. This is reasoned by the necessity to manage the inbound traffic to the specific resource, as the default block is also processed in an inbound manner.

To facilitate the dynamic generation of rules, a Terraform data resource was used to establish the `dataset_resource_information_internal` dataset outlined in the `data.tf` file. It encompasses information about the parent subnet and IP addresses of all

---

resources, including subnets, as they can also be defined as sources or destinations. The scenario where subnets serve as their own parent subnet was also handled.

The necessity of supporting not just single ports, but multiple ports and port ranges, which cannot be configured via the same resource property, necessitated a solution. This was achieved through the utilization of a conditional expression, illustrated by the property `source_port_ranges`. The conditional expression evaluates whether the source ports definition, composed of a list, contains more than one object. If this is not the case, the property `source_port_range` is utilized accordingly.

---

## 7.4 Adjustments Deployment Manager

The task assignment specified the use of an existing Deployment Manager, developed by HL, for the deployment by the students of the generated labs by our application. This Deployment Manager incorporates the ability to implement Terraform deployments and present Terraform output concerning resource information to students.

However, since the existing Deployment Manager was tailored for static labs and specific scenarios, utilizing Guacamole for access to the resources, it required adaptation to fit our use case.

Owing to the Non-Disclosure Agreement (NDA) signed, it is not permitted to share the complete code of the Deployment Manager in the appendix or delve into extensive details about its implementation. Therefore, only the core modifications necessary for its compatibility with our product are documented in the subsequent sections, as agreed upon and authorized by the supervisor.

### Displaying Resource Information

To display the user-defined resource information as per the Terraform output outlined in Section 7.3.4, adjustments were made to the `getTerraformOutput` function in the backend part of the application, as outlined in Listing 7.34. Additionally, the OpenVPN profile was incorporated into the output to facilitate user access. Any not necessary information relating to specific HL deployments was removed.

```
1
2 const getTerraformOutput = async () => {
3   try {
4     const { stdout, _ } = await exec(`terraform output -json
5       -state=${TERRAFORM_STATE_FILE}`)
6     const output = JSON.parse(stdout)
7     return {
8       clientProfile: output?.dockovpn_client_profile,
9       resourcesConnectionInformation:
10        output?.resources_connection_information,
11        ...
12      }
13    } catch (e) {
14      return { error: "No output available" }
15    }
16  }
```

LISTING 7.34 Deployment Manager Output Function

In the application's frontend part, modifications were made to incorporate a table containing all resource information. This was achieved by looping over the `resources_connection_information` received from the backend to construct the table. Additionally, a download button was integrated to provide the OpenVPN profile

received from the `dockovpn_client_profile` output. Instructions to import and configure the OpenVPN profile on the HL Kookarai image were also appended.

The result of the view can be seen in Figure 7.4

Custom Label	Resource	IP Address	DNS Name	Admin Username	Admin Password	Credentials
glockendocker	dpl-kd-6v60ejh	10.100.1.17	dpl-kd-6v60ejh.b0179908-51ce-f2ff-7e1d-dfa89ade105f.dynpentestlab.ch	-	-	<b>Username Password</b> user1 cae70a9e-431f-c843-9bf5-08547fc1e40d user2 pw2
e1winwm	dpl-vm-8au37rx	10.100.2.4	dpl-vm-8au37rx.b0179908-51ce-f2ff-7e1d-dfa89ade105f.dynpentestlab.ch	e1admin	.*jg3E5Gp@wtMO	<b>Username Password</b> ivan1 ea27fd1d-c568-055e-f2de-3974586bb3c2 ivan2 gugus.2

**Destroy:** 2024-06-12T20:31:00Z

Download OpenVPN Profile ⓘ

ⓘ Setup VPN on KOOKARAI

```
nmcli connection import type openvpn file openvpnprofile.ovpn
nmcli connection modify openvpnprofile ipv4.never-default yes
nmcli con up id openvpnprofile
```

FIGURE 7.4 Deployment Manager View

### Support for Multiflags

To enable support for multiflags, it became necessary to ensure that the `MULTIFLAGS` environment variable, which is set automatically in the future by the HL Framework to the deployment manager, is injected into the terraform deployment within the multiflags variable definition. This required an adjustment to the `deployCommand` to read the environment and introduce it to terraform as seen in Listing 7.35.

```
1 const deployCommand = `cd ${TERRAFORM_FOLDER} \  
2   && terraform apply -auto-approve \  
3   ...  
4   -var multiflags='${process.env?.MULTIFLAGS}' \  
5   ...
```

LISTING 7.35 Deployment Manager Deploy Function

**Unused features** All unused features, such as the implemented specific scenarios where only select resources from a lab are deployed or the access with Guacamole, have been removed. This cleanup required thorough analysis of the application architecture and its construction. It was crucial to ensure that no checks depended on



---

the eliminated functionality and that it was entirely removed from all sections of the application without breaking other functionality.

This chapter provides a reflection on the achievements made and references them to the functional and non-functional requirements of this project.

## Framework for Pentest Lab Generation

The project successfully established a solid framework for generating pentest labs. This was achieved using Django [3] for backend operations offering REST-API endpoints and Angular [4] for the frontend, ensuring a scalable and maintainable solution. For the deployment part Terraform [7] was used, which dynamically configures and deploys the necessary resources based on the configuration applied in the frontend. This further fulfills the NFR requirements in Table 3.25, 3.30 and 3.32.

**Integration/Authentication** The integration and authentication within the pentest lab framework implemented using HL's KeyCloak service. The Generator application authenticates users through this service, ensuring secure access and management of the labs. The integration facilitates SSO and identity management across the HL platform, including the Generator Application.

**Lab Portability** Lab portability was a key feature implemented to ensure that labs can be easily saved, loaded, and shared. The persistence of labs on a tenant basis was facilitated through the use of a separate MinIO instance per tenant. Furthermore, the system supports the download of lab configurations as JSON files, which can be re-imported to recreate the lab environment. This allows the reuse of lab deployments across different tenants. This completes the NFR requirement in Table 3.31.

**Resource configuration** Resource creation in the framework is highly customizable. Lab-creators can specify various resources such as VMs, containers, subnets and firewall configurations. These resources are defined through a user-friendly interface, which then get translated into Terraform definitions for deployment.

**Resource customization** Beyond basic configuration, resource customization enables lab-creators to apply custom scripts and configurations to the resources. This includes using custom installation scripts from GitHub repositories and configuring Docker

---

containers with private and customized images. This level of customization ensures that labs can be highly specialized and aligned with the educational objectives.

**Lab Deployment** The Generator application can facilitate deployments directly to Azure in order to test the deployment and capture snapshots of the VMs. The snapshots capture the state of the VMs, including their custom installations. This feature significantly reduces the deployment time for students, allowing them to start using the lab environment much faster than going through the entire installation process. Furthermore, it can be assured that the installation on the deployment is complete and working for all students because it was tested and captured in the snapshot beforehand.

**Lab Validation** While fundamental validation is conducted in the backend, such as verifying the model's required fields and the proper configuration of resources, additional validation processes like input verification or validation of previously configured labs with outdated SKUs were not implemented in the frontend due to lack of time and prioritization of other features. The groundwork for input validation, via regex definition in the model structure, was nonetheless laid down. Generic errors stemming from the backend are displayed as notifications on the frontend, mirroring the outcomes of the basic backend validation.

### **Network with Multiple Subnets**

The framework supports the creation of labs with multiple subnets, enabling complex network segmentation and the simulation of real-world networks.

### **Firewall to Restrict Communication Between Resources**

A firewall-like solution was implemented using Azure security groups, which define and enforce network restrictions based on specified rules. Based on the specific needs for the lab, the lab-creator can define these rules through the frontend, specifying the actions to be granted or denied, the source/destination, as well as the relevant ports or port ranges and the protocol

### **DNS Resolution for Resources**

DNS resolution was set up using Cloudflare for the root domain, while Azure DNS managed subdomains for individual lab environments. These subdomains are automatically configured with a unique UUID for each lab deployment. This setup ensures that resources can be easily accessed using DNS.

---

## **Virtual Machines (Linux, Windows)**

The framework supports the deployment of VMs operating on both Linux and Windows. It allows the lab-creator to specify parameters including OSs, disk capacities, and VM size in terms of CPU and RAM. To ensure broad compatibility, various distributions including Windows Server, Windows Desktop, Ubuntu, Red Hat Enterprise Linux (RHEL), and Suse were tested and confirmed to function without issues.

### **Possibility to perform Custom Installations on Virtual Machines**

Custom installations for VMs were implemented via Github repositories. This enables users to specify a specific repository for each VM. These repositories are subsequently then used during the VM deployment process to install the required software and configurations. The repositories can be set up with two scripts: one for handling the installation and another for defining post-install routines that can be leveraged to distribute multiflags and credentials. Sample repositories have been created to demonstrate fundamental components such as access to multiflags or credentials, along with appropriate error handling. Additionally, a mechanism was implemented to supply the lab-creator with adequate script output information for debugging, facilitated through the Terraform output.

### **Container Services**

Support for container services was integrated, utilizing an Azure Kubernetes Cluster (AKC). This adoption demonstrates a state-of-the-art deployment strategy for containerized applications, satisfying the demands for interoperability for the current Hacking-lab images and isolation through dynamically deploying the corresponding Nodepools for the subnets within the lab architecture framework. It also enables the use of Terraform definitions for deployment, thereby facilitating advanced cloud deployments.

### **Possibility to Deploy Existing HL Container Images**

The framework supports deploying existing HL container images, ensuring compatibility with current already developed images. This is achieved by allowing the definition of custom repositories, along with the necessary credentials, especially if the repository is private. This fulfills the NFR requirements in [Table 3.27](#) and [3.29](#).

---

## **VPN Access for Students**

A secure VPN access method was implemented using OpenVPN. This ensures that students can securely connect to the lab environments. The OpenVPN server was deployed as a containerized application within the Kubernetes cluster, providing robust and secure remote access. Furthermore the OpenVPN Server has been made configurable to accommodate potential future modifications.

## **Adjustments to Existing Deployment Manager**

Enhancements were made to the existing deployment manager to adapt to the dynamically generated Terraform deployments, particularly regarding the output to be displayed, which is dynamically generated from them. Furthermore, it was adjusted to be able to inject multiflags into the Terraform deployment, which are then used by the deployment to distribute to the VMs and containers. This completes the NFR requirement in Table [3.28](#).

## **Dynamic Flag Distribution**

The framework incorporates features for dynamic flag distribution, a critical component for ensuring that the student has genuinely resolved the challenge rather than receiving and using the flags from other students. This was accomplished through the adjustments made to the Deployment Manager and the distribution process to the VMs and containers through the Terraform deployment, followed by the exemplary installation of repositories on how to retrieve the flags for actual implementation into the pentest components.

## **Cost Optimization**

To comply with NFR [3.33](#), several strategies were implemented to optimize costs. Evaluations of costs were performed during the evaluation of solutions like container deployment or VPN establishment. Dynamic scaling was incorporated within the utilized AKS, ensuring only the necessary number of nodes are in operation. Automated purging processes in the backend facilitated the deletion of unsuccessful deployments and orphaned resources, thus averting unnecessary expenses. The adoption of VM snapshots not only decreased deployment times but also reduced computing time costs linked with repetitive installations. For DNS management, cost-effectiveness was achieved by leveraging the free service Cloudflare.

## Conclusion and Outlook **9**

---

This chapter provides a conclusion, a future outlook and recommendation for the project.

The core objective of this thesis was to develop a tool that would simplify the creation and integration of individual pentest labs into the existing HL infrastructure. This tool was designed to enhance the practical application of cybersecurity concepts within an environment that closely mimics corporate IT landscapes, thereby providing a more realistic and comprehensive learning experience.

The successful establishment of a robust foundation and development of the Dynamic Pentest Lab Generator Framework is noteworthy. However, it is crucial to examine specific aspects of the product and, considering its scope and complexity, to contemplate potential future enhancements, further detailed in the subsequent section.

The Terraform deployment has demonstrated the potential for dynamic lab generation through established templating methodologies. It enables the deployment of virtual machines, offering customization options via custom installation repositories, and supports the creation of snapshots to reduce setup times for students. Additionally, the process incorporates robust network configurations with the creation of subnets and designed to simulate real-world IT environments. VPN access for students is streamlined using an OpenVPN container, ensuring secure remote connectivity. The implementation of DNS resolution through Cloudflare and Azure DNS zones enhances the usability of accessing resources. The deployment of containers facilitates the reuse of existing HL Docker images, further enhancing the practicality and relevance of the labs through a state-of-the-art deployment approach for containerized applications via Kubernetes Cluster. Moreover, the framework includes mechanisms for distributing dynamic flags and credentials to the resources, crucial for conducting varied and complex security exercises. The implementation of traffic restrictions, managed through security groups and rules, effectively controls and restricts traffic between the deployed resources.

In future enhancements, an approach to introduce Transport Layer Security (TLS) for containers through the use of side containers as reverse proxies could be taken.

---

Leveraging Let's Encrypt for automated certificate management would seamlessly fit with the existing use of the public DNS solutions Cloudflare and Azure DNS zones. This integration would effectively streamline the integration and renewal processes for Secure Socket Layer (SSL)/TLS certificates.

Furthermore, the Terraform Azure resources, `azurerm_linux_virtual_machine` and `azurerm_windows_virtual_machine`, should be monitored for their ability to support existing OS disk attachments. The current scenario necessitates the utilization of the Terraform resource `azurerm_virtual_machine` to enable the snapshot technology, due to the lack of such a feature in the aforementioned resources. However, this Terraform resource has also introduced certain complications, given that it is not receiving the same enhancement features as the newer resources. These complications have been extensively delineated in Section 7.3.6 which could be solved when utilizing the newer resources.

The established VPN Access solution for students was engineered to be able to dynamically retrieve the VPN profiles and therefore providing great usability since only the profile has to be imported to build up the tunnel. Also it was configured in that way that only the corresponding lab traffic gets routed through the tunnel. However, utilizing a VPN connection for educational purposes within a corporate environment may pose a challenge. This is because most companies typically prohibit the establishment of VPNs within their corporate networks. An alternative solution could be the use of Azure Virtual Desktop as an entry point for the lab, instead of the VPN. Azure Virtual Desktop offers a virtual client that can be directly accessed in the browser. Its deployment also supports the use of custom images, which would allow the integration of the HL Kookarai image, thereby providing all the necessary tools for the pentest lab. There are corresponding Terraform resources for the deployment of Azure Virtual Desktop. This could be integrated into the existing Terraform deployment in such a way that, based on the provided configuration, either the VPN or Azure Virtual Desktop is used as the lab's entry point. Although this alternative would be a robust solution, it would incur additional costs, unlike the OpenVPN container which doesn't introduce any extra expenses as it operates on the already existing Kubernetes Cluster.

The backend serves as a crucial intermediary between the user-facing frontend and the underlying infrastructure management performed by Terraform. Its primary responsibilities include providing API endpoints for the frontend, initiating Terraform deployments, and providing the deployment including the Deployment Manager in the form of a `dockerfiles.tar.gz`. Creating a seamless bridge between the frontend and backend presented significant challenges due to the peculiarities inherent in the different technologies used. These peculiarities, such as the frontend providing slightly adjusted keys in the JSON or Terraform requiring correctly capitalized user inputs, had to be mitigated by the backend. This ensured that user inputs from the frontend ultimately generated valid Terraform code, as even a small mistake could cause the entire deployment to fail. Handling the statuses of all incoming Terraform deployments

---

along with their associated logic posed a significant challenge. It was particularly critical, yet difficult, to ensure that labs were consistently and appropriately cleaned up, even in the face of errors. Such errors could include logical flaws in the user-provided configuration that the backend parser or Terraform's initialization process did not detect. This aspect was key to avoiding any additional costs due to orphaned deployments.

While the backend could still benefit from additional error handling and general code cleanup, the overall quality is good, and the logic is handled cleanly given its complexity. Adjustments in the backend will always be closely tied to changes in the user interface it dynamically hydrates or to updates in Terraform.

When considering potential enhancements, limitations could be established on the accessible SKUs and the number of virtual machines or containers that may be incorporated into a lab. A strategy could also be developed to allocate a budget for costs incurred per lab, on an hourly basis. This could be achieved by dynamically retrieving and validating the costs for each resource added to the lab. While this was not a requirement for this project, considering the lab-creators as trustworthy actors, it may be prudent to implement such restrictions in less trustful environments to avoid excessive costs.

The development of the Generator's frontend presented substantial effort and notable challenges. On one hand, the frontend needed to meet all requirements and include all functionalities for the Terraform deployments and backend logic to be useful to the end-user. On the other hand, there were numerous technical requirements set by the developers of Hacking-Lab AG. The requirements of the specific frameworks, that had to be used for this project, was only announced after the fourth week already in preparation and development. Therefore a compromise had to be made and it was decided to discard the existing frontend in week four and redo it in the freshly defined frameworks. Overcoming the steep learning curve, due to new frameworks and technologies, required considerable time and resources. In the end, these challenges were overcome, resulting in a working application that allows users to visually create lab environments. This includes creating network nodes, configuring the resources represented by those nodes, and connecting them to different subnets that can also be user-created. Furthermore it serves as a central point for management of the labs. This concludes actions such as initiating deployments, downloading deployments, analyzing logs for debugging, and inspecting deployment statuses.

The frontend is not just a static web application displaying an editable graph with some data; it is dynamically built to be expanded and improved without advanced frontend knowledge. This was achieved by building the entire lab creation process in the frontend around the structural JSON files defined by the backend. This approach allows the frontend to dynamically render fully functional content without changing a single line of code in the frontend itself. The frontend consists of different building blocks rendered via templating according to the backend-specified model structure



---

JSON file. This makes the application well-maintained and easily adaptable to new use cases, such as deploying new resources in future cloud-based lab environments.

Despite this, the frontend still requires refinement and is missing certain user-experience functionalities. The primary focus was on core elements, such as features associated with cloud deployment and lab creation, treating user-experience features as secondary, to be addressed if extra resources were available. Ideally, higher priority would have been given to user feedback and input validation during resource modification and validating the correctness of the supplied values. The backend already supplies the necessary information and regular expressions to theoretically validate user input for each resource property. However, the development of basic functionality in the frontend required substantial resources, leaving this aspect incomplete.

Another aspect is the overall design of the application, which currently appears quite rudimentary and practical. With some effort, the design could be significantly improved and adjusted to match the HL Framework even more. Lastly, a feature on the list of optional goals was replacing the simple graph nodes with fitting icons. This is partially implemented on the backend by sending a base64-encoded image for each resource. These images could be used to visualize the resources in the graph more clearly, enhancing the application's visual appeal. This feature ties into the design improvement but was specifically planned as an optional enhancement.

The existing framework, in general, could be adapted to replace the current Deployment Manager. Its capability to deploy Terraform labs and centrally manage them could be utilized directly by students for deployment purposes. By using the already integrated authentication service, a role specifically for students could be created, allowing them only to deploy specific labs and view their own deployments.

In conclusion, with a total investment of 1294 hours, this project has successfully established a framework, providing significant benefits in efficiency, accessibility, and integration for creating pentest labs. Simultaneously, it also has paved the way for further expansion of HL into the public cloud native realm. Future enhancements will further refine the tool, ensuring it remains adaptable and relevant in the evolving field of cybersecurity.

# Personal Reports 10

---

## Personal Report - Dante

This project marked my first experience being solely responsible for an entire frontend implementation, including its creation and architectural design. Meeting the requirements from both the deployment and backend teams sometimes presented significant challenges, particularly in developing a suitable frontend to display these requirements effectively.

Initially working with new technologies and frameworks was a setback. Four weeks into the project, I had to switch from React to Angular with DaisyUI and Tailwind, resulting in several lost hours and the need to rewrite the existing frontend. Integrating the vis-network library with our custom backend data structure was another complex and often unintuitive task, which cost me a couple extra hours.

The project involved numerous requirements and feature requests from different parties, which required prioritization, the translation into the frontend architecture, and eventual implementation. This was sometimes on a tight schedule, with features needing to be developed overnight or within a few days, alongside managing a normal job and attending other lectures. Consequently, I accumulated a significant surplus of hours compared to the planned hours for the project credits.

Despite these challenges, I am extremely happy and content with the final result we achieved as a team. We successfully integrated the backend, deployment, and frontend into a fully functional application. I am proud of my accomplishment in building the frontend from scratch, including its configuration, setup, and scalability for future needs. Implementing dynamic rendering was initially daunting, but I managed to achieve it by staying calm in stressful situations and relying on the support of my great teammates, who consistently pushed me and believed in my abilities.

I am also very grateful to our supervisor, Ivan, who was always ready to help, open to our approach and implementation, and provided us with a lot of trust. His support was invaluable throughout the project.

---

## Personal Report - Janosch

In this project, I was tasked with developing the backend in Django. This role included designing and implementing the server-side logic, database interactions, and core functionalities that our application required. From setting up the initial project structure to ensuring seamless integration with other components, the responsibility was comprehensive and demanding.

Working with Django was both familiar and challenging. While I had prior experience with Python web/API frameworks, the complexity of this project required a deeper understanding and innovative problem-solving skills. We needed to ensure that the data structures and the code in general were easily maintainable by others, that the project was generic and configurable enough to be deployed in various tenants, that it scaled well, and of course, to implement the wishlist of features.

Because all of us were working part-time (40-60%) on different days and also having different study schedules, we had a tight schedule. This led to occasional long late-night meetings after a day of already working 8-9 hours, which was hard on all of us. Despite these challenges, the communication in the team was good and fruitful. Regular check-ins, clear documentation, and collaborative problem-solving were essential to our success. This experience highlighted the importance of effective teamwork and the ability to adapt to varying schedules and high workloads.

This project did feel less like a thesis and more like being a group of freelancers developing a product, but I guess that is a peculiarity in IT. I'm very glad that the real-world setting of our work enabled us to create a product that will be used in the future and not just discarded after completion. However, I do wish we had more time to focus on the academic aspects of our thesis, as the typical IT thesis at OST heavily emphasizes product creation similar to real-world development. This emphasis is understandable, given that OST is a university of applied sciences.

I want to express my gratitude to my teammates for their excellent collaboration and to our supervisor, Ivan. He was always ready to help, open to our approach and implementation, and placed a great deal of trust in us. His support was invaluable throughout the project.

---

## Personal Report - Samuel

Throughout this project, I was tasked with the responsibility of managing the Terraform deployment. While the backend and frontend components were designed to leverage this deployment for a user-friendly experience, my role was to implement the actual logic, architecture, and dynamic aspects of the lab deployments.

Being very interested in cloud deployments and automation, I took great pleasure in designing, overthinking, and building the necessary logic to fulfill the requirements. I had worked on smaller deployments earlier but never had the chance to design such a complex deployment with numerous dependencies and components involved. The challenge of building everything dynamically was something I particularly enjoyed. It required establishing corresponding designs to facilitate this. Furthermore, existing bugs in certain Terraform resources or feature incompleteness required additional troubleshooting and engineering to meet the requirements.

In general, I am very happy with the result. Having built a Terraform deployment that facilitates the deployment of VMs, a Kubernetes cluster to run containers, performs custom installations and debugging of them, automates the snapshot process, provides network restrictions, establishes DNS resolution with public resolution, and provides VPN access for the students — all dynamically and unique, solely based on a JSON file — is something I am very proud of.

Despite the enjoyment I found in the work, it was a challenging time. The project requirements and tight schedule often led to late-night sessions dedicated to implementing, debugging, and improving the deployments to meet the necessary requirements and ensure their stability. Additionally, like all of us, managing a regular job and other academic tasks left little room for free time. As a result, I also accumulated a surplus of hours compared to the planned hours for the project credits.

Overall, I am also very grateful for the teamwork. Without such a great performance from all of us and the team spirit to build something innovative and functional, we wouldn't have achieved such success. I also want to thank Ivan, our supervisor, for this great assignment and the support and help he provided.

## List of Figures

---

2.1	Manual Pentest Creation System-Context . . . . .	6
2.2	Dynamic Pentest Generator System-Context . . . . .	7
3.1	Use-Case Diagram . . . . .	9
4.1	DPL Domain Model . . . . .	37
4.2	C4 Context Diagram . . . . .	38
4.3	C4 Container Diagram . . . . .	40
4.4	C4 Component Diagram . . . . .	42
4.5	Azure Lab Architecture . . . . .	44
4.6	Frontend Figma Overview . . . . .	53
4.7	Frontend Figma Flow . . . . .	54
4.8	Frontend Architecture Diagram . . . . .	56
4.9	Pricing Azure Container Deployment . . . . .	74
6.1	Git Branching Workflow . . . . .	86
7.1	Info Page . . . . .	98
7.2	Lab Editor Page . . . . .	99
7.3	Labs Page . . . . .	100
7.4	Deployment Manager View . . . . .	146

## List of Tables

---

3.1	User Story: Create Subnets Resources . . . . .	12
3.2	User Story: Create Virtual Machine Resources . . . . .	13
3.3	User Story: Create Container Resources . . . . .	14
3.4	User Story: Create Connection between Resources . . . . .	15
3.5	User Story: Access to the Lab via VPN . . . . .	16
3.6	User Story: Subnet Configuration . . . . .	16
3.7	User Story: Container Configuration . . . . .	17
3.8	User Story: Virtual Machine Configuration . . . . .	18
3.9	User Story: Firewall Configuration . . . . .	19
3.10	User Story: Custom Installation Repositories . . . . .	20
3.11	User Story: Customized Installations VM . . . . .	21
3.12	User Story: Custom/HL Container Images from Private Registry . . . . .	22
3.13	User Story: Selective Resource Exposure . . . . .	22
3.14	User Story: Credential Management and Distribution . . . . .	23
3.15	User Story: Multiflag Integration and Management . . . . .	24
3.16	User Story: Save Labs . . . . .	24
3.17	User Story: Load Labs . . . . .	25
3.18	Export Lab for Deployment Manager . . . . .	25
3.19	User Story: Download Designed lab . . . . .	26
3.20	User Story: Upload existing Lab . . . . .	27
3.21	User Story: Lab Validation . . . . .	28
3.22	User Story: Deploy Lab . . . . .	28
3.23	User Story: Snapshot Lab . . . . .	29
3.24	User Story: Authenticate with HL SSO . . . . .	30
3.25	Non-Functional Requirements: NFR01 . . . . .	31
3.26	Non-Functional Requirements: NFR02 . . . . .	31
3.27	Non-Functional Requirements: NFR03 . . . . .	32
3.28	Non-Functional Requirements: NFR04 . . . . .	32
3.29	Non-Functional Requirements: NFR05 . . . . .	32
3.30	Non-Functional Requirements: NFR06 . . . . .	33
3.31	Non-Functional Requirements: NFR07 . . . . .	33
3.32	Non-Functional Requirements: NFR08 . . . . .	34
3.33	Non-Functional Requirements: NFR09 . . . . .	34
3.34	Non-Functional Requirements: NFR10 . . . . .	35

3.35	Tracking of the NFRs . . . . .	35
4.1	Azure Resource Naming . . . . .	48
4.2	Container Deployment Evaluation . . . . .	75

## List of Listings

---

4.1	Model Data Structure . . . . .	51
4.2	Model Data Structure for a new Resource . . . . .	58
7.1	Dynamic Button Creation through JSON File . . . . .	102
7.2	Resource Editor Text Input Example . . . . .	103
7.3	Azure Offer Dropdown Component . . . . .	104
7.4	Notification Component . . . . .	107
7.5	Celery Task Chain . . . . .	118
7.6	Terraform Structure . . . . .	121
7.7	Virtual machine definition . . . . .	122
7.8	Configuration Definition . . . . .	123
7.9	Dynamic Generation utilizing Loops . . . . .	124
7.10	Conditional for each . . . . .	124
7.11	Dynamic Block . . . . .	124
7.12	Creation of dynamic Credentials . . . . .	125
7.13	Resource Information Dataset . . . . .	126
7.14	Kubernetes Provider . . . . .	127
7.15	Resource Group Name . . . . .	127
7.16	Dynamic Creation of Node Pools . . . . .	128
7.17	Dockovpn Deployment ENV . . . . .	130
7.18	Root DNS Zone NS Records . . . . .	131
7.19	Creation of DNS A-Records . . . . .	131
7.20	Virtual Machine Sleep Provisioner . . . . .	133
7.21	Creation of virtual Machine Snapshot . . . . .	133
7.22	Usage of virtual Machine Snapshots . . . . .	134
7.23	Custom installation Linux . . . . .	136
7.24	Example Install Output . . . . .	137
7.25	Example failing Install Output . . . . .	138
7.26	Error handling Powershell . . . . .	139
7.27	Error handling Bash . . . . .	139
7.28	Deployment Node Selector . . . . .	140
7.29	Kubernetes Manifest wait Configuration . . . . .	141
7.30	Multiflag env Parsing for Container . . . . .	141
7.31	Env Assignment Container . . . . .	141



7.32 Docker Pull Secrets . . . . .	142
7.33 Network Security Rules . . . . .	143
7.34 Deployment Manager Output Function . . . . .	145
7.35 Deployment Manager Deploy Function . . . . .	146

## Bibliography

---

- [1] „Visio“. (2024), [Online]. Available: <https://www.microsoft.com/en-us/microsoft-365/visio/flowchart-software> (visited on 14/06/2024).
- [2] adr.github.io. „Architectural decision records (adrs)“. (2024), [Online]. Available: <https://adr.github.io/> (visited on 12/06/2024).
- [3] T. C. D. contributors. „Cookiecutter django“. (2024), [Online]. Available: <https://github.com/cookiecutter/cookiecutter-django/> (visited on 12/06/2024).
- [4] A. T. at Google. „Angular - the modern web developer’s platform“. (2024), [Online]. Available: <https://github.com/angular/angular> (visited on 12/06/2024).
- [5] T. vis-network contributors. „Vis-network“. (2024), [Online]. Available: <https://github.com/visjs/vis-network> (visited on 12/06/2024).
- [6] „Hacking-lab“. (2024), [Online]. Available: <https://www.hacking-lab.com/services/> (visited on 14/06/2024).
- [7] terraform.io. „Terraform“. (2024), [Online]. Available: <https://www.terraform.io/> (visited on 12/06/2024).
- [8] „Azure cloud computing azure“. (2024), [Online]. Available: <https://azure.microsoft.com> (visited on 12/06/2024).
- [9] „Managed Kubernetes Service (AKS) | Microsoft Azure“. (2024), [Online]. Available: <https://azure.microsoft.com/en-us/products/kubernetes-service> (visited on 13/06/2024).
- [10] github.com. „Github“. (2024), [Online]. Available: <https://github.com/> (visited on 12/06/2024).
- [11] „Secure access and network connectivity reimaged“. (2024), [Online]. Available: <https://openvpn.net/> (visited on 12/06/2024).
- [12] Cloudflare. „Cloudflare“. (2024), [Online]. Available: <https://www.cloudflare.com> (visited on 12/06/2024).
- [13] c4model.com. „C4 model for visualising software architecture“. (2024), [Online]. Available: <https://c4model.com/> (visited on 12/06/2024).
- [14] „Let’s Encrypt“. (2024), [Online]. Available: <https://letsencrypt.org/> (visited on 14/06/2024).

- 
- [15] docker.com. „Docker platform for delivering software in containers“. (2024), [Online]. Available: <https://www.docker.com/> (visited on 12/06/2024).
- [16] G. Contributors. „Git version control system“. (2024), [Online]. Available: <https://github.com/git/git> (visited on 12/06/2024).
- [17] redis.io. „Redis: In-memory data structure store“. (2024), [Online]. Available: <https://redis.io/> (visited on 12/06/2024).
- [18] celeryproject.org. „Celery: Distributed task queue“. (2024), [Online]. Available: <https://docs.celeryproject.org/en/stable/> (visited on 12/06/2024).
- [19] min.io. „Minio | s3 kubernetes native object storage for ai“. (2024), [Online]. Available: <https://min.io/> (visited on 09/06/2024).
- [20] M. Azure. „Azure products“. (2024), [Online]. Available: <https://azure.microsoft.com/en-us/products/> (visited on 12/06/2024).
- [21] M. Support. „Resource naming restrictions - azure resource manager“. (2024), [Online]. Available: <https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/resource-name-rules> (visited on 12/06/2024).
- [22] F. GmbH. „Figma“. (2024), [Online]. Available: <https://www.figma.com/> (visited on 12/06/2024).
- [23] T. daisyUI contributors. „Daisyui“. (2024), [Online]. Available: <https://github.com/saadeghi/daisyui> (visited on 12/06/2024).
- [24] react.dev. „React as web frontend framework“. (2024), [Online]. Available: <https://react.dev> (visited on 12/06/2024).
- [25] vuejs.org. „Vuejs as web frontend framework“. (2024), [Online]. Available: <https://vuejs.org> (visited on 12/06/2024).
- [26] T. R.-D. Contributors. „React dnd as graph framework“. (2024), [Online]. Available: <https://react-dnd.github.io/react-dnd/docs/overview> (visited on 12/06/2024).
- [27] jointjs.com/. „Jointjs vis as graph framework“. (2024), [Online]. Available: <https://jointjs.com> (visited on 12/06/2024).
- [28] js.cytoscape.org. „Jointjs vis as graph framework“. (2024), [Online]. Available: <https://js.cytoscape.org> (visited on 12/06/2024).
- [29] tailwindcss.com. „Tailwind as css framework“. (2024), [Online]. Available: <https://tailwindcss.com> (visited on 12/06/2024).
- [30] daisyui.com. „Daisy-ui as component library in addition to tailwind“. (2024), [Online]. Available: <https://daisyui.com> (visited on 12/06/2024).
- [31] react-bootstrap.netlify.app. „React bootstrap ui as css framework“. (2024), [Online]. Available: <https://react-bootstrap.netlify.app> (visited on 12/06/2024).
- [32] mui.com. „Material ui as css framework“. (2024), [Online]. Available: <https://mui.com> (visited on 12/06/2024).

- [33] „Chocolatey“. (2024), [Online]. Available: <https://chocolatey.org/> (visited on 14/06/2024).
- [34] „Nix“. (2024), [Online]. Available: <https://nixos.org/> (visited on 14/06/2024).
- [35] „Introduction to snaps“. (2024), [Online]. Available: <https://ubuntu.com/core/services/guide/snaps-intro> (visited on 14/06/2024).
- [36] „Azure Container Instances | Microsoft Azure“. (2024), [Online]. Available: <https://azure.microsoft.com/de-de/products/container-instances> (visited on 13/06/2024).
- [37] just-containers. „Just-containers/s6-overlay“. (2024), [Online]. Available: <https://github.com/just-containers/s6-overlay> (visited on 12/06/2024).
- [38] M. Support. „S6-overlay support in container instances“. (2024), [Online]. Available: <https://learn.microsoft.com/en-us/answers/questions/1153255/s6-overlay-support-in-container-instances> (visited on 12/06/2024).
- [39] M. Support. „About Azure Point-to-Site VPN connections - Azure VPN Gateway“. (2023), [Online]. Available: <https://learn.microsoft.com/en-us/azure/vpn-gateway/point-to-site-about> (visited on 12/06/2024).
- [40] DockOVPN. „Dockovpn“. (2024), [Online]. Available: <https://dockovpn.io/index.html> (visited on 12/06/2024).
- [41] J. A. Donenfeld. „Wireguard: Fast, modern, secure vpn tunnel“. (2024), [Online]. Available: <https://www.wireguard.com/> (visited on 12/06/2024).
- [42] „Keycloak“. (2024), [Online]. Available: <https://www.keycloak.org/> (visited on 13/06/2024).
- [43] <https://github.com/manfredsteyer/angular-oauth2-oidc>. „About azure point-to-site vpn connections - azure vpn gateway“. (2024), [Online]. Available: <https://github.com/manfredsteyer/angular-oauth2-oidc> (visited on 12/06/2024).
- [44] Okta. „Auth0 by okta“. (2024), [Online]. Available: <https://auth0.com/blog/complete-guide-to-angular-user-authentication/> (visited on 12/06/2024).
- [45] P. Contributors. „Welcome to pyjwt“. (2024), [Online]. Available: <https://pyjwt.readthedocs.io/en/stable/> (visited on 12/06/2024).
- [46] P. A. Contributors. „The ultimate python library in building oauth and openid connect servers and clients.“ (2024), [Online]. Available: <https://pypi.org/project/Authlib/> (visited on 12/06/2024).
- [47] D.-O. Contributors. „Django oauth toolkit“. (2024), [Online]. Available: <https://django-oauth-toolkit.readthedocs.io/en/stable/index.html> (visited on 12/06/2024).
- [48] greg-lindsay. „What is an azure private dns zone?“ (2024), [Online]. Available: <https://learn.microsoft.com/en-us/azure/dns/private-dns-privatednszone> (visited on 12/06/2024).

- [49] greg-lindsay. „Dns zones and records overview - azure public dns“. (2024), [Online]. Available: <https://learn.microsoft.com/en-us/azure/dns/dns-zones-records> (visited on 12/06/2024).
- [50] T. Registry. „Cloudflare<sub>record(resource)</sub>“. (2024), [Online]. Available: <https://registry.terraform.io/providers/cloudflare/cloudflare/latest/docs/resources/record> (visited on 12/06/2024).
- [51] T. Registry. „Azurerm<sub>dns\_a\_record</sub>“. (2024), [Online]. Available: [https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/dns\\_a\\_record](https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/dns_a_record) (visited on 12/06/2024).
- [52] M. ATT&CK. „Gather victim network information: Dns, sub-technique t1590.002 - enterprise“. (2024), [Online]. Available: <https://attack.mitre.org/techniques/T1590/002/> (visited on 12/06/2024).
- [53] Microsoft. „Typescript“. (2024), [Online]. Available: <https://github.com/microsoft/TypeScript> (visited on 12/06/2024).
- [54] T. ruff contributors. „Typescript-eslint“. (2024), [Online]. Available: <https://github.com/typescript-eslint/typescript-eslint> (visited on 12/06/2024).
- [55] T. ruff contributors. „Ruff“. (2024), [Online]. Available: <https://github.com/astral-sh/ruff> (visited on 12/06/2024).
- [56] postgresql.org. „Postgresql database“. (2024), [Online]. Available: <https://www.postgresql.org/> (visited on 12/06/2024).
- [57] kubernetes.io. „Kubernetes container orchestration platform“. (2024), [Online]. Available: <https://kubernetes.io/> (visited on 12/06/2024).
- [58] tflint. „Tflint“. (2024), [Online]. Available: <https://github.com/terraform-linters/tflint> (visited on 13/06/2024).
- [59] visualstudio.com. „Visual studio code“. (2024), [Online]. Available: <https://code.visualstudio.com/> (visited on 12/06/2024).
- [60] jetbrains.com. „Jetbrains pycharm“. (2024), [Online]. Available: <https://www.jetbrains.com/pycharm/> (visited on 12/06/2024).
- [61] jetbrains.com. „Jetbrains webstorm“. (2024), [Online]. Available: <https://www.jetbrains.com/webstorm/> (visited on 13/06/2024).
- [62] postman.com. „Postman api platform“. (2024), [Online]. Available: <https://www.postman.com/> (visited on 12/06/2024).
- [63] insomnia.rest. „Insomnia: The api design platform and rest client“. (2024), [Online]. Available: <https://insomnia.rest/> (visited on 12/06/2024).
- [64] latex-project.org. „Latex document preparation system“. (2024), [Online]. Available: <https://www.latex-project.org/> (visited on 12/06/2024).
- [65] overleaf.com. „Overleaf collaboration tool for latex“. (2024), [Online]. Available: <https://www.overleaf.com/> (visited on 12/06/2024).

- [66] jetbrains.com. „Jetbrains qodana“. (2024), [Online]. Available: <https://www.jetbrains.com/qodana/> (visited on 12/06/2024).
- [67] zotero.org. „Zotero“. (2024), [Online]. Available: <https://www.zotero.org/> (visited on 13/06/2024).
- [68] T. pre-commit contributors. „Pre-commit“. (2024), [Online]. Available: <https://pre-commit.com/> (visited on 12/06/2024).
- [69] atlassian.com. „Jira bug tracking and project management“. (2024), [Online]. Available: <https://www.atlassian.com/software/jira> (visited on 12/06/2024).
- [70] atlassian.com. „Confluence collaboration software“. (2024), [Online]. Available: <https://www.atlassian.com/software/confluence> (visited on 12/06/2024).
- [71] miro.com. „Miro collaborative online whiteboarding platform“. (2024), [Online]. Available: <https://www.miro.com/> (visited on 12/06/2024).
- [72] microsoft.com. „Teams virtual meetings and chatting tool“. (2024), [Online]. Available: <https://www.microsoft.com/en-us/microsoft-teams/> (visited on 12/06/2024).
- [73] nginx.com. „Nginx web server and reverse proxy“. (2024), [Online]. Available: <https://www.nginx.com/> (visited on 12/06/2024).
- [74] T. ngx-markdown Contributors. „Ngx-markdown library“. (2024), [Online]. Available: <https://github.com/jfcere/ngx-markdown> (visited on 12/06/2024).
- [75] T. B. Practices. „Code structure“. (2022), [Online]. Available: <https://www.terraform-best-practices.com/code-structure> (visited on 12/06/2024).
- [76] M. Support. „Analyze decision criteria - training“. (2024), [Online]. Available: <https://learn.microsoft.com/en-us/training/modules/choose-network-plugin-aks/3-analyze-decision-criteria> (visited on 12/06/2024).
- [77] K. Documentation. „Assigning pods to nodes“. (2024), [Online]. Available: <https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/> (visited on 12/06/2024).
- [78] T. D. contributors. „Dockovpn“. (2024), [Online]. Available: <https://github.com/dockovpn/dockovpn> (visited on 12/06/2024).
- [79] T. Registry. „Support for managed disk from an existing disk“. (2024), [Online]. Available: <https://github.com/hashicorp/terraform-provider-azurerm/issues/8195> (visited on 12/06/2024).
- [80] T. Registry. „Azurerm linux virtual machine“. (2024), [Online]. Available: [https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/linux\\_virtual\\_machine](https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/linux_virtual_machine) (visited on 12/06/2024).
- [81] M. Support. „Create a vm from a specialized image version - azure virtual machines“. (2024), [Online]. Available: <https://learn.microsoft.com/en-us/azure/virtual-machines/vm-specialized-image-version> (visited on 12/06/2024).

- 
- [82] T. T. P. A. Contributors. „Support for importing osdisk“. (2024), [Online]. Available: <https://github.com/hashicorp/terraform-provider-azurerm/issues/8794> (visited on 12/06/2024).
- [83] T. Registry. „Meta issue: Support for terraform-plugin-framework“. (2024), [Online]. Available: <https://github.com/hashicorp/terraform-provider-azurerm/issues/25765> (visited on 12/06/2024).
- [84] T. A. Contributors. „[bug] agent breaks when deploying template with vm and runcommand“. (2024), [Online]. Available: <https://github.com/Azure/WALinuxAgent/issues/2521> (visited on 12/06/2024).
- [85] T. Registry. „Error: Failed to construct rest client on kubernetes\_manifest resource“. (2024), [Online]. Available: <https://github.com/hashicorp/terraform-provider-kubernetes/issues/1775> (visited on 12/06/2024).

# Glossary

---

**Capture The Flag** A competitive cybersecurity exercise in which participants tackle security problems to capture or defend computer systems. Participants must then discover and exploit hidden "flags," secret values protected by system vulnerabilities, in order to score points.. 5

**Hacking-Lab** Hacking-Lab is a platform that provides ethical hacking for educational purposes. The overall goal is to promote awareness in the field of hacking and security. The corresponding challenges are developed in the format of a capture the flag scenario.. 2

**Infrastructure as a Service** A cloud computing model which offers infrastructure components such as virtual machines, storage, and networking resources on a pay-as-you-go basis.. 79

**Network Security Group** A Network Security Group (NSG) is a feature that acts as a virtual firewall for virtual machines, containers, and other resources within a virtual network. It manages inbound and outbound traffic using security rules that allow or deny access based on IP address, port, and protocol. 46

**Penetration Testing** A method of evaluating the security of a computer system or network by simulating an attack from malicious outsiders. Penetration testing involves attempting to exploit system vulnerabilities. Such assessments are valuable for validating the effectiveness of defensive mechanisms and end-user adherence to security policies.. ii

**RESTful API** An architectural style for designing networked applications, using a set of constraints to create scalable and stateless communication over HTTP. 33



# Acronyms

---

- ACI** Azure Container Instance. 72–74
- ADR** Architecture Decision Records. 2
- ADR** Architectural Decision Record. 36, 45, 59, 121, 128, 129, 131, 135, 142
- AKC** Azure Kubernetes Cluster. 150
- AKS** Azure Kubernetes Service. 46, 72–74, 127–129, 139, 151
- API** Application Programming Interface. 18, 41, 43, 49, 50, 85, 100, 102, 107, 108, 111–113, 129, 141, 148, 157
- AVM** Azure Virtual Machine. 46, 72–74
- AZNG** Azure Virtual Network Gateway. 78, 79
- 
- CD** Continuous Delivery. 88, 90, 91
- CI** Continuous Integration. 88, 90, 91
- CMID** Client Machine ID. 132
- CORS** Cross-Origin Resource Sharing. 97
- CPU** Central Processing Unit. 17, 140, 150
- CRUD** Create, Read, Update and Delete. 49
- CSS** Cascading Style Sheets. 57, 84, 101, 110
- CTF** Capture The Flag. 6, 137
- 
- DinD** Docker-in-Docker. 88
- DNS** Domain Name System. 2, 6, 39, 45, 82, 83, 97, 121, 123, 126, 127, 129, 132, 149, 152, 153, 158
- DOM** Domain Object Model. 95
- DPL** Dynamic Pentest Lab. 1, 4, 36, 37, 39, 49, 50, 59, 60, 93, 105, 159

- 
- ECTS** European Credit Transfer System. 7
- GUI** Graphical User Interface. 62, 63, 92
- HL** Hacking-Lab. v, 5, 6, 10, 22, 24, 25, 30, 32, 34, 39, 41, 59–64, 68–70, 72, 73, 77, 79–81, 93, 94, 107, 110, 111, 119, 142, 145, 146, 148, 150, 152, 153, 155, 160
- HTML** HyperText Markup Language. 95, 98, 100, 102, 103, 107
- HTTP** Hypertext Transfer Protocol. 41, 97
- HTTPS** Hypertext Transfer Protocol Secure. 97
- ID** Identifier. 47, 48, 57, 109, 118
- IDE** Integrated Development Environment. 85
- IEC** International Electrotechnical Commission. 31
- IP** Internet Protocol. 126, 130, 140
- ISO** International Organization for Standardization. 31
- JDBC** Java Database Connectivity. 41
- JSON** JavaScript Object Notation. ii, v, 51, 52, 57, 58, 95, 96, 100, 102–106, 108–110, 112–114, 116, 117, 135, 148, 153–155, 158
- JWK** JSON Web Key. 81, 82, 120
- JWT** JSON Web Token. 49, 81, 120
- MVC** Model View Control. 94, 95
- NDA** Non-Disclosure Agreement. 145
- NFR** Non-Functional Requirement. 8, 31, 148, 150, 151
- NSG** Network Security Group. 46
- OS** Operating System. 21, 132, 134, 135, 150
- OST** Ostschweizerische Fachhochschule. ii
- P2S** Point-to-Site. 78
- Pentest** Penetration Testing. 1–4, 10, 36, 70, 73, 152, 155
- PoC** Proof of Concept. 79

- 
- RAM** Random-Access Memory. 140, 150
- REST** Representational State Transfer. 33, 120, 148
- RHEL** Red Hat Enterprise Linux. 150
- SKU** Stock Keeping Unit. 18, 28, 113, 114, 154
- SSL** Secure Socket Layer. 153
- SSO** Single Sign-On. 10, 148
- TCP** Transmission Control Protocol. 19
- TLS** Transport Layer Security. 4, 152, 153
- UDP** User Datagram Protocol. 19
- UI** User Interface. 39
- URI** Uniform Resource Identifier. 81, 82, 120
- URL** Uniform Resource Locator. 57, 109
- UUID** Universally Unique Identifier. 47, 115, 127, 149
- VM** Virtual Machine. ii, 1–3, 5, 6, 13–15, 18, 19, 21–24, 29, 36, 39, 46, 48, 50, 57, 66, 67, 71, 73, 74, 93, 105, 113, 114, 121, 122, 124–126, 128, 132, 133, 135–137, 148–151, 158
- VPN** Virtual Private Network. 2, 4, 6, 16, 19, 31, 36, 45, 78, 85, 93, 121, 128–131, 151–153, 158, 160
- YAML** YAML Ain't Markup Language. 140