Milestone 3: Practical Implementation of LowTech Gmbh Webshop in CSP Platform

Wladymir Alexander Brborich Herrera (1437876)
wladymir.brborich-herrera@stud.fra-uas.de,
Vishwaben Pareshbhai Kakadiya (1471845)
vishwaben.kakadiya@stud.fra-uas.de,
Hellyben Bhaveshkumar Shah (1476905)
hellyben.shah@stud.fra-uas.de,
Heer Rakeshkumar Vankawala (1449039)
heer.vankawala@stud.fra-uas.de, and
Priyanka Dilipbhai Vadiwala (1481466)
priyanka.vadiwala@stud.fra-uas.de

Frankfurt University of Applied Sciences
(1971-2014: Fachhochschule Frankfurt am Main)
Nibelungenplatz 1
D-60318 Frankfurt am Main

Abstract This report presents a summary of all the planification and execution of the LowTech GmbH Cloud Transformation Project. The final objective is to develop a proof of concept of the proposed infrastructure of a Cloud Native application, in this case, the Webshop. As detailed in milestone 2 the application will be hosted in Microsoft Azure. This report also discusses the advantages in system reliability, security, and scalability offered by the proposed approach. Moreover, we include a detailed description of the development techniques using tools like Terraform for infrastructure management and GitHub Actions for continuous integration and deployment (CI/CD). Special attention is given to the advantages in development time and repeatability of such tools and techniques. Furthermore, we evaluate the cost of the overall infrastructure identifying areas for optimization. The report also discusses future scalability strategies, ensuring that the application can handle different types of loads and it actually makes sense in a cloud environment Finally, the report provides a summary on the lessons learned and interesting findings while using the tools and techniques mentioned.

1 Introduction

LowTech GmbH, a medium-sized enterprise specializing in wooden furniture production, is modernizing its IT infrastructure as part of a comprehensive cloud transformation. Initially relying on traditional on-premises systems, the company faced challenges in scalability, security, and operational efficiency.

The transformation began with a thorough assessment of the existing infrastructure, which revealed the following key challenges:

- Limited scalability due to fixed hardware constraints.
- High operational costs and energy consumption of legacy systems.
- Outdated security measures, including basic firewall protection.
- Lack of automation, requiring manual interventions for maintenance and scaling.

To address these challenges, a private cloud migration strategy was adopted. The aim was to improve scalability, security, and cost-efficiency while ensuring minimal downtime and business continuity. The company transitioned its Webshop as a cloud native application to Microsoft Azure.

1.1 Overview of the Previous Milestones

1.1.1 Starting point LowTech GmbH, a small to medium-sized enterprise with 45 employees specializing in wooden furniture production, is facing a critical need for technological transformation. Initially relying on sales representatives, the company transitioned to an online store a few years ago, which has now become their primary selling platform. This shift has led to a significant increase in user numbers over the past two years, necessitating a modernization of their IT infrastructure. The CEO of LowTech GmbH, recognizing the need for modernization, has expressed interest in cloud computing to make the company future-ready.

1.1.2 Milestone 1: The objective of the technological transformation for LowTech GmbH is to modernize its server and application infrastructure by migrating to a private cloud environment. This initiative aims to create a scalable, secure, and efficient IT framework that aligns with the company's growing needs, particularly in response to increased online sales and user traffic. By adopting a private cloud strategy, LowTech GmbH will retain control over its data while outsourcing hardware maintenance to a third-party provider. This transition will involve leveraging newer technologies to enhance performance and availability, ensuring compliance with security standards, and reducing operational costs. Ultimately, the transformation seeks to provide a future-ready infrastructure that supports the company's evolving business model and operational requirements.

This was our assessment of the original infrastructure:

Scalability

- **Fixed Hardware & Inflexible Infrastructure**:
  Current infrastructure consists of 7 on-premises servers housed in a single 19-inch rack, along with 17 clients and 19 laptops all with predetermined, static configurations. Moreover, the physical constraints of the on-premises setup, with no additional space for expansion, severely limit scaling options. This inflexibility makes it challenging to accommodate growth or adapt to changing business needs.

- **Absence of Resource Utilization/pooling**:
  There's no apparent way to quickly scale resources up or down based on demand or user traffic fluctuation in the current infrastructure. Each application typically runs on a dedicated server with fixed resources. This approach leads to inefficient resource utilization, as some servers may be underutilized while others are overloaded which may lead to performance issues during peak times or resource waste during low-demand periods due to no dynamic resource allocation.

- **Manual Processes & High Cost**:
  Any changes in capacity would likely require manual hardware upgrades or replacements including hardware installation, and configuration, making the process time-consuming and potentially leading to downtime. Replacement of hardware is not only tedious but also financially burdensome due to high costs of new hardware.

Availability

- **Obsolete Hardware/OS & Runtime Environments**:
  Many components of the current infrastructure is based on very old hardware and outdated operating systems such as Windows XP SP3 (Finance clients), Windows 7 SP3 (HR clients and Customer Service laptops), Debian 5.0 Lenny (Warehouse clients and server), Ubuntu 16.04 LTS (Sales CRM Storage server) etc. Several applications are running on outdated software versions such as Java 1.7/1.8, MySQL 5.5/5.7, PHP 5.3 and Firefox 3.6 etc. which makes this whole infrastructure more susceptible to failure.

- **Lack of Redundancy and Backup Mechanism**:
  There's no mention of redundant systems or data backup solutions which might lead to significant service disruptions as well as data loss in case of any system failure.

- **Manual Maintainance**:
  It is impossible to meet high availability requirements without a robust failover mechanism due to manual maintenance operations. This increases the possiblity of human errors, leads to longer downtime and reduces overall reliability.

Security

- **Basic Windows Firewall**:
  It provides minimal outward traffic control, which may allow malware to interact easily. Without additional tools, centralized administration is impossible to maintain consistent security rules across all 36 client devices (17 clients and 19 laptops). It lacks advanced threat protection features, leaving the system vulnerable to sophisticated attacks.

- **pfSense for Network Packet Filtering**:
  There are no built-in antivirus features, which could let malicious payloads get through. Its effectiveness heavily relies on proper configuration, which may be challenging without dedicated IT staff. Moreover, advanced intrusion detection requires additional setup and maintenance.

As detailed in the analysis the infrastructure was becoming quite expensive and inefficient given the company objectives.

Plan and execution based on the original requirements: This milestone resulted in a migration and optimization plan for the entire infrastructure:

- Talk with server space providers, we need to stablish a contract and negotiate price and SLOs based on the server requirements
- Gather all the installation information, binaries, licenses and documentation for all applications.
- Configure a development environment to upload all artifacts such as configuration files and container images.
- Create Ansible configuration files for each application, to automate the installation and replication process.
- Develop scripts to automate application functionality and load testing. To determine if the configurations will be on par or better than the current infrastructure.
- Provision and install the private cloud management software, including the networking configuration
- Configure the storage
- Create the virtual machines for the base hosts
- Configure Prometheus to monitor the virtual machine installations
- Establish the network links between the different virtual domains
- Install all database servers
- Develop a migration process to replicate the current data into the new database servers.
- Review that data is up to parity with the legacy services
- Create a proxy getaway for the services, so we can redirect the traffic from the old services to the new ones.
- Deploy the new applications into the private cloud
- Check that the data replication is working
- Prepare the load shifting in sequence, and off hours
- Shift load in sequence, starting from the most isolated applications first, then the ones with the most dependencies.
  - Migrate finance and HR, since they are self-contained

- Migrate operations
- Migrate customer service
- Migrate warehouse
- Migrate webshop
- Migrate sales

Optimization

- Measure the performance of the system using Prometheus metrics and create an assessment report of the migration.
- Create documentation regarding the new deployment process, provisioning and scaling.
- Assess potential improvement areas, and stablish follow up tasks if needed.
- Once we have the virtualized applications, we could start thinking into improving elasticity by enabling either ProxMox autoscaling (Which provides additional resources to VMs on the fly) or install an orchestrator that will create new virtual machines and load balance them.
- During optimization we also need to allocate time for training, and documenting the different processes such as: how to scale, deploy and provision new services.

1.1.3 Milestone 2 Once the client for the project decided to actually migrate to a public cloud service provider, we developed a new strategy. Given that the Webshop in particular was going to be developed from scratch following modern approaches with new technologies.
We proposed a new standard for this kinds of applications:

Webshop Infrastructure, Tools, and Services: This application will be newly developed following the standard for new applications. As a special consideration, this application is the only public-facing application that needs to handle customer requests. Our standard for newly developed applications enables this critical piece of the business to easily scale and be secured with valid certificates with no extra configurations.

Cloud Context	Products And Technologies	Service Models
Public Cloud	Azure App Service	PaaS or CaaS
Public Cloud	Azure Static Web App	PaaS or CaaS
Public Cloud	Azure Database For PostgreSQL	PaaS

Table 1: Webshop: Website Deployment Strategy

General Considerations: Importantly, for all newly developed applications, we will:

- Set up a private GitHub repository
- Set up GitHub actions to enable continuous integration and continuous deployments in Azure
- Set up an infrastructure folder to hold terraform files. These files will define the infrastructure and will allow for easy repeatability
- Use Microsoft Entra ID in case of authentication with the organization is needed
- Use Docker to package the application with containers that will be deployed in Azure App Service in case of the backend, and Azure Static Web Apps in case of the frontend

Standard For a Cloud Native Application: When developing a cloud-native application Figure 1 shows the basic setup to integrate DevOps concepts into our workflow. The website is going to be developed under this standard.

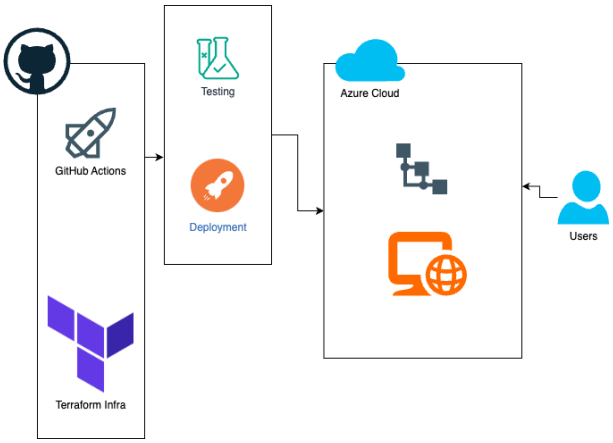

Figure 1: Standard Configuration To Deploy An Application

1.2 Objectives of the Cloud Implementation of Webshop

The Webshop is a critical component for LowTech GmbH, serving as the company's primary sales platform. The goal of its cloud implementation is to enhance performance, scalability, and security, ensuring a seamless user experience.

Key objectives for the Webshop's cloud-based deployment include:

- **Scalability and Performance Optimization:** The Webshop is deployed on Azure App Service with auto-scaling capabilities, enabling efficient traffic handling during peak periods. A load balancer ensures even traffic distribution across multiple instances.
- **High Availability and Reliability:** Azure Virtual Machine Scale Sets provide fault tolerance with automatic failover. Azure Blob Storage is used to securely store digital assets with high availability.
- **Security and Compliance:** The Webshop integrates with Microsoft Entra ID for user authentication and Azure Security Center for enhanced threat protection. Encryption and Role-Based Access Control (RBAC) are employed to safeguard sensitive customer data.
- **Continuous Deployment and DevOps Automation:** A CI/CD pipeline powered by GitHub Actions automates code deployments, improving deployment speed and reducing manual intervention.
- **Cost Efficiency and Resource Optimization:** The dynamic allocation of resources optimizes compute and storage usage, reducing operational expenses. Azure's pay-as-you-go model aids in cost forecasting and budget management.
- **Future-Proofing and Cloud-Native Development:** The Webshop follows cloud-native best practices, utilizing Docker containers for portability and Azure Kubernetes Service (AKS) for container orchestration, positioning the Webshop for seamless integration with future cloud services.

These objectives ensure that the Webshop is scalable, secure, and cost-effective, delivering a fast and reliable shopping experience while adapting to the evolving needs of the business.

2 Application Design

2.1 Architectural Overview

Detailed description of the three-tier structure with CSP service mapping

2.1.1 Presentation-Tier (Frontend) - User Interface (UI)

Technology Stack

- Frontend Framework: React.js (JavaScript)
- State Management: React Context API
- Hosting & Deployment: Azure Static Web Apps

component-based architecture

1. Navigation & Routing Users can easily navigate between different sections of application, such as viewing product lists, accessing product details, and managing their shopping cart. React Router enables seamless client-side navigation, keeping your app as a single-page application (SPA).
   - React Router Setup
     - React Router (react-router-dom) to handle the routing of different components, allowing users to navigate through pages like the homepage (/), product detail pages (/product/:id), and potentially a shopping cart or checkout page.
     - For product detail pages, you're using dynamic routes with product/:id to fetch and display specific product data based on the product's unique id. For instance:
     - When a user clicks on a product in the catalog, the URL changes to something like /product/123, and the ProductDetail component is rendered with data for product 123.
     - This is achieved using useNavigate and useParams hooks provided by React Router to capture the dynamic part of the URL.
   - Navigation Links
     - On the homepage (Home component), displaying a list of products. Each product has an image and name that users can click. When clicked, the app navigates to the product detail page using the navigate(/product/$prod.id) function.

2. API Communication
   - API communication in app is responsible for sending and receiving data from the backend server. This includes fetching product data to populate catalog, handling cart actions (adding or removing items), and processing payments during checkout.
   - The app communicates with the backend to update the cart whenever an item is added or removed. When a user clicks "Add to Cart" or "Remove from Cart," an API call might be made to update the user's cart data on the server. This interaction is handled by `CartContext`, which uses `dispatch` to update the cart state and might also send a request to the backend to keep the cart in sync.
   - For payment, we use PayPal to securely handle credit card transactions. The app makes API calls to PayPal's backend to process the payment once the user submits their payment details.

3. Data Fetching
   Data fetching refers to the process of retrieving data from external sources (your backend API) to populate app with dynamic information, such as product details, cart contents, and order history.
   - The product data, which includes the list of products with details such as images, names, prices, and categories, is fetched when the homepage or product catalog page loads. This is done using useEffect to trigger an API call and update the state with the fetched data.

Key Features of the UI

1. UI Components & Design
   The UI components in your app are designed to provide an intuitive and engaging shopping experience. Each component serves a specific role, ensuring modularity, reusability, and maintainability.
   - Key Features
     - Reusable Components: Components like SingleProduct.js, ProductDetail.js, and Cart.js ensure a structured and modular UI.

- Bootstrap Integration: react-bootstrap is used to style UI components such as Card, Button, and Container, ensuring a consistent and responsive design.
- Dark & Light Theme Support: A theme context (ThemeContextProvider) dynamically adjusts UI styles based on user preference.

2. Product Display & Interaction
   The UI effectively displays products with details, allowing users to browse, select, and view product specifications before making a purchase.
   – Key Features
   - Product Catalog: Displays a grid of available products with images, names, and prices.
   - Product Cards (SingleProduct.js):
     * Clickable product images navigate to the product details page.
     * Shows product information such as name, category, description, and price.
     * "Add to Cart" and "Remove from Cart" buttons enable quick cart management.
   - Product Details Page (ProductDetail.js):
     * Displays full product details, including stock availability.
     * Users can add/remove items from the cart.

3. Shopping Cart UI & Checkout
   The cart and checkout sections provide an intuitive way for users to review their selected products, manage quantities, and complete purchases.
   – Key Features
   - Cart UI (Cart.js)
     * Displays a list of selected products with prices and a "Remove from Cart" button.
     * Updates total price dynamically based on the cart's contents.
     * Navigates users to checkout when ready.
   - Checkout UI (CheckoutForm.js)
     * Collects user details (name, email).
     * Integrates Stripe and PayPal for secure payment processing.
     * Uses emailjs to send order confirmation emails.
     * Displays success messages with reference numbers for placed orders.

4. Notifications
   Notifications is crucial to enhancing the shopping experience. The app uses toast notifications and error messages to keep users informed.
   – Key Features
   - Toast Notifications (react-toastify)
     * Displays success messages when products are added/removed from the cart.
     * Shows error messages if something goes wrong (e.g., out-of-stock products, payment failures).

2.1.2 Application-Tier (Backend) - Business Logic

Technology Stack

– FastAPI
– SQLAlchemy with PostgreSQL
– Asyncio

Product Management

The Product Management module handles the lifecycle of products in the system. This includes operations for adding new products, updating product details, and retrieving product information. The core business logic ensures that products are categorized correctly, their stock levels are managed, and prices are updated as needed.

- Key Operations:
  - Add, update, and delete products.
  - Manage product details such as descriptions, prices, and categories.
  - Track stock availability and reorder levels.
  - Calculate the public unit price and manage suppliers' pricing.

Order Management

The Order Management module manages customer orders, from order creation to order completion. It tracks the order status and ensures that orders are processed correctly, including payment validation, stock management, and shipping.

- Key Operations:
  - Create and update orders with customer and product information.
  - Monitor order statuses (e.g., processing, dispatched, delivered).
  - Validate inventory and ensure product availability during order processing.
  - Handle refunds, cancellations, and partial shipments.

Payment Processing

The Payment Processing module integrates with third-party payment gateways (e.g., Stripe) to securely process customer payments. This module validates payment details, checks for fraud, and ensures payment is successfully processed before confirming orders.

- Key Operations:
  - Process payments securely through APIs like Stripe and PayPal.
  - Validate payment information (e.g., credit card details).
  - Handle refunds, cancellations, and partial shipments.

Inventery Management

The Inventory Management module tracks the stock levels of all products, ensuring that inventory is updated in real-time based on orders placed and products received from suppliers. It also tracks reorder levels and alerts the system to restock low inventory.

- Key Operations:
  - Monitor and update product stock levels after each sale.
  - Track the supplier's stock and delivery lead times.
  - Generate reports on product availability, low-stock items, and reorder recommendations.

Email Notification

The Email Notification module sends emails to customers and administrators for various events in the system, such as order confirmations, shipment tracking updates, and payment status notifications. It integrates with FastMail to send HTML-formatted emails with dynamic content.

- Key Operations:
  - Send order confirmation emails to customers.
  - Notify customers of order status updates, such as dispatched or delivered.
  - Alert administrators of low stock levels and other system alerts.
  - Provide a user-friendly email template system for various types of notifications.

2.1.3 Data-Tier (Database) - Databases

Technology Stack

- PostgreSQL
- SQLAlchemy ORM
- Asyncio & Asynchronous SQLAlchemy
- Alembic
- Azure Database for PostgreSQL

Data bases

1. Product Data
   - Product
     - Stores the core product information and links to categories, suppliers, and inventory.
     - Key Attributes:
       - id: Unique product identifier (Primary Key).
       - name: Product name.
       - description: Product description.
       - category_id: Foreign Key linking to the Categories table.
       - supplier_id: Foreign Key linking to the Suppliers table.
       - stock_id: Links to the Stock table for inventory tracking.
       - public_unit_price: Price displayed to customers.
       - supplier_unit_price: Price from the supplier (for internal calculations).
       - ean_code: Unique product code for identification.
       - reorder_level: Threshold for restocking.
       - img_link: URL to the product image.
       - extra_info: JSON field to store additional product attributes.
   - Catagories
     - This table categorizes products for better organization and searchability.
     - Key Attributes:
       - id: Unique category identifier (Primary Key).
       - name: Product name.
       - description: Category description.
       - extra_info: JSON field for additional metadata.
2. Order Data
   - Order
     - This table records general order details and customer information.
     - Key Attributes:
       - id: Unique order identifier (Primary Key).
       - customer_name: Name of the customer.
       - customer_email: Customer's email address for notifications.
       - customer_phone: Contact number.
       - order_total: Total value of the order.
       - status: Enum representing the order state (e.g., "active", "cancelled").
       - tracking_status: Enum tracking delivery progress (e.g., "dispatched", "delivered").
       - payment_method: JSON field storing payment details (e.g., card type).
       - customer_shipping_info: JSON field for customer delivery address.
       - created_at: Timestamp when the order was created.
       - updated_at: Timestamp when the order was last updated.
   - Order Details
     - This table provides itemized details of each product within an order.
     - Key Attributes:
       - id: Unique identifier for each order detail record (Primary Key).

    * order_id: Foreign Key linking to the Orders table.
    * product_id: Foreign Key linking to the Products table.
    * quantity: Number of units of the product ordered.
    * product_price: Unit price at the time of the order.
    * subtotal: Calculated subtotal for the item.

3. Inventory Data

   – Stocks
     • Monitors the available quantity of each product and manages restocking.
     • Key Attributes:
       * id: Unique stock identifier (Primary Key).
       * quantity: Number of available units for a product.
       * updated_at: Timestamp of the last stock update.
       * created_at: Timestamp when the stock record was created.

   – suppliers
     • Manages supplier-related information and links to products for procurement.
     • Key Attributes:
       * id: Unique supplier identifier (Primary Key).
       * name: Supplier name.
       * address: Supplier's physical address.
       * phone: Contact number.
       * email: Contact email address.

4. Database Migrations with Alembic

   – It allows us to track database changes, apply updates to production, and roll back changes when needed.
   – Key Operations with Alembic:
     • *Schema Migration:* Automatically generates migration scripts to apply changes to the database (e.g., adding new fields to a table).
     • *Version Control:* Each migration is versioned, allowing us to track and audit changes.
     • *Rollback Support:* Provides the ability to downgrade to previous schema versions if issues arise.

2.2 System Diagrams

3 Implementation Process

3.1 Cloud Environment Setup

Step-by-step account configuration and resource provisioning

3.2 Service Integration

– Azure Load Balancer configuration
– Database replication setup

3.3 Development Challenges

– State management in scaled environments
– Database connection pooling
– CSP-specific limitations encountered

4 Operational Characteristics

4.1 Performance Metrics

4.1.1 Functional Test Cases

Test Case ID	Test Scenario	Expected Outcome	Status (Pass/Fail)
TC-001	Load homepage and verify product listing	Homepage loads with products displayed correctly.	
TC-002	Apply price filter (Ascending/Descending)	Products reorder correctly based on selected price.	
TC-003	Apply Öut of Stockffilter	Only out-of-stock items are displayed.	
TC-004	Apply FFast Deliveryffilter	Only products eligible for fast delivery show up.	
TC-005	Filter by category	Products are filtered correctly by selected category.	
TC-006	Search product by name or category	Products matching search are shown correctly.	
TC-007	Clear filter functionality	All filters are removed, showing the full product list.	
TC-008	Product Detail Page - Click Product	Clicking a product opens its detailed page.	
TC-009	Product Detail Page - Load Product Details	Product details (name, description, price) are displayed.	
TC-010	Add a product to cart	Product appears in cart with correct details.	
TC-011	Increase product quantity in cart	Quantity updates and is reflected in the cart.	
TC-012	Remove product from cart	Product is removed from the cart immediately.	
TC-013	Proceed to checkout	Checkout page loads with the correct order summary.	
TC-014	Select payment method - Stripe	Stripe payment option is selected and processed.	
TC-015	Select payment method - PayPal	PayPal payment option is selected and processed.	
TC-016	Complete order processing	Order confirmation message is displayed.	
TC-017	Order confirmation email is received	Email is sent after order is placed.	
TC-018	Shipment notification email is received	Email is sent when the order is shipped.	
TC-019	Toggle Dark/Light Theme	Application switches between themes successfully.	

4.2 Security Considerations

- Network security groups configuration
- Database encryption implementation
- Access control mechanisms

5 Critical Analysis

5.1 Cloud Service Evaluation

Cost-benefit analysis of selected Azure services

5.2 Architectural Decisions

Trade-off discussion between containerized vs serverless approaches

6 Repository Documentation

6.1 GitHub Structure

1. **Branching Strategy:**
   The project follows a simple main branch strategy where all contributors work directly on the main branch. This ensures seamless integration without managing multiple branches. All changes should be committed with meaningful messages. Code should be reviewed and tested before pushing to the main branch.

2. **Continuous Integration/Continuous Deployment:**
   - **Continuous Integration (CI):**
     Every code commit triggers an automated build and testing process. This includes:
     - Running unit tests to validate individual components.
     - Performing integration tests to ensure compatibility between different modules.
     - Static code analysis for linting and security vulnerabilities.
   - **Continuous Deployment (CD):**
     Once the CI stage passes successfully, the application is automatically deployed to the appropriate environment. This process includes:
     - Deploying to a staging environment for final testing.
     - Running automated acceptance tests before production deployment.
     - Deploying to production with rollback mechanisms in case of failure.
   - **Documentation Standards**

6.2 Contribution Tracking

Commit history analysis and individual contribution breakdown

7 Conclusion

7.1 Project Outcomes

Summary of achieved objectives and demo capabilities

7.2 Future Enhancements

Potential improvements for production readiness

References