Presentify: Automating Deployments with Docker and Azure

Abstract :

Presentify is a full-stack web application that leverages AI to automate the generation of presentation outlines. The project highlights the use of a robust DevOps setup, incorporating Docker for containerization and GitHub Actions for continuous integration and continuous delivery (CI/CD). This setup ensures the seamless deployment of the application to Azure Virtual Machines. The core technologies utilized in the project include Bun, Prisma, and Next.js, which power the web application's functionality. The objective of this project is to establish a scalable, consistent, and secure deployment pipeline, ensuring high availability and reliability in production environments.

Index Terms :
Docker, DevOps, Bun, GitHub Actions, Azure, Prisma, Containerization, CI/CD, Presentify

1. Introduction

Presentify addresses the common challenge of inconsistent application behavior across different environments by leveraging containerization. By utilizing Docker and Bun (a modern JavaScript runtime), the project ensures efficient development and reliable deployment. This report provides a comprehensive overview of the application's lifecycle, from local development to its production deployment on Azure Virtual Machines, highlighting the importance of Docker in maintaining consistency across various stages.

2. Methodology

The architecture of **Presentify** integrates a Bun-powered backend with Prisma ORM, an AI module utilizing Google GenAI, and a frontend styled with Tailwind CSS. Each component is containerized using **Docker** to ensure consistency across all environments, enabling smooth development and deployment.

1. Backend with Bun and Prisma ORM

The backend is powered by **Bun**, a high-performance JavaScript runtime, chosen for its speed and efficiency. **Prisma ORM** is used for type-safe database interactions, simplifying data queries and migrations. This ensures clean and consistent access to the database throughout the application.

2. AI Module with Google GenAI

The **AI module** leverages **Google GenAI** to automate the generation of presentation outlines. It processes user inputs and delivers relevant content, minimizing manual effort in creating slides. The module operates independently, communicating with the backend for seamless integration.

3. **Frontend with Tailwind CSS**

The frontend is built with **Tailwind CSS**, a utility-first CSS framework that allows for rapid UI development and ensures a consistent, responsive design. This approach reduces the need for custom CSS and promotes reusable components, enhancing both the speed and maintainability of the UI.

4. **Containerization with Docker**
All components, including the backend, AI module, and frontend, are containerized using **Docker**. This ensures that the application runs consistently across all environments, solving the common problem of "it works on my machine." A specialized **Dockerfile** for Bun packages the backend and dependencies, providing an isolated and reproducible environment for development and production.

5. **Component Schema for UI Consistency**
To maintain UI consistency, a **components.json** file is used to define the structure of UI elements. This schema standardizes the design of components, ensuring that they follow a consistent pattern throughout the application.

6. **Development Workflow with tmux and Shell Scripts**
Developers can use a **shell script (dev-app.sh)** to easily start the application and launch **Prisma Studio**. Additionally, **tmux**, a terminal multiplexer, allows developers to manage multiple terminal sessions within a single window, improving workflow and productivity.

3. DevOps Pipeline Implementation

To ensure continuous integration and deployment, Presentify leverages GitHub Actions to implement an automated DevOps pipeline. This pipeline is defined through two key YAML workflow files located in the `.github/workflows/` directory: `build.yml` and `deploy.yml`. The `build.yml` workflow is triggered on every push to the main branch. It first checks out the latest source code and proceeds to build Docker images tailored for the Bun-based backend and the Tailwind CSS-styled frontend. The pipeline is configured for multi-platform support using Docker Buildx to ensure broader deployment compatibility.
All secrets and environment variables—such as the Google GenAI API key, database connection strings, and authentication tokens—are securely stored using GitHub Secrets. These secrets are dynamically injected during the workflow execution, ensuring they are never exposed in the codebase or logs. A temporary `.env` file is generated inside the container environment to hold these variables during the build process.

Once the Docker images are built, they are pushed to Docker Hub using credentials stored in GitHub Secrets. This allows the latest application state to be accessed from a centralized registry. The `deploy.yml` file handles the deployment phase by establishing a secure SSH connection to an Azure Virtual Machine. From there, it pulls the updated Docker image and orchestrates the container lifecycle—stopping the previous container, removing it, and spinning up the new one—ensuring the latest version is always running in production.

This structured pipeline not only automates the CI/CD flow but also enforces best practices in secret management, infrastructure control, and deployment consistency across environments.

IV. GOALS

1. **Automate the deployment pipeline for full-stack Bun applications:**
To streamline and accelerate the software delivery process, Presentify adopts a fully automated deployment pipeline for its Bun-powered full-stack architecture. This ensures that code changes are continuously integrated, tested, and deployed with minimal manual intervention. By leveraging

GitHub Actions, the pipeline automatically builds Docker images, runs essential scripts, and deploys the updated application to production environments, significantly reducing deployment time and human errors.

2. **Maintain consistency across environments using Docker:**
A critical goal is to ensure that the application behaves identically in development, staging, and production environments. This is achieved by containerizing the entire application—backend, frontend, and dependencies—using Docker. Docker's encapsulation of the environment eliminates common configuration mismatches, making the app more reliable and easier to debug across various stages of the development lifecycle.

3. **Securely manage secrets using GitHub Secrets:**
Security is a foundational concern. Presentify avoids hardcoding sensitive data such as API keys, database URLs, and cloud credentials by managing them through GitHub Secrets. These are injected securely into the GitHub Actions workflow during runtime, protecting confidential information and ensuring the integrity of the CI/CD process.

4. **Simplify developer onboarding with preconfigured tmux sessions:**
To enhance developer experience and streamline local setup, the project includes a shell script (`dev-app.sh`) that launches preconfigured `tmux` sessions. These sessions open the Bun development server and Prisma Studio in split terminals, providing a ready-to-go development environment. This significantly reduces the onboarding time for new contributors by abstracting away complex environment setup steps.

5. **Utilize Azure Virtual Machines for production-grade deployment:**
For hosting the production environment, Presentify deploys its containerized application on Azure Virtual Machines. This approach provides greater control over infrastructure, scalability, and resource allocation. The deployment process involves secure SSH access to the VM, followed by orchestrating Docker containers to ensure smooth updates and high availability in the live environment.

5. **Docker and Deployment Architecture**
Presentify adopts a containerized architecture to ensure consistency, modularity, and streamlined deployment. At its core, the application follows a simplified microservices structure where the full-stack Bun application is containerized using Docker. The primary service, which includes both backend and frontend logic, is defined through a custom `Dockerfile` that uses Bun to install dependencies, generate Prisma clients, build the project, and run the application in development or production mode based on the deployment context.

To optimize build times and image size, `.dockerignore` and `.gitignore` files are used to exclude unnecessary files such as node modules, logs, and other development artifacts. This ensures a clean and minimal Docker image.

All containers are deployed within a user-defined Docker network, allowing internal services to communicate using container names as hostnames. This eliminates the need for hardcoded IP addresses and simplifies cross-service networking during local development and on production servers. For example, if the architecture is extended to support a separate database or caching service, those containers can seamlessly interact using predefined network aliases.

Environment-specific secrets such as database URIs, JWT tokens, and API keys are not embedded within the image. Instead, they are securely injected at runtime using GitHub Secrets during the CI/CD pipeline execution. This results in a dynamically created `.env` file used solely for runtime configuration, ensuring credentials remain protected.

In production, the Docker image is pulled onto an Azure Virtual Machine. The deployment script connects to the VM via SSH, gracefully stops any running containers, and spins up a fresh instance of the application container, exposing it on port 3000. This process guarantees zero-downtime updates and reliable infrastructure management. Though the current deployment includes a monolithic service structure, the architecture is designed with modularity in mind, making it easy to scale toward a more granular microservice model if needed.

6. **RESULTS AND OBSERVATIONS**

The implementation of the CI/CD pipeline using GitHub Actions for Presentify has consistently delivered reliable and efficient results. Each push or pull request to the repository's main branch automatically triggers the pipeline, initiating the complete cycle of building Docker images, securely injecting environment variables, and deploying to the Azure Virtual Machine.

On average, the pipeline completes the entire build and deployment process within a few minutes. Docker images are built using Bun, tagged, and pushed to Docker Hub through the `build.yml` workflow. The `deploy.yml` workflow then connects to the Azure VM via SSH, gracefully stops any existing container, pulls the latest image, and restarts the application container, all with minimal downtime.

Real-time logs from GitHub Actions provide transparency into each step of the pipeline, aiding in quick identification and resolution of errors. Azure's live feedback and status monitoring tools complement this by offering additional observability post-deployment. During multiple test deployments, the pipeline exhibited high resilience and stability, consistently deploying new updates without failures or unexpected behavior.

Secrets such as the database URI, JWT tokens, and other sensitive credentials are managed through GitHub Secrets, ensuring they remain protected and are never exposed in logs or code. The dynamic creation of the `.env` file during the workflow ensures that only secure, runtime-accessible values are used, adhering to best practices in DevOps security.

Overall, the integration of GitHub Actions, Docker, and Azure VM results in a highly repeatable, scalable, and secure deployment process. The system not only minimizes manual intervention but also enables faster release cycles and reliable application uptime, significantly enhancing the developer experience and deployment confidence.

7 . **CONCLUSION**

Presentify showcases a secure, automated, and scalable deployment strategy by leveraging modern DevOps practices and open-source tools. The integration of Docker for containerization ensures that the application performs consistently across development, staging, and production environments. This uniformity eliminates environment-specific issues and enhances reliability during deployment.

The adoption of GitHub Actions for continuous integration and deployment significantly accelerates the development lifecycle. Automated workflows handle every step—from building and testing the application with Bun, generating the `.env` file securely using GitHub Secrets, to pushing Docker

images and deploying them on Azure. This hands-off pipeline not only minimizes manual effort but also ensures that the application is always in a production-ready state.

Azure Virtual Machines provide a robust hosting solution, pulling Docker images directly from the registry and running them with minimal downtime. Logs and feedback from GitHub Actions and Azure offer strong visibility, aiding in fast debugging and stable monitoring post-deployment. Secrets management remains a top priority, with no credentials hard-coded or exposed during the deployment process.

This project successfully implements a modern DevOps pipeline suited for full-stack applications. Going forward, it can be enhanced by introducing automated deployment triggers, real-time performance monitoring, and infrastructure-as-code tools like Terraform for repeatable, scalable infrastructure provisioning. Incorporating automated E2E testing and container orchestration tools like Kubernetes could further boost resilience and scalability.

In summary, Presentify not only meets its deployment and automation goals but also serves as a future-ready template for secure, efficient, and scalable application deployment workflows.

Acknowledgments
This project was made possible by the extensive documentation and support provided by the open-source communities behind Docker, GitHub Actions, Bun, Prisma, and Azure. Their robust tooling and best practices enabled the creation of a secure and fully automated DevOps pipeline. Special thanks to **Google GenAI** for enhancing the AI-powered features, and **ShadCN** for providing modern UI components that significantly improved the user interface and developer experience.

The development process also benefited from publicly shared CI/CD workflows, forums, and the collaborative spirit of the software development community. Without these freely available resources, building a scalable, maintainable, and efficient deployment infrastructure would not have been achievable.

References
[1] Docker Documentation: https://docs.docker.com/
[2] GitHub Actions Docs: https://docs.github.com/en/actions
[3] Bun Documentation: https://bun.sh/docs
[4] Prisma Documentation: https://www.prisma.io/docs
[5] Microsoft Azure Docs: https://learn.microsoft.com/en-us/azure/
[6] Google GenAI: https://deepmind.google/technologies/gemini
[7] ShadCN UI Components: https://ui.shadcn.dev/

Project Link:
https://github.com/SumitGarg11/Presentify