TresVista
Catalyzing Your Impact

Platform Engineering and DevOps

Case Studies

Platform Engineering and DevOps

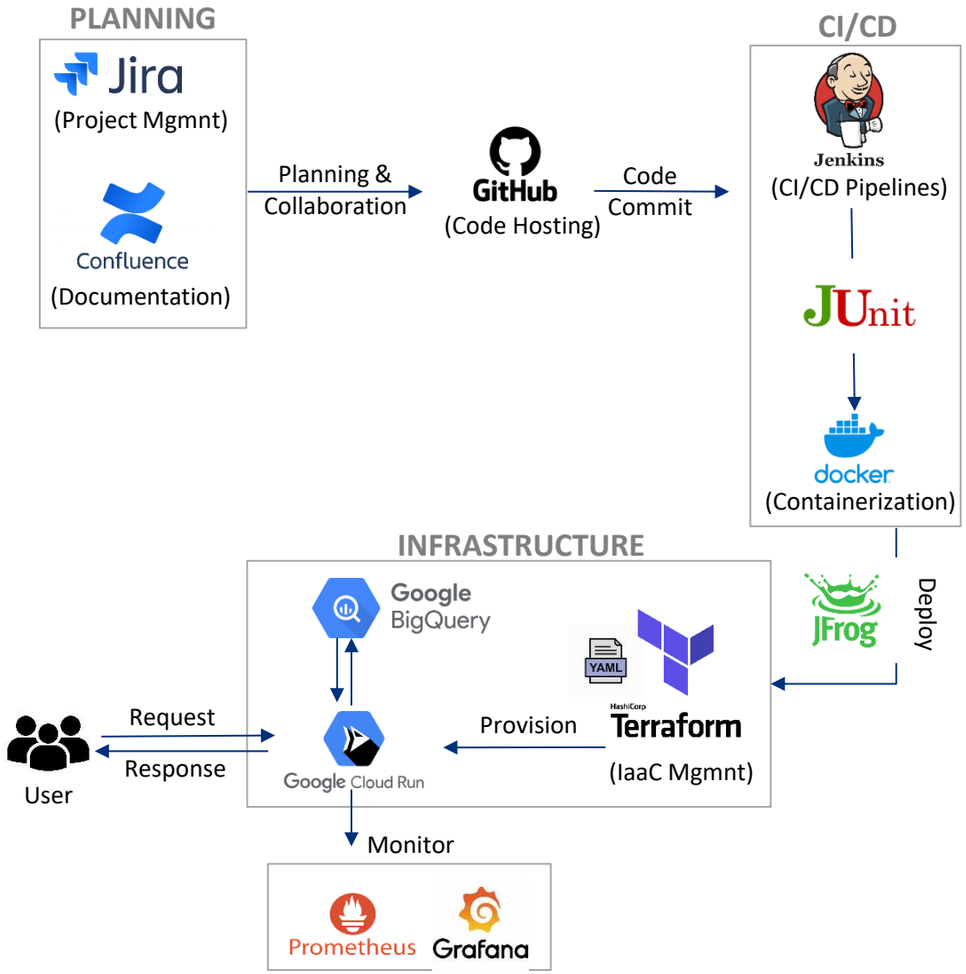
DevOps (CI/CD)

Request and Guidelines Provided

- An Investment Banking firm, managing a set of applications, wanted to create a process to perform frequent releases
- They faced major challenges with the diverse IT environments, manual deployment processes, slow releases of their software development processes, and complying with the industry regulations
- The firm wanted to adopt DevOps practices, with a strong focus on CI/CD, to address these challenges

Methodology and Final Deliverable

- Continuous Integration was implemented using Jenkins that automated the building and testing process (like unit tests and end-to-end tests) thereby validating code changes early and reducing the risk of bugs
- Docker was used to containerize applications, ensuring that they ran consistently across development, testing, and production environments. Implemented stage gates to ensure that that only thoroughly tested and approved changes were released, maintaining high standards of quality and compliance
- Jenkins pipelines were configured to automate the deployment of Docker containers to various environments with version control functionalities. A structured release planning process ensured continuous improvement and timely delivery of new features
- Furthermore, Terraform was used to automate the provisioning and management of infrastructure on cloud services like Good Cloud Platform (GCP)
- Monitoring was achieved by using tools like Prometheus and Grafana to collect, store, and visualize metrics, enabling real-time tracking and analysis of system performance and health.



The IB firm not only improved software quality and deployment speed, but also fostered collaboration and continuous improvement

Platform Engineering and DevOps

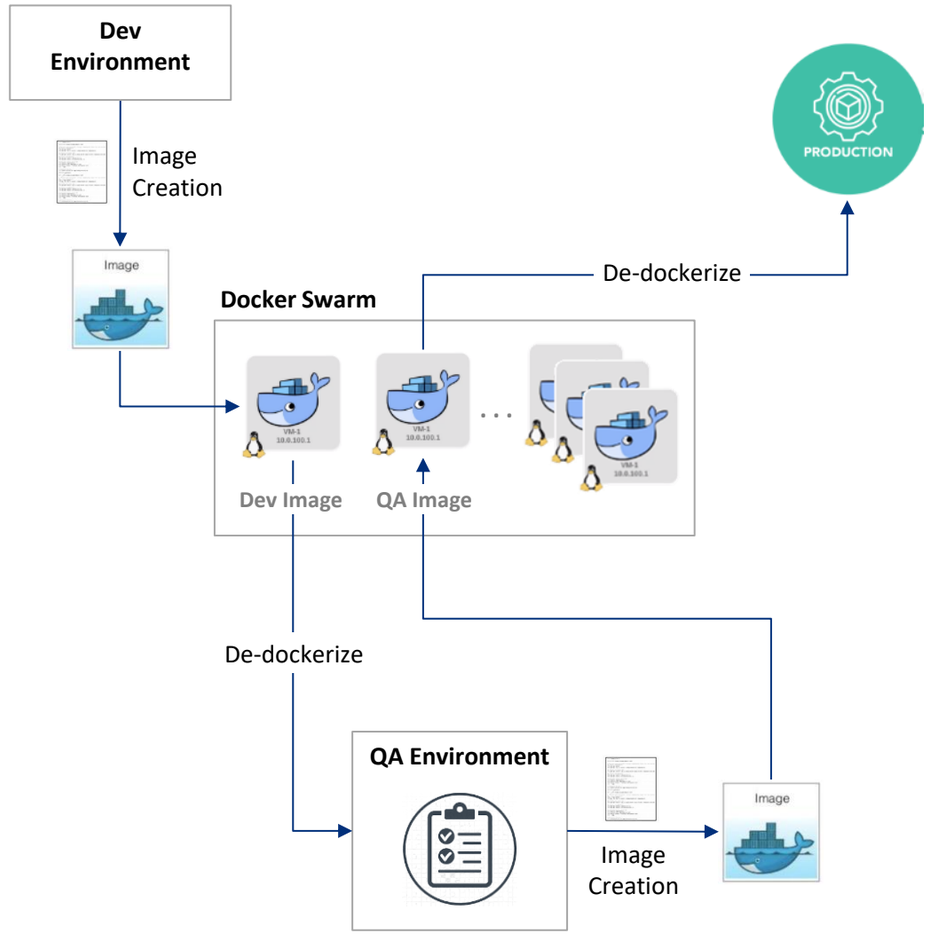
Application Development using Docker

Request and Guidelines Provided

- A leading secondaries firm, aimed to develop a robust application that could track different accounts and transactions, that could operate and integrate seamlessly across testing and development environments
- We addressed the challenges faced by them due to different environments that led to issues with configurations and discrepancies between development, QA, and production environments

Methodology and Final Deliverable

- **Containerization with Docker:**
  - **Application Packaging:** Ensure consistent performance and configurations across environments by packaging applications and dependencies such as libraries and packages into Docker containers, saving configuration time
- **Docker Swarm:**
  - **Orchestration:** Utilized Docker Swarm for orchestrating and managing clusters of Docker images
  - **Scalability:** Enabled easy scaling of services across multiple nodes
  - **Docker Compose:** Orchestrated multi container applications
  - **Repository:** The images can be reused by multiple teams for future projects
- **Cost-Saving:** We were able to recreate the infrastructure even with inferior dev and QA infrastructure
- **Time-Saving:** A significant amount of time was saved in recreating and/or reconfiguring the environments
- **Environment Recreation:** Dev. environment was dockerized and loaded into Docker Swarm, from which QA env was created. Similarly, QA environment was dockerized post necessary enrichments, from which the Prod. environment Was created, resulting in high efficiency



By leveraging Docker, the firm could achieve a consistent, scalable, and efficient application management and deployment system

Platform Engineering and DevOps

Cross Platform Integration (Slide 1 of 2)

Request and Guidelines Provided

- A financial services company needs to integrate data from two other cloud-based platforms into a unified application hosted on Microsoft Azure so that cross-service integration can be performed
- The goal is to ensure seamless data processing while maintaining high-security standards to protect sensitive financial information while adhering to platform guidelines at a lower cost and higher efficiency

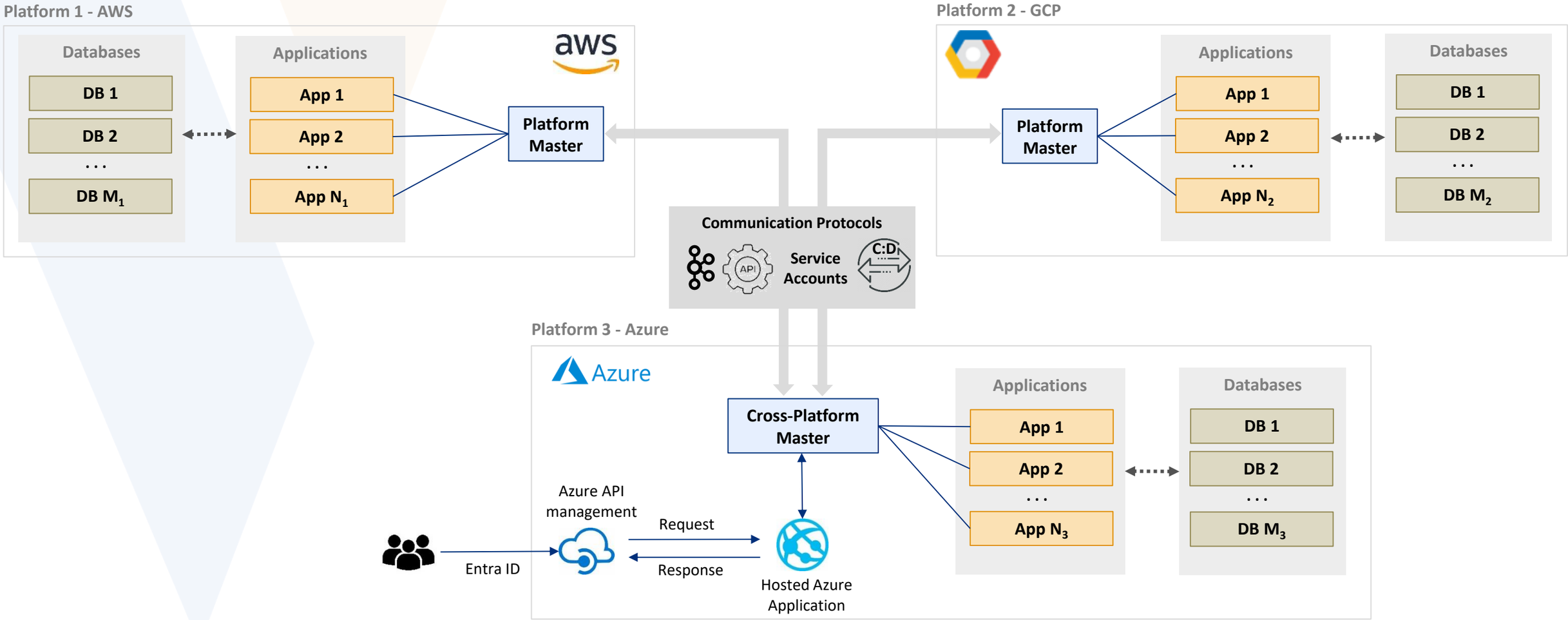
Methodology and Final Deliverable

- **Master-Slave Architecture** with an overarching cross-platform Master
  - Each platform follows a master-slave setup where the metadata is stored in the master application
  - Cross-platform master communicates with platform masters for any cross-platform communication
  - This architecture is applicable to all applications, databases, and communication mechanisms
  - Each platform has a dedicated access governance process, that is native to the respective platform on which it's hosted
- **Communication Protocols**
  - The communication between applications is done by using technologies like APIs, Connect-Direct, Message Queues across the HTTPS and SMTP protocols
- **Azure Active Directory Integration for Azure:**
  - **Unified Identity Management:** Implement Active Directory (AD) to manage user identities across all systems, ensuring consistent access control and simplifying user management
  - **Single Sign-On (SSO):** Use SSO with AD to provide seamless access to all integrated systems, reducing the need for multiple logins and enhancing security
- **Service Accounts:** Leverage service accounts with the optimal permissions to access data from multiple cloud platforms to attain higher throughput
- **OAuth:** Utilize OAuth 2.0 for secure authorization between the application and the cloud systems, allowing users to grant access without sharing their credentials
- **Ansible for infrastructure management:** Ansible playbook was leveraged across all platforms to manage infrastructure, thereby providing high scalability and low TAT

Cross-platform integration was done to optimize compute, storage, time and costs, along with better governance

Platform Engineering and DevOps

Cross Platform Integration (Slide 2 of 2)



Master-Slave architecture was designed for better IT governance

salessupport@tresvista.com | www.tresvista.com