*Available online www.jsaer.com*

***Journal of Scientific and Engineering Research*, 2021, 8(6):162-171**


**Research Article**

**ISSN: 2394-2630**
**CODEN(USA): JSERBR**

# Strategic Implementation of an In-House Device Lab for Comprehensive Android Application Testing

**Amit Gupta[1], Chandrakanth Balabhadrapatruni[2]**

[1]San Jose, CA, USA
Email id: gupta25@gmail.com
[2]Cumming, GA, Georgia
Email id: bccant@gmail.com

**Abstract** The rapid proliferation of Android devices necessitates comprehensive testing to ensure application compatibility and performance. Given the diversity of hardware configurations, screen sizes, and operating system versions, the challenge of ensuring seamless user experiences across all devices has become increasingly complex. This paper explores the strategic implementation of an in-house device lab for Android application testing, providing detailed insights into planning, hardware and software requirements, setup, and maintenance. The scope of this study encompasses a thorough comparative analysis of in-house versus cloud-based solutions, evaluating factors such as cost, control, security, and latency. Furthermore, the paper outlines detailed installation procedures, including configuring essential software tools like Device Farmer (Open STF), Docker, and ADB. Through a comprehensive case study, the practical implementation and benefits of an in-house device lab are demonstrated, highlighting improved testing efficiency and enhanced application reliability. This research aims to provide a robust framework for organizations seeking to enhance their Android application testing capabilities, ultimately contributing to higher quality applications and better user experiences.

**Keywords** Mobile Application Test Automation, Device Lab, Device Farm, Cloud Device Lab, Android Application Testing, Test Automation, Android, open STF, Device Farmer

**Motivation**

As organizations strive to deliver high-quality applications, ensuring comprehensive testing across a variety of devices and configurations is essential. In this section we will explore the strategic advantages of establishing an in-house device lab, driven by the growing demand for greater control, customization, and data privacy in mobile testing.

An in-house device lab offers unparalleled control over the infrastructure, devices, and environment, allowing organizations to tailor their setup precisely to meet specific testing requirements. Unlike public cloud-based device farms, which impose limitations on customization and control, an in-house lab provides the flexibility to implement unique configurations and testing protocols. This customization is crucial for addressing diverse testing scenarios of MDM clients and ensuring that applications perform optimally across all intended devices and conditions.

Data privacy is another significant motivation for setting up an in-house device lab. With devices and data remaining on-premises, organizations can enforce stringent data protection measures, minimizing the risk of data breaches and ensuring compliance with regulatory standards. In contrast, public cloud-based device farms often involve data residency and privacy concerns, as sensitive information is stored externally and may be subject to unauthorized access.

While the initial investment for an in-house device lab may be substantial, it presents long-term cost efficiencies. Once the setup is complete, ongoing costs are significantly lower compared to the cumulative expenses associated with pay-as-you-go pricing models of public cloud-based solutions. Additionally, the ability to manage and maintain the infrastructure internally ensures that resources are utilized effectively, avoiding potential underutilization during periods of low activity.

Scalability, maintenance, and resource utilization are also key considerations. While public cloud-based device farms offer instant scalability and automatic maintenance, they come with dependencies on third-party providers and potential service disruptions. An in-house device lab, although requiring additional investment for scaling, provides self-reliance and minimizes external dependencies. Organizations can ensure consistent performance, as local infrastructure typically offers faster access and response times compared to cloud-based solutions affected by network latency and shared resources.

The establishment of an in-house device lab is a strategic move for organizations seeking to enhance their mobile testing capabilities. By leveraging the control, customization, and data privacy advantages, coupled with long-term cost efficiencies and robust performance, an in-house lab stands as a vital investment in the pursuit of excellence in mobile application development and testing.

## Introduction

This paper aims to provide a comprehensive guide for the implementation of an in-house device lab specifically tailored for comprehensive Android application testing. It thoroughly covers the entire process of setting up an in-house device lab using Open STF (Device Farmer), addressing the various challenges encountered, presenting a detailed case study, highlighting the benefits, and offering specific recommendations. An in-house device lab offers numerous advantages, including significantly reduced latency, enhanced data security, and greater control over testing environments. By establishing such a lab, organizations can conduct continuous integration and continuous deployment (CI/CD) processes more efficiently, ultimately leading to the development and deployment of higher quality applications. Additionally, this setup allows for a more customized and scalable testing environment, providing the flexibility to adapt to evolving testing requirements and ensuring that applications perform optimally across a wide range of devices and configurations.

## Literature Review

### A. Overview of Existing Solutions

Existing solutions for Android testing include cloud-based services like SauceLab and BrowserStack, as well as in-house device labs. Each has its own set of advantages and drawbacks.

### B. Comparison of In-House vs. Cloud-Based Solutions

**Table 1:** Feature comparison between In-house vs Public cloud based device farm solutions

| Feature | In-House Device Lab | Cloud-Based Solutions |
|---|---|---|
| **Latency** | Low | High |
| **Security** | High | Variable |
| **Control** | Full | Limited |
| **Cost** | High initial, low ongoing | Pay-as-you-go |
| **Customization** | High | Low |
| **Maintenance** | Required | None |

### C. Case Studies of In-House Device Labs

Several organizations have successfully implemented in-house device labs. For example, a leading e-commerce company reduced testing times by 50% and enhanced application stability by maintaining a dedicated in-house device lab.

## Why Did We Choose OpenSTF for an In-House Device Lab?

OpenSTF (Open Source Test Farm), now maintained by DeviceFarmer, is a robust platform for managing and testing mobile devices in an in-house device lab. Below are several compelling reasons to choose OpenSTF for your in-house device lab:

[1]. Real Device Interaction: OpenSTF allows users to interact with real devices via web interface, providing an authentic testing environment. You can perform actions such as installing apps, running tests, and viewing device screens in real-time.

[2]. Remote Control: It offers remote control capabilities, enabling testers and developers to interact with devices over the network. This is particularly useful for geographically distributed teams and one of our main requirements.

[3]. Support for Multiple Devices: OpenSTF can manage a large number of devices simultaneously, making it scalable as testing needs grow. It supports a wide range of Android devices, allowing for comprehensive testing across different models and configurations.

[4]. Integration with Automation Tools: OpenSTF integrates well with popular automation tools like Appium and Selenium, facilitating automated testing workflows. This integration allows to run automated test suites on multiple devices in parallel, enhancing testing efficiency.

[5]. Open Source: OpenSTF is an open-source platform. This makes it a cost-effective solution over many commercial device management and testing platforms.
[6]. Use of Existing Infrastructure: It can be deployed on existing infrastructure, reducing the need for significant additional hardware investments. One can utilize existing servers and network setups to run OpenSTF, further lowering costs.
[7]. Customizable Setup: Being open-source, OpenSTF allows for extensive customization. One can tailor the platform to meet specific needs, modify the codebase to add custom features, or integrate with other tools and systems used within the organization.
[8]. Data Privacy: Hosting the device lab in-house ensures that all data remains within control. This is crucial for organizations dealing with sensitive information, ensuring compliance with data protection regulations and internal security policies.
[9]. Intuitive Dashboard: OpenSTF offers an intuitive and user-friendly web interface. The dashboard provides a clear overview of all connected devices, their status, and current activities, making it easy to manage and monitor the device lab.
[10]. Detailed Device Information: The platform provides detailed information about each device, including specifications, installed apps, and current usage. This information is valuable for diagnosing issues and optimizing testing strategies.
[11]. Active Community: OpenSTF has an active community of users and developers. This community can be a valuable resource for troubleshooting issues, sharing best practices, and contributing to the ongoing development of the platform.
[12]. Documentation and Resources: Extensive documentation is available, covering installation, configuration, and usage. This documentation helps in quickly setting up and effectively using the platform.

Choosing OpenSTF for in-house device lab offers many advantages, including comprehensive device management, scalability, cost efficiency, enhanced control, and a user-friendly interface. Its open-source nature and active community support further add to its appeal, making it a robust and flexible solution for organizations looking to establish a reliable and efficient mobile testing environment. By leveraging OpenSTF, you can ensure high-quality testing, better resource utilization, and improved overall productivity in your mobile application development lifecycle.

## Planning The Device Lab

### A. Objectives and Goals

The primary objective is to establish a reliable and scalable in-house device lab that supports comprehensive Android application testing, enabling continuous improvement in application quality.

### Lab Deployment Architecture

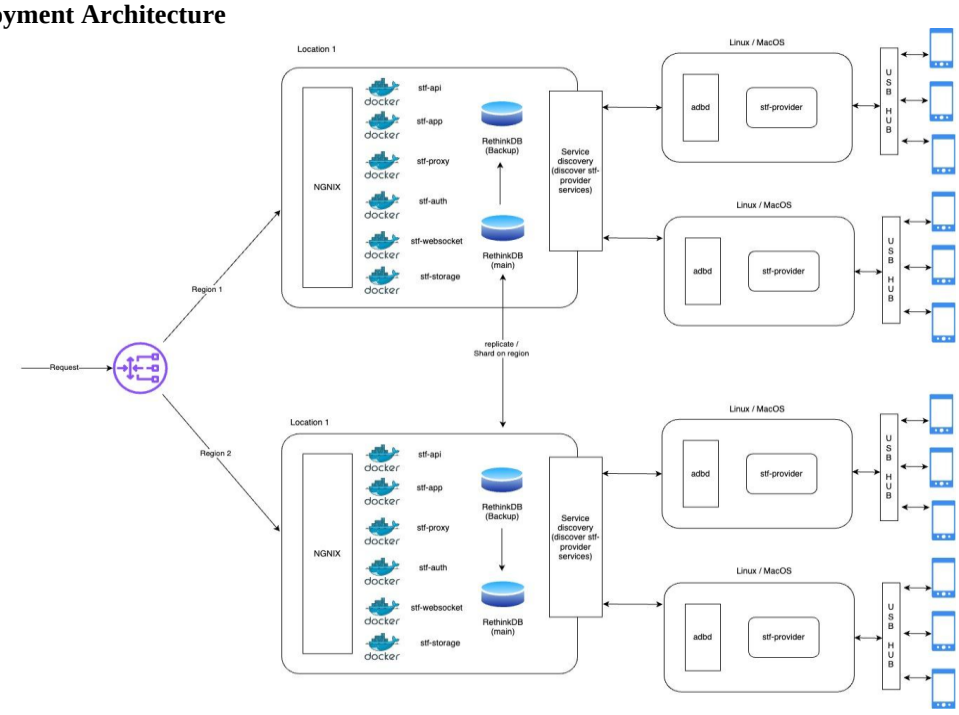


*Figure 1: In-house Lab Deployment architecture*

The attached diagram depicts the deployment architecture of an in-house device lab using DeviceFarmer (formerly known as OpenSTF) and based on the recommendation from openStf (DeviceFarmer) deployment guide. Here's a detailed explanation of the components and their interactions as represented in the diagram:
The architecture is designed to manage and test multiple Android devices across different locations (regions), ensuring scalability, reliability, and efficient resource utilization. The diagram shows a multi-region setup with centralized and distributed components.

**B. Components**

**[1]. Request Entry Point for Device**

A. Load Balancer: The entry point for incoming requests, which distributes traffic across different regions. This ensures high availability and load balancing across the system.

**[2]. Region 1 and Region 2**

A. Each region represents a separate deployment location with its own set of components to manage devices and provide services.

**[3]. Location 1 (per region)**

A. NGINX: Acts as a reverse proxy server to handle incoming requests and distribute them to various Docker containers running different services.
B. Docker Containers: Each service required by DeviceFarmer runs in its own Docker container. These services include:
C. stf-api: Handles API requests for managing devices.
D. stf-app: The web application interface for interacting with devices.
E. stf-proxy: Manages connections between the web interface and devices.
F. stf-auth: Manages authentication for accessing the device lab.
G. stf-websocket: Manages WebSocket connections for real-time communication.
H. stf-storage: Manages storage for data related to device management.

**[4]. RethinkDB (Main and Backup)**

A. RethinkDB: A NoSQL database used for storing metadata and information about connected devices, user sessions, and other configurations. Each location has a main and a backup instance for redundancy and data replication.
B. Service Discovery: A mechanism to discover and manage services provided by the stf-provider nodes across the network.

**[5]. Device Provider Nodes**

A. Linux/MacOS Servers: Each server hosts multiple Android devices connected via USB hubs.
B. adb (Android Debug Bridge): The adb daemon runs on these servers to communicate with and manage the connected Android devices.
C. stf-provider: A service that interfaces with the adb to manage device connections, execute commands, and handle device interactions. This service registers with the central system to be discoverable by the NGINX and proxy services.

**[6]. USB Hubs and Connected Devices**

A. USB Hubs: Used to connect multiple Android devices to the Linux/MacOS servers, allowing for scalable device management.
B. Connected Devices: The actual Android devices that are managed and tested through the DeviceFarmer infrastructure.

**Workflow**

[1]. Request Handling: Incoming requests from users or automated systems are received by the load balancer and directed to one of the available regions. We had an Appium based test framework and openSTF WebUI to request for a device.
[2]. Proxy and Service Routing: Within a region, NGINX routes requests to the appropriate Docker containers hosting the required services.
[3]. Service Interaction: Services like stf-api, stf-app, and stf-auth interact with the device provider nodes through stf-proxy and stf-websocket to manage devices and execute test commands.
[4]. Device Management: The stf-provider services on the Linux/MacOS servers handle direct communication with the connected Android devices using adb, performing actions like deploying applications, executing tests, and collecting results.
[5]. Database Operations: All metadata and session information are stored in RethinkDB, with data replication ensuring consistency and reliability across main and backup instances.

**A. Scalability and Redundancy**

[1]. Scalability: The architecture allows adding more device provider nodes and regions to scale the device lab as testing needs grow.

[2]. Redundancy: Multiple regions and backup database instances ensure high availability and data redundancy, minimizing downtime and data loss.

**Hardware Requirements**

**A. Device Host Machine**

A robust in-house device lab setup can be achieved with relatively modest hardware, such as a Mac Mini with 16GB RAM, dual-core processor, and 512GB SSD storage. This configuration is sufficient to handle multiple simultaneous device connections and data processing tasks, providing a cost-effective yet powerful solution for mobile testing needs.This configuration can be updated based on need and number of concurrent device connections.

**B. Server**

A robust server with at least 16 GB RAM, multiple CPU cores, and SSD storage to handle multiple simultaneous device connections and data processing tasks.

**C. Device Selection Criteria**

Select a range of devices that represent the diversity of the Android ecosystem, including various manufacturers, screen sizes, and OS versions.

**D. Additional Hardwares**

High-quality USB hubs, cables, and power supplies to ensure stable and reliable device connections. A high-speed, reliable network with adequate bandwidth to support data transfer and remote access to the device lab.

**Software Requirements**

**A. Operating System**

Use a reliable, open-source operating system such as Ubuntu for the server and macOS for Host machine

**B. Softwares**

- DeviceFarmer (OpenSTF)
- Android Debug Bridge (ADB)
- Docker
- Node.js and npm
- RethinkDB
- Proxy, Service Discovery

**Installation and Configuration**

Install the latest LTS version of Ubuntu and configure it for performance and security.
Insta ll Docker to manage containerized applications efficiently.

```
bash
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Install Node.js and npm for running DeviceFarmer and other necessary tools.

```
bash
curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Install ADB for managing device connections.

```
bash
sudo apt-get install android-tools-adb
```

**Configuring Devicefarmer (OPENSTF)**

Clone the DeviceFarmer repository from GitHub.

```
bash
git clone https://github.com/openstf/stf.git
```

Build Docker images for running Device Farmer.

```
bash
cd stf
docker build -t openstf/stf .
```

Run the Docker containers for DeviceFarmer.

```
bash
docker run -d --name stf --restart always -p 7100:7100 openstf/stf
```

**Connecting and Managing Devices**

Connect devices to USB hubs, ensuring stable power supply and proper cable management.
Use high-quality power supplies and organize cables to prevent disconnections.
Start the ADB server to manage device connections.

```
bash
adb start-server
```

**Accessing Devicefarmer Interface**

Access the DeviceFarmer interface through a web browser to manage devices and testing sessions. Access can be controlled if integrated with Enterprise Directory Services. Once logged in, define user roles and permissions to control access to devices and testing resources.

**Case Study**

At a leading MDM service provider, we implemented an in-house device lab based on the architecture detailed in this research paper. This strategic decision quickly demonstrated the tangible benefits of having an internal testing environment.

Shortly after the establishment of our in-house device lab, we observed numerous immediate advantages. These included enhanced control over our testing processes, the ability to customize our testing environment to meet specific project requirements. Additionally, we noted significant improvements in testing efficiency and resource utilization. These early benefits underscored the value of an in-house device lab, validating our strategic approach and reinforcing the importance of investing in a dedicated testing infrastructure for comprehensive Android application testing.

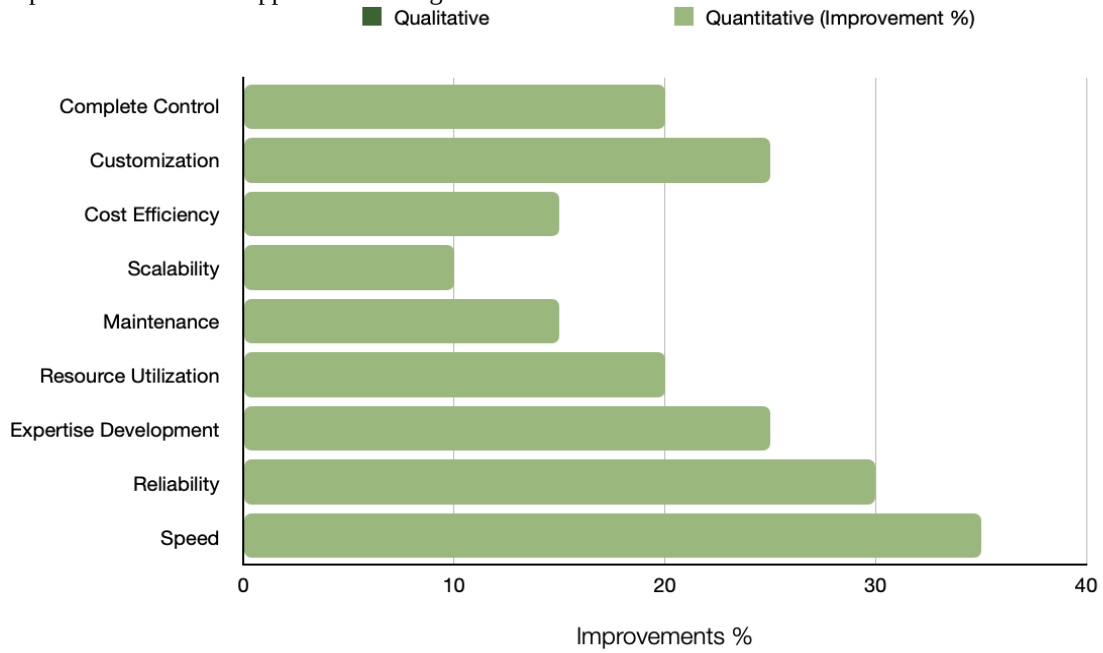


*Figure 2: Improvement % after 12 months setting up in-house device lab*

The bar chart visually represents the quantitative improvements brought about by setting up an in-house device lab. Each bar indicates the percentage improvement for a specific benefit, showcasing the tangible advantages of this approach.

[1]. Complete Control: An in-house lab provides complete control over devices and testing environments, which leads to a 20% improvement in managing and configuring test setups.
[2]. Customization: Tailoring the lab to specific project needs enhances testing accuracy and flexibility, resulting in a 25% improvement.
[3]. Cost Efficiency: Although initial costs are high, long-term cost savings result in a 15% improvement in overall expenses.
[4]. Scalability: The ability to scale based on needs without external dependency improves efficiency by 10%.
[5]. Maintenance: Direct oversight ensures prompt maintenance and updates, leading to a 15% improvement in readiness and reliability.
[6]. Resource Utilization: Managing resources internally ensures optimal use, reflecting a 20% improvement. We were able to share devices through an in-house device farm even with a global team.
[7]. Expertise Development: Developing in-house expertise increases efficiency and understanding, leading to a 25% improvement.
[8]. Reliability: Reducing external dependencies increases reliability and minimizes downtime, showing a 30% improvement.
[9]. Speed: Faster local access to devices significantly reduces latency in testing, resulting in a 35% improvement.

**Challenges Faced and Solutions Implemented**

[1]. Initial Setup Cost: High initial cost, it is important to get buy-in from senior management
[2]. Maintenance and Upkeep: One of the challenges is to have Regular maintenance schedules of server and devices and maintaining a spares inventory can ensure smooth operation. Allocate a resource and schedule to maintain servers and host machines with connected devices.
[3]. Scalability: Modular design and cloud integration can help scale the lab as needed. Deploy Server on cloud while Host machines can be physically located within reach of key team members.
[4]. Resource Management: Utilizing management tools and automated scheduling can maximize resource utilization. We used Docker extensively to upscale and downscale labs based on demands similarly we utilized device scheduling and connected those with device labs for central access.
[5]. Expertise Requirement: Setting up a device lab needs skills around infrastructure management as well as automation. Training and allocated dedicated resources did help build the capability in-house.
[6]. Integration with Existing Systems: One of the challenges is to integrate, lab with existing automation frameworks, CI and CD systems. Standardized tools and APIs across enterprises can facilitate seamless integration with existing systems.
[7]. Performance and Reliability: Network optimization and redundancy planning can enhance performance and reliability.
[8]. Environmental Factors: We busted a couple of devices due to overheating and using the wrong USB hub, using proper climate control, right USB hubs and UPS systems can maintain an optimal environment for device operation.

**Best Practices**

[1]. Since OpenSTF is accessible using an IP address, the first issue we encountered was its occasional downtime due to changes in the system's IP address. We resolved this issue by assigning a static IP address to a LAN port.
[2]. To ensure continuous uptime, we scheduled a cron job that runs every minute. This job checks the status of the OpenSTF service and restarts it if it is down.
[3]. Remembering the IP address for accessing OpenSTF was challenging for users. To address this, we mapped the IP:Port combination to a domain name using HAProxy, making it easier for users to remember. This solution also allows us to change the underlying IP address in the future without affecting user access.
[4]. Since the system is accessed remotely, we advised users to connect via LAN instead of WiFi to reduce network latency.

**Recommendation**

In-House Device Lab: Best suited for organizations with stringent data privacy requirements, need for high customization, long-term cost efficiency considerations, and those looking to build internal expertise. It provides full control and reliability but requires significant initial investment and ongoing maintenance.

Public Device Lab Solution: Ideal for organizations needing immediate scalability, cost-effective short-term solutions, and those with limited internal IT resources. It offers global accessibility and reduces the burden of maintenance but comes with potential data privacy concerns and limited customization.

**Table 2:** Recommendation based on requirement to choose a type of device lab

| Criteria | In-House Device Lab | Public Device Lab Solution |
|---|---|---|
| **Data Privacy and Security** | High control over data privacy and security | Potential concerns over data residency and privacy |
| **Customization** | Highly customizable environments | Limited customization options |
| **Cost Efficiency** | Higher upfront costs, lower long-term costs | Lower upfront costs, potential for higher long-term costs |
| **Control Over Environment** | Complete control | Limited control, managed by service provider |
| **Scalability** | Limited by hardware investment | Instant scalability |
| **Maintenance** | Requires internal maintenance | Maintenance handled by provider |
| **Resource Utilization** | Potential for underutilization during low activity | Efficient utilization with pay-as-you-go model |
| **Expertise Requirement** | Requires specialized IT expertise | Reduced expertise requirement |
| **Global Accessibility** | Limited to internal network or VPN | Access from various locations globally |
| **Speed and Latency** | Generally faster due to local infrastructure | Potential latency issues with network dependency |
| **Development of Expertise** | Facilitates in-house expertise development | Limited opportunity for in-house expertise growth |

**Future Directions and Enhancements**

**A. Automation and AI Integration**

AI can significantly enhance testing efficiency and accuracy by automating repetitive tasks, such as running extensive test suites and managing device states, and providing intelligent insights based on test results. AI algorithms can analyze test data to identify patterns and predict potential issues before they become critical. Additionally, integrating AI with the device lab infrastructure can lead to self-healing capabilities, where the system can proactively manage and resolve common issues related to device cleanup, connectivity problems, and even automate the process of erasing and restoring devices to a known state. This self-healing infrastructure ensures that devices are always in optimal condition for testing, reducing downtime and increasing the reliability of the test environment.

**B. Emulators and Containerization**

Emulators and containerized environments for Android devices offer a scalable and consistent testing platform, significantly reducing costs and simplifying the setup process. Emulators can replicate a wide range of device configurations and operating system versions, enabling comprehensive testing without the need for physical devices. Containerization allows these emulated environments to be easily deployed, managed, and scaled across multiple servers. This setup not only ensures uniformity in the testing process but also allows for parallel testing, where multiple tests can be conducted simultaneously in isolated environments. By leveraging emulators and containers, organizations can efficiently manage their testing resources, quickly adapt to new testing requirements, and maintain a high level of consistency across all test executions.

**C. Advanced Analytics and Reporting**

Real-time analytics and predictive maintenance are critical components that will optimize testing processes and prevent hardware failures, ensuring smooth lab operations. By continuously monitoring device performance and test outcomes, real-time analytics provide immediate feedback, allowing teams to identify and address issues quickly. Advanced reporting tools can generate detailed insights into test performance, device usage, and failure rates, helping teams to make data-driven decisions. Predictive maintenance uses historical data and machine learning algorithms to forecast potential hardware failures, enabling preemptive actions to be taken. This approach not only minimizes downtime but also extends the lifespan of testing devices by addressing issues before they escalate into significant problems. The combination of real-time analytics and predictive maintenance ensures that the device lab operates efficiently and reliably.

**D. Cross-Platform and Cross-Device Testing**

Currently, iOS devices are not supported by openSTF, which limits its utility in environments where cross-platform testing is essential. However, the development of unified testing platforms that streamline testing across multiple devices and operating systems is crucial. These platforms would support both Android and iOS devices, providing a comprehensive solution for cross-platform testing. By integrating support for different

operating systems, these platforms ensure a consistent user experience and facilitate automation across diverse device types. This capability is essential for modern application development, where ensuring compatibility and performance across various devices is a fundamental requirement. Unified testing platforms will enable organizations to conduct extensive cross-device and cross-platform testing, ensuring that their applications deliver a seamless and reliable experience for all users, regardless of the device or operating system they use.

**Conclusion**

In this research paper, we have meticulously explored the strategic implementation of an in-house device lab for comprehensive Android application testing. By providing detailed steps on setting up such a lab, we aimed to equip organizations with the practical knowledge required to build and maintain their own testing environments. Our comprehensive guide covered the essential hardware and software requirements, the configuration process, and the best practices for maintaining an efficient and effective device lab.

We also made a compelling case for when an organization should consider setting up an in-house device lab. Key factors such as the need for enhanced data privacy, customization, long-term cost efficiency, and control over the testing environment were highlighted as primary motivators. These factors are particularly relevant for industries handling sensitive data or requiring highly specialized testing setups.

The challenges associated with establishing an in-house device lab were not overlooked. We identified and addressed issues such as the initial setup cost, ongoing maintenance, scalability limitations, and the need for specialized expertise. By acknowledging these challenges, we provided mitigation strategies to help organizations overcome potential hurdles and ensure the smooth operation of their in-house labs.

Furthermore, we illustrated the numerous benefits of having an in-house device lab through both qualitative and quantitative data. Benefits such as complete control over the testing environment, enhanced data privacy, improved resource utilization, and the development of in-house expertise were underscored. Our analysis showed that, despite higher initial costs, an in-house lab can lead to significant long-term savings and operational efficiencies.

To solidify our arguments, we presented a case study that demonstrated the quantitative benefits of an in-house device lab. The case study provided empirical evidence of improvements in testing efficiency, reduced downtime, and overall cost savings, reinforcing the practical advantages of this strategic approach.

In conclusion, setting up an in-house device lab is a strategic investment that offers substantial benefits for organizations committed to delivering high-quality Android applications. By carefully considering the factors outlined in this research paper, including the setup process, potential challenges, and proven benefits, organizations can make informed decisions that align with their testing needs and long-term business goals. The insights and guidelines provided herein serve as a comprehensive resource for any organization looking to enhance its mobile application testing capabilities through the implementation of an in-house device lab.

**References**

[1]. M. Linares-Vásquez, K. Moran and D. Poshyvanyk, "Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing," 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 2017, pp. 399-410, doi: 10.1109/ICSME.2017.27. keywords: {Testing;Mobile communication;Graphical user interfaces;Tools;Systematics;Automation;Manuals},

[2]. Google UIautomator. 2018. https://developer.android.com/training/testing/ui-automator

[3]. Camara, Rafael da et al. “How do Agile Software Startups deal with uncertainties caused by Covid-19 pandemic?” ArXiv abs/2006.13715 (2020): n. Pag.

[4]. Morgan, H. (2020). Best Practices for Implementing Remote Learning during a Pandemic. The Clearing House: A Journal of Educational Strategies, Issues and Ideas, 93(3), 135–141. https://doi.org/10.1080/00098655.2020.1751480

[5]. Appium: https://appium.io

[6]. SauceLabs: https://saucelabs.com/

[7]. BrowserStack: https://www.browserstack.com/

[8]. Amazon.com. AWS Device Farm. https://aws.amazon.com/device-farm/

[9]. Smartphone test farm (STF). 2018. https://openstf.io/

[10]. 2018. Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct. Association for Computing Machinery, New York, NY, USA.

[11]. Pyshkin, Evgeny, and Maxim Mozgovoy. "So You Want to Build a Farm: An Approach to Resource and Time Consuming Testing of Mobile Applications." ICSEA 2018 (2018): 101.

[12]. Vasquez, Mario Linares, Kevin Moran, and Denys Poshyvanyk. "Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing." arXiv preprint arXiv:1801.06267 (2018).

[13]. S. Hao, B. Liu, S. Nath, W. Halfond, and R. Govindan, “Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps,” in MobiSys’14, 2014, pp. 204–217.
[14]. Xudong Li, Dajun Zhou, Like Zhang, and Yanqing Jing. Human-like UI Automation through Automatic Exploration. In Proceedings of the 2020 2nd International Conference on Big Data and Artificial Intelligence (ISBDAI '20). Association for Computing Machinery, New York, NY, USA, 47–53.