The American Journal of Applied Sciences
ISSN 2689-0992 | Open Access

TYPE Original Research
PAGE NO. 20-29
DOI 10.37547/tajas/Volume07Issue07-03

A Comparative Analysis of Pivotal Cloud Foundry and OpenShift Cloud Platforms

Srikanth Reddy Gudi
IEEE Member: 101425246, USA

OPEN ACCESS

SUBMITED 14 May 2025
ACCEPTED 29 June 2025
PUBLISHED 09 July 2025
VOLUME Vol.07 Issue 07 2025

CITATION

Srikanth Reddy Gudi. (2025). A Comparative Analysis of Pivotal Cloud Foundry and OpenShift Cloud Platforms. The American Journal of Applied Sciences, 7(07), 20–29. https://doi.org/10.37547/tajas/Volume07Issue07-03.

Abstract: The paper presents a comprehensive comparative analysis of two leading enterprise-grade Platform as a Service (PaaS) solution: Pivotal Cloud Foundry (PCF) and Red Hat OpenShift. It examines their architectures, deployment models, operational characteristics, developer experiences, security features, performance attributes, and ecosystem support. The research highlights key differences between PCF's custom architecture with Warden containers and OpenShift's Kubernetes-native approach. The analysis covers installation procedures, management tools, application deployment workflows, and migration strategies between platforms. Through case studies and literature review, the paper provides organizations with guidance for making informed decisions about which platform best suits their specific requirements and constraints.

Keywords: Platform as a Service (PaaS), Cloud Foundry, OpenShift, Kubernetes, containerization, microservices, cloud computing, container orchestration, developer experience, cloud migration.

1. Introduction:

The way businesses deploy, scale, and manage applications has been completely transformed by cloud computing. According to Zhang et al. (2010), cloud computing "eliminates the requirement for users to plan ahead for provisioning and allows enterprises to start from the small and increase resources only when there is a rise in service demand." It has "emerged as a new paradigm for hosting and delivering services over the Internet." [1]. Platform as a Service (PaaS) offering have become increasingly potent as cloud technologies have

advanced, abstracting infrastructure complexities and freeing developers to concentrate on application development rather than operational issues.

Red Hat OpenShift and Pivotal Cloud Foundry (PCF) are two of the top enterprise-grade platforms among the many PaaS options currently on the market. According to Lomov (2014), "OpenShift and Cloud Foundry have gathered the strongest development communities of any open-source projects in the category known as Platform-as-a-Service. They are regarded by many as the top open-source PaaS. [2]. Organizations making strategic decisions regarding their cloud infrastructure must comprehend the distinctions between these platforms.

This comparative analysis is significant for several reasons. First, according to IDC, the PaaS market is expected to expand dramatically, reaching $14 billion by 2017. [2]. Second, the choice of orchestration platform becomes crucial for developer productivity and operational efficiency as more organizations embrace microservices architectures and containerization technologies. Third, while PCF has gradually moved toward Kubernetes compatibility, OpenShift is Kubernetes-native, and both platforms represent distinct methods for addressing related issues.

This development is highlighted by Gelley (2022), who observes that businesses are moving "from Pivotal Cloud Foundry to Kubernetes" more frequently because of "high licensing costs" and to "increase the deployment flexibility." [3]. This change emphasizes how crucial it is to comprehend the operational and technical distinctions between these platforms.

Research Aim

This research paper's main goal is to present a thorough comparison of Pivotal Cloud Foundry and OpenShift by looking at their features, architectures, deployment strategies, and operational traits. Organizations will be better able to choose the platform that best fits their unique needs and limitations thanks to this analysis.

Main Contributions

Several new insights into cloud platforms are provided by this paper:

1. It offers a thorough architectural comparison of PCF and OpenShift, emphasizing the main parallels and divergences between their implementation strategies and design philosophies.
2. It examines both platforms' operational features, such as management tools, monitoring capabilities, and installation processes.
3. It looks at the developer experience on both platforms, emphasizing application lifecycle management, service integration, and deployment workflows.
4. It assesses both platforms' pricing factors, licensing schemes, and community support networks.
5. Using best practices and case studies from the real world, it talks about migration tactics between the platforms.

2. Literature Review

Definitions of Key Concepts

Platform as a Service (PaaS): "A development platform and environment providing services and tools such as programming language execution environment, database, web server, etc." is what Zhang et al. (2010) claim PaaS offers. [1]. Instead of managing servers, networking, or storage, PaaS abstracts the underlying infrastructure, freeing developers to concentrate on creating applications.

Containerization: A lightweight type of virtualization known as containerization condenses an application and all of its dependencies into a single, transportable unit known as a container. Containerization is described as "a process that encapsulates an application and its dependencies into a single, lightweight unit, or container" by Daram et al. (2021) [4]. Containers are more effective and quicker to start than traditional virtualization because they share the host operating system's kernel.

Orchestration: The automated placement, synchronization, and administration of containers is referred to as orchestration. Orchestration platforms "offer a comprehensive suite of tools for orchestrating containers, managing workloads, and automating deployment processes," according to Daram et al. (2021) [4]. For containerized applications, orchestration tools manage operations like networking, scaling, deployment, and service discovery.

Kubernetes: Google was the original developer of the open-source container orchestration platform known as Kubernetes. Gelley (2022) defines it as "a container runtime that provides developers with a robust distributed framework that automatically scales clusters and applications and handles failovers" and "is used to

manage the lifecycle of applications across environments." [3]

Microservices: An application is organized using the microservices architectural style as a group of loosely coupled, independently deployable services. "Independently deployable by fully automated deployment machinery" is how Simioni (2017) characterizes microservices [5]. highlighting how microservices-based application deployment and management require automation.

Evolution of Cloud Computing and PaaS

Cloud computing paradigms have clearly evolved from Infrastructure as a Service (IaaS) to Platform as a Service (PaaS) and beyond, according to the literature. According to Zhang et al. (2010), John McCarthy's vision of "computing facilities will be provided to the general public like a utility" dates back to the 1960 [1]. But cloud computing didn't really take off until the 2000s, when commercial cloud services started to appear.

The need to streamline application deployment and management in cloud environments has fueled the growth of PaaS offerings. Chris Richardson's Cloud Tools project, which was "a set of tools for deploying Java applications to Amazon EC2" in 2007, is where Cloud Foundry's history started, according to Lomov (2014) [2]. Similar to this, Red Hat's PaaS offering, OpenShift, debuted in 2011 and focuses on offering an application deployment platform that is easy for developers to use.

Containerization and Orchestration

The transition from traditional virtualization to containerization is a prominent theme in the literature. "Virtual machines (VMs), while revolutionary at the time of their inception, come with significant overheads," as noted by Daram et al. (2021) [4], However, containers provide a lighter and more effective method for packaging and deploying applications.

Advanced orchestration platforms have emerged as a result of containerization. According to Daram et al. (2021), "the need for effective management and orchestration of these containers becomes evident as organizations increasingly adopt containerization." [4]. Because of this, platforms like Kubernetes, which serve as the basis for OpenShift, have developed.

Architectural Approaches

Various architectural approaches to creating PaaS platforms are revealed in the literature. In a thorough analysis of Cloud Foundry and OpenShift's architectures, Lomov (2014) points out that both systems have "components with similar functionality" like messaging buses, working nodes, routers, and managers [2].

Simioni (2017) highlights the value of microservices architecture in contemporary cloud platforms, pointing out that this strategy offers advantages in terms of team organization, scalability, and resilience [5]. Despite their differing implementations, PCF and OpenShift both clearly embrace the microservices architecture.

Developer Experience and Workflow

The significance of developer experience and workflow in PaaS platforms is a recurrent theme in the literature. In his comparison of Cloud Foundry and Kubernetes, Gelley (2022) points out that "Kubernetes, on the other hand, offers developers a resilient distributed framework that automatically scales clusters and applications and takes care of failovers," while "Cloud Foundry offers a higher-level abstraction for deploying applications so that developers can mainly concentrate on application development and deployment." [3].

The trend toward greater control and flexibility, even at the expense of greater complexity, is also highlighted in the literature. Gelley (2022) notes that "developers have more responsibility because they have to write and maintain the configuration needed for deployment and scalability due to Kubernetes' increased flexibility" [3].

Migration Between Platforms

The transition from Cloud Foundry to Kubernetes-based platforms is a recurring theme in recent literature. "To increase the deployment flexibility and to decrease licensing costs" were the main reasons for moving an application from PCF to Kubernetes, according to Gelley (2022) [3]. This is in line with a larger trend in the industry that Kubernetes is the most popular container orchestration platform.

The literature also highlights migration challenges, such as the fact that "no one in the development team had any previous technical expertise related to the Kubernetes environment," according to Gelley (2022). Therefore, during the migration, a significant learning path was required [3]. This emphasizes how crucial it is to take into account the training requirements and learning curve when switching between platforms.

3. Architectural Comparison

3.1 Core Architectural Components

3.1.1 Cloud Foundry Architecture

Cloud Foundry uses a modular architecture in which a few essential parts cooperate. Lomov (2014) states that the main elements consist of as shown in **Figure 1**:

- **Router**: Manages user traffic and directs it to the relevant instance of the application.
- **Cloud Controller**: Co-ordinates the deployment process, keeps up with the database of application metadata, and oversees applications and services.
- **DEA (Droplet Execution Agent)**: Uses Warden containers to run applications.
- **NATS (Message Bus)**: Offers a simple messaging system for component-to-component communication.
- **Build packs and Services**: Offer applications resources and services [2].

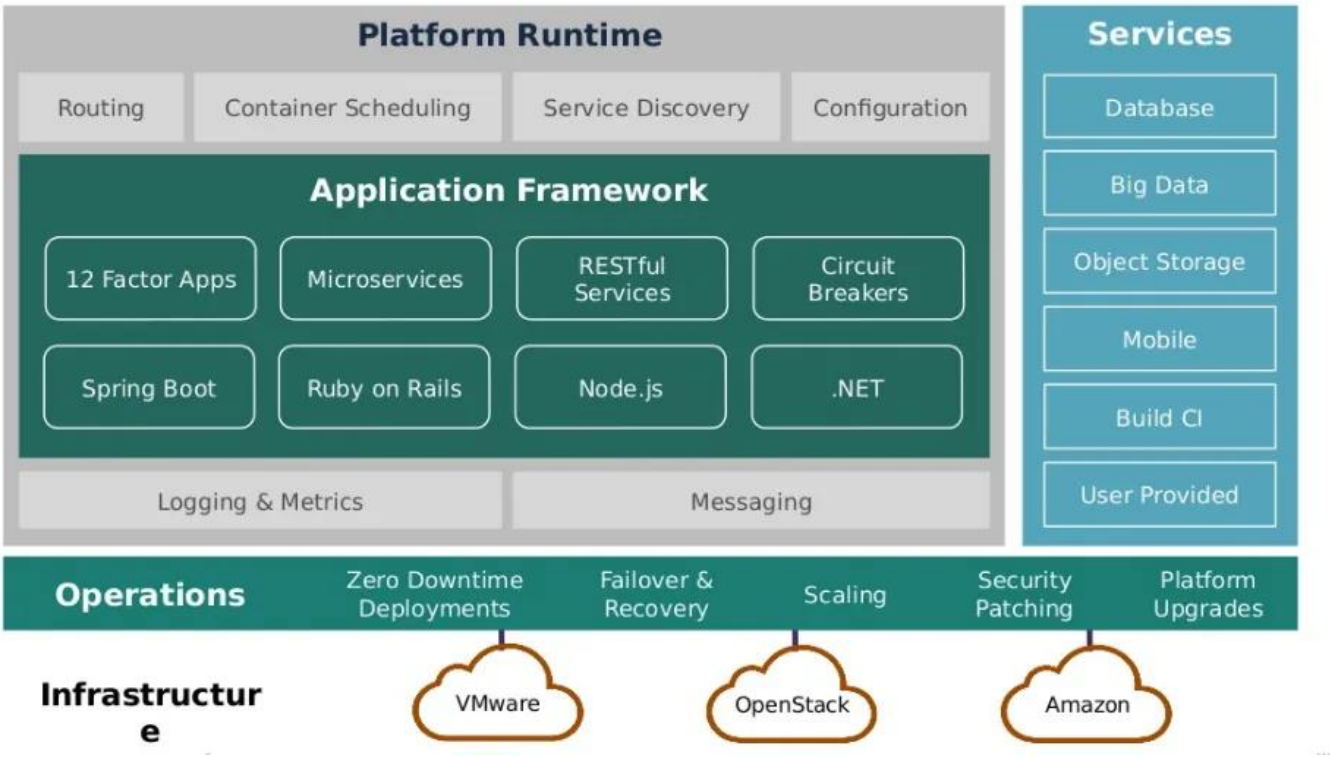

Figure 1: Pivotal Cloud Foundry — detailed look [9]

3.1.2 OpenShift Architecture

The architecture of OpenShift, which is based on Kubernetes, differs slightly. The following essential elements are identified by Lomov (2014):

- **Router/HAProxy Gears**: Control user traffic by directing it to the relevant service.
- **Brokers**: Serve as the liaison for all traffic and application management-related activities.
- **Gears**: Applications running in lightweight containers have independent access to shared resources.
- **ActiveMQ**: Acts as the component communication messaging bus.
- **Cartridges**: Provide the features required to run applications, such as database access and support for programming languages [2].

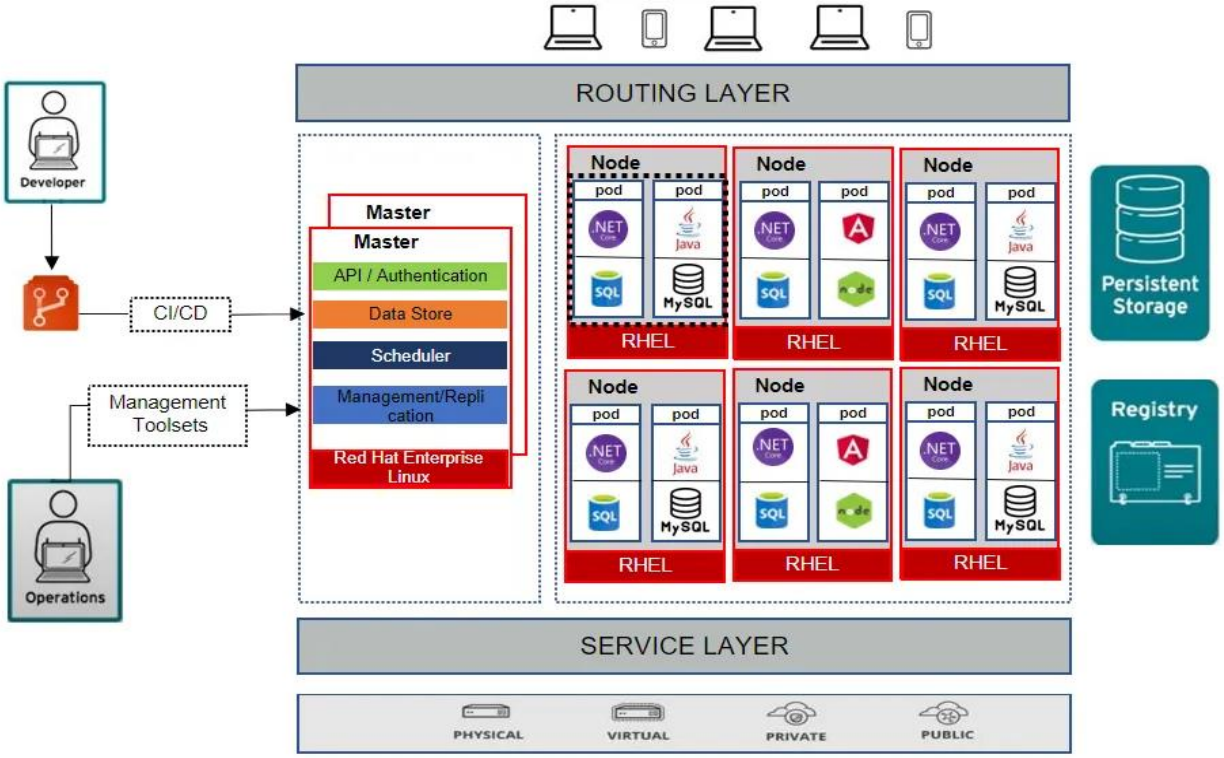

Figure 2: Components of Kubernetes cluster [8]

3.2 Virtualization and Containerization Approaches

3.2.1 Cloud Foundry's Warden Containers

Warden containers, which offer process isolation via Linux namespaces and control groups (cgroups), were initially used by Cloud Foundry. As stated by Lomov (2014), "Cloud Foundry uses Warden containers" [2], which, prior to Docker's widespread use, were created especially for Cloud Foundry.

3.2.2 OpenShift's Docker and Kubernetes Foundation

Kubernetes orchestrates the use of Docker containers by OpenShift. Lomov (2014) asserts that as shown in **Figure 2** "OpenShift uses Docker containers" [2]. Since OpenShift has adopted the industry-standard container runtime and orchestration platform, this signifies a fundamental architectural difference.

"Docker, a leading platform in this domain, has become synonymous with containerization, offering developers and IT operations teams a powerful tool to streamline the development, testing, and deployment of applications," according to Daram et al. (2021) [4]. OpenShift makes use of widely accepted industry standards by expanding upon Docker and Kubernetes.

3.3 Networking Architecture

3.3.1 Cloud Foundry Networking

A software-defined networking technique that offers application isolation is used by Cloud Foundry. The NATS messaging system facilitates internal communication, while the router component manages external traffic.

3.3.2 OpenShift Networking

Kubernetes networking features, such as services, ingress, and network policies, are utilized by OpenShift. "OpenShift supports deploying applications through a Git repository, hot deploys, and auto scaling," according to Lomov (2014) [2], which depends on its networking system.

3.4 Storage Architecture

3.4.1 Cloud Foundry Storage

Applications run on Cloud Foundry's ephemeral storage by default, with service bindings enabling persistent storage. With state externalized to supporting services, this methodology promotes stateless application design.

3.4.2 OpenShift Storage

A more adaptable storage architecture with support for multiple storage classes and persistent volumes is provided by OpenShift via Kubernetes. This makes it possible for the platform to support both stateful and stateless applications.

3.5 Scalability and High Availability

Although they take different approaches, both platforms offer mechanisms for scaling and guaranteeing high availability.

3.5.1 Cloud Foundry Scalability

By increasing the number of instances of components and application containers, Cloud Foundry can scale horizontally. The ability of cloud platforms to "automatically scale up and down according to the service-level agreements" is explained by Zhang et al. (2010) [1], an ability that Cloud Foundry has put into practice.

3.5.2 OpenShift Scalability

OpenShift makes use of the native scaling features of Kubernetes, such as cluster and horizontal pod autoscaling. Kubernetes offers "dynamic orchestration," which is beneficial for "improving the responsiveness and the operational agility of the system," according to Simioni (2017) [5].

4. Installation and Operations

4.1 Installation Procedures

4.1.1 Cloud Foundry Installation

Cloud Foundry installation can be challenging. "There are many ways to install Cloud Foundry," according to Lomov (2014), but it takes "a lot of RAM" and may require several steps [2], AWS, Google Cloud Platform, and vSphere are just a few of the infrastructure platforms on which Cloud Foundry can be installed.

4.1.2 OpenShift Installation

The installation of OpenShift is also complicated. The documentation for the OpenShift Container Platform [6]. Covers prerequisites, setup, and post-installation activities in its comprehensive installation and configuration instructions. Like Cloud Foundry, OpenShift is compatible with several infrastructure platforms.

4.2 Operational Tools and Interfaces

4.2.1 Cloud Foundry Operational Tools

Cloud Foundry offers a number of operational tools, such as:

- **CF CLI**: Command-line interface for Cloud Foundry interaction.
- **Apps Manager**: Web-based UI for managing applications and services.
- **BOSH**: Tool for deployment and lifecycle management of distributed systems.

4.2.2 OpenShift Operational Tools

OpenShift offers a different set of operational tools:

- **OC CLI**: Command-line interface for OpenShift.
- **Web Console**: Web-based UI for managing OpenShift clusters and applications.
- **Ansible**: Used for automated installation and configuration.

4.3 Monitoring and Logging

4.3.1 Cloud Foundry Monitoring

Through the Log aggregator component, which gathers and streams logs and metrics from applications and platform components, Cloud Foundry offers integrated monitoring capabilities.

4.3.2 OpenShift Monitoring

Operators can keep an eye on cluster health, resource utilization, and application performance with OpenShift's monitoring features via Prometheus and Grafana. Features for "monitoring and managing resources" are described in the OpenShift Container Platform documentation [6].

4.4 Upgrades and Maintenance

4.4.1 Cloud Foundry Upgrades

BOSH offers rolling updates with little downtime, making it possible to upgrade Cloud Foundry. But, particularly for large deployments, the procedure can be complicated.

4.4.2 OpenShift Upgrades

Mechanisms for rolling cluster and application updates are provided by OpenShift. According to Gelley (2022), "the Kubernetes rolling update strategy makes it simple to migrate the application without downtime." [3].

5. Developer Experience and Workflow

5.1 Application Deployment Models

5.1.1 Cloud Foundry Deployment Model

Cloud Foundry deploys using a straightforward push-based methodology. As "Cloud Foundry automatically identifies all the necessary runtime tools needed for the application and packages the application uses to build packs," Gelley (2022) explains that "in PCF, developers do not need to provide any descriptor about the dependencies required for the application to run in the cloud environment." [3].

5.1.2 OpenShift Deployment Model

OpenShift employs a Kubernetes-based deployment model that is more configuration-driven. According to Gelley (2022), "deployment manifests were needed to deploy to the Kubernetes environment" during the Kubernetes migration [3]. Because of this, developers must explicitly define several aspects of the deployment of their applications.

5.2 Build and Deployment Automation

5.2.1 Cloud Foundry Build Automation

Build packs are used by Cloud Foundry to automate the build procedure. "Buildpacks provide the actual functionality necessary to run a user application" is how Lomov (2014) puts it. [2].

5.2.2 OpenShift Build Automation

OpenShift automates builds using Docker builds and Source-to-Image (S2I). The statement "Docker provides a solid foundation for creating and managing containers" is made by Daram et al. (2021). [4].

5.3 Service Integration

5.3.1 Cloud Foundry Service Integration

A service broker API offered by Cloud Foundry makes it simple to integrate with outside services. By binding to services, applications can introduce login credentials and connection details into the application environment.

5.3.2 OpenShift Service Integration

OpenShift offers a more Kubernetes-native approach to service management by integrating services using Kubernetes service catalogs and operators.

5.4 Application Scaling and Management

5.4.1 Cloud Foundry Scaling

Cloud Foundry makes it simple to scale apps using the Apps Manager or CF CLI. The quantity of memory allotted to each instance, as well as the number of instances, can be changed by developers.

5.4.2 OpenShift Scaling

Through Kubernetes features like horizontal pod autoscaling, which can scale apps according to CPU usage or custom metrics, OpenShift offers more sophisticated scaling capabilities.

6. Security Features and Compliance

6.1 Authentication and Authorization

6.1.1 Cloud Foundry Authentication

Cloud Foundry supports multiple authentication providers, such as LDAP, SAML, and OAuth, and manages identities using UAA (User Account and Authentication).

6.1.2 OpenShift Authentication

OpenShift can integrate with multiple identity providers and uses OAuth. Features for "authentication and authorization" are described in the OpenShift Container Platform documentation. [6].

6.2 Network Security

6.2.1 Cloud Foundry Network Security

In addition to offering network isolation between apps, Cloud Foundry can be set up with extra security features like router-side TLS termination.

6.2.2 OpenShift Network Security

For more precise control over network traffic between pods, OpenShift makes use of Kubernetes network policies. For extra network security features, SDN (Software-Defined Networking) is also supported.

6.3 Container Security

6.3.1 Cloud Foundry Container Security

Applications are isolated from the host system and from one another using Cloud Foundry's Warden containers, which have multiple security features.

6.3.2 OpenShift Container Security

Additional features like security contexts, pod security policies, and SELinux integration are some of the ways

that OpenShift improves Docker container security. "OpenShift uses Docker containers, which have a different kind of abstraction" in contrast to Cloud Foundry's Warden containers, according to Lomov (2014) [2].

7. Performance and Scalability

7.1 Resource Efficiency

7.1.1 Cloud Foundry Resource Efficiency

The use of the Garden container runtime and Warden containers in Cloud Foundry contributes to its resource efficiency.

7.1.2 OpenShift Resource Efficiency

Kubernetes' sophisticated scheduling and resource management features enhance OpenShift's resource efficiency. "Kubernetes offers 'dynamic orchestration,' which is beneficial for 'improving the responsiveness and the operational agility of the system,'" according to Simioni (2017) [5].

7.2 Scalability Limits

7.2.1 Cloud Foundry Scalability Limits

Although Cloud Foundry's architecture allows it to scale to thousands of application instances, very large deployments may present difficulties.

7.2.2 OpenShift Scalability Limits

Large-scale deployments are the focus of OpenShift, which is based on Kubernetes and can grow to tens of thousands of pods and thousands of nodes. In order to overcome "the limitation of centralized Kubernetes architectures," researchers have begun studying "distributed Kubernetes architectures" after observing that even Kubernetes has difficulties with very large clusters. [7].

7.3 Performance Benchmarks

Although direct comparisons are challenging because of the disparities in architecture and deployment scenarios, several performance benchmarks have been carried out for both platforms.

8. Ecosystem and Community

8.1 Community Support and Development

8.1.1 Cloud Foundry Community

The Cloud Foundry Foundation serves as the focal point of the Cloud Foundry community. According to Lomov (2014), "in 2013, 732 contributors contributed more than 15,000 commits to Cloud Foundry." [2].

8.1.2 OpenShift Community

Red Hat is the main commercial sponsor of OpenShift, and the OpenShift community is closely related to the larger Kubernetes community.

8.2 Third-Party Integrations

Both platforms support a wide range of third-party integrations, though their approaches differ.

8.2.1 Cloud Foundry Integrations

Cloud Foundry integrations are primarily through service brokers and build packs.

8.2.2 OpenShift Integrations

OpenShift integrations leverage Kubernetes operators and the service catalog.

8.3 Commercial Support Options

8.3.1 Cloud Foundry Commercial Support

Commercial support for Cloud Foundry is available from VMware (formerly Pivotal) and other vendors.

8.3.2 OpenShift Commercial Support

Red Hat provides commercial support for OpenShift, with various subscription options available.

9. Case Studies

9.1 Cloud Foundry Adoption Cases

With proven advantages in terms of developer productivity and operational efficiency, Cloud Foundry has been used by numerous organizations for their PaaS requirements.

9.2 OpenShift Adoption Cases

Comparably, a lot of businesses have embraced OpenShift, especially those looking for a Kubernetes-native platform or those who have already made investments in the Red Hat ecosystem.

9.3 Migration Case Studies

Numerous companies have provided documentation of their transition from Cloud Foundry to OpenShift and

other Kubernetes-based platforms. A thorough case study of moving an insurance-related application from PCF to Kubernetes is given by Gelley (2022), who notes that the migration was successful and offered advantages like increased deployment flexibility and lower licensing costs [3].

10. Conclusion and Recommendations

10.1 Summary of Key Differences

The key differences between PCF and OpenShift include:

1. **Architectural Foundation**: OpenShift is based on Kubernetes and Docker, whereas PCF uses a unique architecture with Warden containers.
2. **Developer Experience**: In contrast to OpenShift, which offers greater flexibility and control at the expense of greater complexity, PCF offers a more abstracted, developer-friendly experience.
3. **Installation and Operations**: Although the installation processes for both platforms are intricate, their operational tools and methodologies differ.
4. **Ecosystem and Integration**: OpenShift integrates with the Red Hat ecosystem, whereas PCF works well with VMware products.
5. **Cost Model**: Due to the higher licensing costs associated with PCF, some organizations have shifted to alternatives based on Kubernetes.

10.2 Recommendations for Different Use Cases

Depending on their unique needs and limitations, various organizations may find one platform more appropriate than the other. A few things to think about are:

1. **Existing Investments**: Adopting PCF or OpenShift may be simpler for companies that have already made investments in the Red Hat or VMware ecosystems, respectively.
2. **Developer Skills**: While PCF may be preferred by organizations seeking maximum abstraction, OpenShift may be more approachable for those with developers who are familiar with Kubernetes.
3. **Scaling Requirements**: Organizations with very large-scale deployments might benefit from OpenShift's Kubernetes foundation.
4. **Budget Constraints**: Organizations with tight budget constraints might find OpenShift's licensing model more attractive.

10.3 Future Trends and Developments

The PaaS landscape continues to evolve, with several notable trends:

1. **Kubernetes Dominance**: Both PCF and OpenShift have been impacted by Kubernetes' rise to prominence as the leading container orchestration platform.
2. **Serverless Computing**: To accommodate serverless computing paradigms, both platforms are growing.
3. **Edge Computing**: Research on "distributed Kubernetes architectures" indicates that there is growing interest in bringing cloud platforms to edge environments. [7].
4. **AI and Machine Learning Integration**: Both platforms are investigating "MLOps Tools for Kubernetes" to better support AI and ML workloads. [7].

To sum up, PCF and OpenShift are both established, enterprise-class PaaS platforms, each with unique advantages and disadvantages. When deciding between them or thinking about switching from one to the other, organizations should carefully consider their unique needs and limitations.

REFERENCES

[1] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7-18.

[2] Lomov, Alexander. "OpenShift and Cloud Foundry PaaS: High-level Overview of Features and Architectures." Altoros, 2014.

[3] Gelley, S. (2022). Migrate Cloud Foundry Application to Kubernetes. Master's Thesis. Metropolia University of Applied Sciences, Master of Engineering, Information Technology.

[4] Daram, S., Jain, A., & Goel, O. (2021). Containerization and Orchestration: Implementing OpenShift and Docker. Innovative Research Thoughts, 7(4), 255-263. DOI: https://doi.org/10.36676/irt.v7.i4.1457

[5] Simioni, Alberto. "Implementation and evaluation of a container-based software architecture." Master's thesis, Università degli Studi di Padova, Dipartimento di Matematica "Tullio Levi-Civita", July 2017. Supervisor: Prof. Tullio Vardanega.

[6] 13 Red Hat, Inc. (2018). OpenShift Container Platform 3.6 Installation and Configuration. Red Hat Documentation.

[7] Patel, Indravadan. "D-K8S: Container Orchestration Through Nodes Empowerment and Participation." International Journal of Computer Trends and Technology, vol. 73, no. 2, Feb. 2025, pp. 23-30. https://doi.org/10.14445/22312803/IJCTT-V73I2P104

[8] https://darshanadinushal.medium.com/openshift-architecture-63c9e2974abe

[9] https://www.slideshare.net/Pivotal/t3-pivotal-cloud-foundry-a-technical-overview