International Conference on Computer Science, Engineering, and Modern Technology Innovations (ICCSEMTI-2026)
ISSN: 3107-9458|ICCSEMTI-2026 - Conference Proceeding
DOI: https://doi.org/10.64137/3107-9458/ICCSEMTI26-101

Original Article

Engineering for Scale: Performance Testing and Capacity Planning in OpenShift and Pivotal Cloud Foundry

Devenderrao Takkalapally
Performance Architect at Virtusa Corporation, USA.

Abstract: *As enterprises rush their cloud-native adoption, they put an enterprise container platform under an increased pressure to deliver consistent performance even with the unexpected and varying workloads that keep increasing rapidly. Hence, one can clearly see a demand for engineering work that goes beyond just deployment automation, specifically, a comprehensive performance testing and capacity planning based on data performance. Here we introduce a practical, scalable approach to engineering for performance in two very popular enterprise platforms: Red Hat OpenShift and Pivotal Cloud Foundry (PCF). We put forward a methodology integrating workload modeling, stress and endurance testing, service-level-objective (SLO) validation, and profiling of resource utilization spanning compute, memory, storage, and network layers. We illustrate the case of an enterprise application environment where bottlenecks unfold differently in OpenShift and PCF due to the differences in orchestration behavior, routing, autoscaling mechanisms, and platform abstractions. Our results highlight common failure modes such as the noisy-neighbor effect, the saturation of ingress components, and misaligned autoscaling thresholds leading to overprovisioning or performance degradation. The paper offers a testing framework which can be repeated, a capacity forecasting model that is business-demand oriented, and platform configuration tuning instructions that are both essential and feasible. Being fruits of the joint effort the paper lists improved reliability, predictable scaling, and quantifiable cost efficiency as the outcomes that shift teams to work stress-free in growth planning while still focusing on user experience.*

Keywords: *OpenShift, Pivotal Cloud Foundry (PCF), Performance Testing, Capacity Planning, Load Testing, Stress Testing, Scalability, Kubernetes, Cloud Native, Observability, Benchmarking, Autoscaling, Resource Utilization, SLO Validation, Bottleneck Analysis, Enterprise Platforms.*

1. INTRODUCTION

More and more enterprise companies are adopting cloud-native principles to build and modernize their apps, which means they consider scalability, resilience, and fast delivery as the basic character traits of their apps rather than the optional ones. Which is why traditional enterprise platforms like Red Hat OpenShift and Pivotal Cloud Foundry (PCF) have been gaining popularity as they offer standardized methods for deploying and managing apps at scale while also minimizing the operational costs of the development teams. The transition from monoliths to microservices-based architectures and containerized workloads, however, has put enterprises under greater strain to guarantee consistent performance, a problem that the control of the traffic, whether high or low, does not help solve. Basically, many teams come to realize that "operating a modern platform" isn't an automatic ticket to the jackpot of on-demand scaling, low latency, and high resource utilization. Besides being a challenge on the technical side, the change has also methodological dimensions: companies usually don't have the performance testing discipline, the practice of checking how an app scales, and the conversion of usage into capacity planning, that is, the three components neatly knitted into one framework. The result of lacking these engineering cornerstones is that platforms may unexpectedly scale, the bill may grow rapidly because more resources than actually required are being made available, and the real causes of performance degradation are found only after the system has been put to work. The current document concentrates on performance testing and capacity planning as the requisite practices for the engineering of large-scale systems in OpenShift and PCF, highlighting the importance of carrying out tests that closely resemble the real-world ones, identifying bottlenecks, and making scaling decisions based on empirical data.

1.1. CHALLENGES

Nowadays, enterprises mostly use microservices to build their applications. This results in a dramatic increase in the complexity of the architecture and runtime behavior of the application. Every service locally scales, talks to the other through the network, and depends on various shared infrastructure components such as service meshes, ingress controllers, message brokers, and databases. Moreover, this complexity is getting out of control when traffic patterns are unpredictable since real workloads seldom have a smooth linear growth and rather characterizes spikes, seasonal surges, sudden bursting driven by business events.

Resource contention is also a very significant issue in shared enterprise clusters. Multi-tenant environments are regularly subject to "noisy neighbor" situations, which illustrate that one team's workload dominates CPU, memory, I/O, or network bandwidth, and the other teams get the indirect impact. Besides, contention may still occur in shared components such as routers, storage systems, or logging pipelines even if quotas are set.

Platform abstractions make it even more difficult to track down the root cause of a problem. Red Hat OpenShift and Pivotal Cloud Foundry (PCF) provide the greatest support in making deployments easy by hiding the details of the infrastructure, but they can also make identifying bottlenecks quite difficult. Moreover, scaling or stretch limitations may also be different from one platform to another: For example, OpenShift scaling may be restrained by the number of pods, nodes, ingress controllers, or the cluster autoscaler behavior, whereas PCF may be restrained by routers, buildpacks, and Diego cells.

1.2. PROBLEM STATEMENT

While enterprises have generally gone for OpenShift and PCF to support large-scale application delivery, most of them don't have a clearly defined blueprint for how their workloads will behave when they grow. Platform capacity is often a guess based on rough rules of thumb, vendor guidance, or assumptions which do not actually mirror the organization's real application mix. Hence, performance testing is usually done as an afterthought, performed just before going live or after big releases. This kind of reactive approach makes it more certain that problems will show up in production where they are the hardest to diagnose and the most expensive to fix.

Besides that, capacity planning is usually very far away from actual workload behavior. So, teams may end up planning resources based on fixed CPU and memory allocations rather than on realistic traffic, concurrency, and response-time requirements. There are sometimes autoscaling policies put in place without checking how fast the platform reacts, whether the scaling thresholds are right, and what limitations exist at the routing or node stages.

Hence, a structured methodology is required that will allow enterprises to estimate capacity with measurable confidence, load test applications with realistic patterns, discover platform and application bottlenecks, and optimize scaling decisions before going to production. This article fills that void by offering a repeatable framework specifically designed for OpenShift and PCF.

1.3. MOTIVATION

There are two main reasons justifying the need for regular performance testing as well as capacity planning. In terms of costs, the most obvious sign would be when the IT infrastructure has been overprovisioned merely as a buffer for possible demand solely on a production environment. Large clusters of compute, storage, and networking can easily become very expensive if they are scaled without reasonable limits. At the same time, the risk of underprovisioning also presents a risk such as increased latencies, error rates, and higher likelihood of outages during periods of maximum demand. Taking a methodical approach can be a great help in the work that needs to be done to make cost optimization and performance needs agree with each other.

In fact, a reliability goal is quite another important reason. It is only natural that today enterprises are embracing the idea of setting service level objectives (SLOs) and service level agreements (SLAs) that are very strict, and these are cases where any amount of downtime or degraded performance can fairly be seen as a direct hit to revenue, loss of customer trust, and therefore breach of contract. Performance testing should be a matter of proof that the scaling strategy can really be adequate in meeting these targets rather than just putting one's faith in the platform's capability to handle growth inherently.

2. LITERATURE REVIEW

Performance testing and capacity planning activities have always been key areas in the domain of enterprise system engineering.

However, their significance has dramatically escalated with the development of cloud-native architectures. Where earlier single-piece monolithic releases were the norm, today we have microservices that are only loosely coupled and each is running on a platform that has dynamic scheduling, autoscaling, service discovery, and layered networking features. These characteristics serve to increase agility; however, they also confound us with performance behavior which becomes more complicated and less predictable. In cloud-native settings, performance problems frequently arise not only from the application code but also from platform components such as ingress controllers, container runtimes, overlay networks, control planes, and shared observability pipelines. Consequently, both academic research and industrial references are pointing out that performance testing should not be a part of the validation step only but rather it should be an ongoing activity that is cognizant of the platform and consistent with the actual workload behavior.

A large portion of prior work categorizes performance testing into distinct test types that serve different engineering goals. Load testing is a typical example focused on checking the normal behavior of the system with various measurable factors of the workload such as throughput, latency percentiles, and error rates. Stress testing keeps increasing the workload until the system breaks and points out the first resource or component that becomes a bottleneck. Soak testing (or endurance testing) is a long stability test that characterizes the behavior of the system in terms of memory leaks, resource exhaustion, thread pool depletion, and gradual performance degradation over several hours or days. Spike testing is concerned with abrupt increases in traffic and assesses how fast the platform and application can recover and if the autoscaling mechanisms can adjust on time.

3. PROPOSED METHODOLOGY

3.1. METHODOLOGY OVERVIEW

This methodology framework offers a coherent, consistent process to the engineering of scalable enterprise container platforms mainly OpenShift and Pivotal Cloud Foundry (PCF). It first articulates the performance and scalability objectives of the business which could be the desired throughput level, latency percentiles, error rates, and resource efficiency. Different performance measures (KPI) and Service Level Objectives (SLO) are identified so that testing results can later be quantified and used for decision-making. A workload profile that represents the behavior of end users, the nature of transactions, and the volume of traffic is conceptualized.

A comprehensive testing regime is then conducted to determine the response of the application and the platform to various load scenarios. The procedure uses the outcomes of the performance tests as an input to a capacity planning model which is a tool that helps teams to forecast the requirements of the platform, find out the limiting factors, and come up with an optimizing scale strategy that can be implemented before going live.

3.2. WORKLOAD CHARACTERIZATION

Workload Characterization is a crucial element in performance testing and capacity planning. It is well known that an enterprise application system may drastically change its behavior based on the request types, payload sizes, concurrency levels, and traffic bursts. Firstly, a critical aspect is determining the Peak Transactions Per Second (TPS) or Request Per Second (RPS) besides the average load and the worst-case spikes. The term concurrency can refer to concurrent users, concurrent sessions, or concurrent API requests, depending on the application model. Difficulty Factors, as well as payload size, not only influence the situation significantly since bigger payloads require more CPU processing, memory allocation, and network I/O. The API call mix is later examined, among other things, to find out the percentage of operations that have been read or written, the calls for authentication, the dependencies on external services, and the background jobs. In order to imitate user behavior more naturally instead of continuous request flooding, "think time" is added. To create this workload profile, production logs and monitoring data are undoubtedly the most suitable sources. If production data is not available, synthetic assumptions are made based on domain knowledge, however, these assumptions are thoroughly documented and later checked. Finally, workload profiles are divided into normal traffic, bursty traffic, and seasonal demand so that the tests represent real-world enterprise usage scenarios.

4. CASE STUDY

4.1. SCENARIO DESCRIPTION

In order to demonstrate the correctness of our approach under a real business scenario, we decided to do a case study of a digital transaction platform having the combined workload of an e-commerce and banking API. The platform was composed of various microservices, each responsible for handling a different aspect such as user authentication, product/account queries, transaction processing, and notification delivery. Consequently, the identical application stack was first launched on two leading enterprise

platforms, Red Hat OpenShift and Pivotal Cloud Foundry (PCF), with the objective of comparing their scaling features and capacity output at the similar demand levels.

The workload pattern was a mixture of read-heavy and write-heavy API calls, and the peak events in the profile were interpreted as flash sales or payroll transaction bursts. The amount of traffic was modeled at various levels from 5,000 to 20,000 requests per second (RPS). The concurrency was adjusted to show actual user sessions rather than artificially generated such flooding. Performance goals were mostly directed at the ability to handle peak throughput and at the same time respond to latency SLOs, keep error rates very low, and demonstrate constant scaling behavior during ramp-up and burst scenarios.

4.2. ENVIRONMENT SETUP

The OpenShift environment was set up as a multi-node cluster that reflected the standards of an enterprise production. There was one control plane that was fully dedicated and a worker pool that was prepared for the application workloads. Scalability would be easier to analyze as the worker nodes have the same CPU and memory size. OpenShift routing was accessible through the default ingress/router level, and cluster networking was accomplished via standard CNI settings. The OpenShift built-in monitoring stack (Prometheus, Alert manager, and Grafana) was enabled to collect both the platform-level and application-level metrics, thus providing thorough oversight. In addition, a few other dashboards were created to check out latency percentiles, request throughput, CPU throttling, memory pressure, and pod scheduling delays.

The PCF environment was constructed using a standard foundation layout. Applications were run in Diego cells that were large enough to hold multiple container instances per single cell. Request routing and load balancing were done by gorouter instances, and the platform services consisted of UAA user authentication and Loggregator for collecting logs and streaming metrics.PCF Metrics was used to show the status of the applications, the activities of the router, and the consumption of container resources.

5. RESULTS AND DISCUSSION

5.1. KEY FINDINGS

Results from the performance tests revealed that both OpenShift and PCF are capable of handling very high throughput workloads, however, their performance under load varied in significant and operationally impactful ways. In OpenShift, latency kept on rising slowly throughout the incremental load tests until it essentially reached a limit, at which time p95/p99 latency shot up very quickly due to CPU throttling and routing saturation. Provided that the Kubernetes scheduler and Horizontal Pod Autoscaler (HPA) are properly set up, they will respond as expected; however, the quickness of scaling response is very much dependent on how stable the metrics are and on whether there are available nodes. If Cluster Autoscaler had to intervene and add new nodes, then scaling delays were noticeable and this was especially the case during abrupt spikes when the pods could not be scheduled at the required speed.

For continuous ramp-up situations, PCF demonstrated that instance scaling could be made more flexible, especially when Diego cells had sufficient resources. It was very easy to scale instances, and throughput improvements were very consistent. However, when the shared platform components center (Gorouter and UAA) was reaching its capacity, latency spikes were noticed. Compared with OpenShift, PCF routing was more centralized, thus making scaling and tuning of the router an essential task much sooner in the load curve.

Essentially,OpenShift delivered more flexible autoscaling capabilities such as custom metrics and very detailed pod-level scaling; however, it needed to be adjusted more to avoid noisy metrics and throttling effects. PCF offered a more opinionated scaling experience with quicker operational simplicity but showed a slightly stronger dependency on platform component sizing. The above conclusions show that platform maturity is not only a matter of autoscaling features but also of how routing, identity, and platform services perform under heavy concurrency.

5.2. CAPACITY PLANNING OUTCOMES

Then, capacity requirements were determined from performance measures and converted into firm sizing proposals for both platforms. Four latency percentiles were checked and that helped to establish at what load level the pod CPU utilization was so high that latency went beyond their defined SLO. Hence that number of pods to support the peak load (20k RPS) was counted. Afterwards, there was extra capacity for failover situations and traffic spikes. The model was also able to find out how many nodes

are necessary by changing the resource requests of the pods into worker node capacity, and this was done by taking the platform overhead into consideration and by reserving about 30% for additional capacity. Since routing saturation happened even before the computer exhaustion in some tests, router scaling requirements were also treated as a separate factor. The capacity model of PCF focused on application instances and Diego cell sizing. A series of incremental load tests were used to identify the top throughput of an instance, which in turn helped to calculate the number of instances that are required to handle the peak traffic. The requirements for Diego cells were derived from instance memory and CPU usage, which ensured that the system wouldn't reject instances during scaling events. Moreover, the tests discovered that router scaling needs to be very carefully planned as the router throughput limits influenced the latency and error rates even when there was still enough compute capacity available.

Each platform exhibited different kinds of tradeoffs in their cost-to-performance ratio: the slight overprovisioning of resources reduced the risk of latency, but increased the cost of the infrastructure, while aggressive autoscaling helped to keep the baseline cost low, but at the risk of not being able to scale quickly during the spikes. In the end, the choices were oriented towards maintaining a balance between steady-state efficiency and peak-load reliability.

6. CONCLUSION AND FUTURE SCOPE

6.1. CONCLUSION

In this article, we presented a comprehensive step-by-step framework for conducting performance testing and capacity planning of container enterprise platforms, with OpenShift and Pivotal Cloud Foundry (PCF) being the focus. Combining workload characterization, incremental performance testing, and a quantifiable capacity planning model, the approach enables the team to determine scale limits, validate SLOs, and locate bottlenecks in performance before the production release. The case study reveals that while both platforms are able to process a very high level of throughput, their scaling limits.

One of the main points brought out through the performance results is that performance engineering must be a continuous process tightly integrated with product release cycles and operational planning, not merely a final-stage validation. The proposed framework is therefore one that can be easily replicated, takes the platform into account, and is feasible considering real-life enterprise projects where cost, reliability, and scalability are very important factors.

6.2. FUTURE SCOPE

In the future, this work can be extended by starting AI-driven capacity planning that trains itself through learning historical traffic, resource usage and incident patterns to forecast demand more accurately. Predictive autoscaling models might help in scaling lag reduction during spikes by load anticipation instead of load reaction. Chaos engineering integration would, no doubt, add another layer of resilience testing by considering the recovery from failures under the load conditions typical of the production environment. Moreover, service mesh performance impacts (Istio/OSSM) especially for latency-sensitive microservices, should also be studied further. More importantly, enterprises nowadays deploy multi-cluster and hybrid-cloud environments. Hence, cross-cluster capacity planning and global scaling strategies are undoubtedly the main directions for future work.

REFERENCES

[1] Gudi, Srikanth Reddy. "A Comparative Analysis of Pivotal Cloud Foundry and OpenShift Cloud Platforms." Emerging Frontiers Library for The American Journal of Applied Sciences 7.07 (2025): 20-29.
[2] Timilehin, Oladoja. "Performance Engineering for Hybrid Multi-Cloud Architectures: Strategies, Challenges, and Best Practices." (2024).
[3] Winn, Duncan CE. Cloud Foundry: the cloud-native platform. " O'Reilly Media, Inc.", 2016.
[4] Lomov, Alexander. "OpenShift and cloud foundry PaaS: high-level overview of features and architectures." white paper, Altoros (2014).
[5] Pereira, Carlos Diego Cavalcanti. "Capacity Planning of Cloud Computing Workloads." (2025).
[6] Suleiman, Nadia, and Yusuf Murtaza. "Scaling microservices for enterprise applications: Comprehensive strategies for achieving high availability, performance optimization, resilience, and seamless integration in large-scale distributed systems and complex cloud environments." Applied Research in Artificial Intelligence and Cloud Computing 7.6 (2024): 46-82.
[7] Dakić, Vedran, Mario Kovač, and Jurica Slovinac. "Evolving High-Performance Computing Data Centers with Kubernetes, Performance Analysis, and Dynamic Workload Placement Based on Machine Learning Scheduling." Electronics 13.13 (2024): 2651.
[8] Vitui, Arthur Marius. Automation And Intelligence In IT Operation Management: Machine Learning for Capacity Planning and Load Testing Optimization. Diss. Concordia University, 2025.
[9] Fontana, Giovanni, and Rafael Pecora. OpenShift Multi-Cluster Management Handbook. 2022.
[10] Caban, William. Architecting and Operating OpenShift Clusters: OpenShift for Infrastructure and Operations Teams. Apress, 2019.

[11] Mathur, Prateek. "Cloud computing infrastructure, platforms, and software for scientific research." High Performance Computing in Biomimetics: Modeling, Architecture and Applications (2024): 89-127.
[12] Chowdary, Mandepudi Nobel, et al. "Automated pipeline for the deployment using openshift." Procedia Computer Science 215 (2022): 220-229.
[13] Kumar, E. Saravana, et al. "Comparative study and analysis of cloud container technology." 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2024.
[14] Elder, Michael, Jake Kitchener, and Brad Topol. Hybrid Cloud Apps with OpenShift and Kubernetes. " O'Reilly Media, Inc.", 2021.
[15] Raj, Pethuru, Skylab Vanga, and Akshita Chaudhary. Cloud-Native Computing: How to design, develop, and secure microservices and event-driven applications. John Wiley & Sons, 2022.
[16] Reddy, K. K., Gunupati, K., Kumar, M., Reddy, P. R. R., Julakanti, R., & Jonnalagadda, R. R. (2025, September). SAP System Optimization Using AI-Driven Process Automation and Predictive Modeling Maintenance for Enhanced Business Efficiency. In 2025 International Conference on Computing and Communications (COMPUTINGCON) (pp. 1-6). IEEE
[17] PellReddy, R. (2024). Empowering cloud security: Pioneering an interactive multi-factor authentication framework for cloud user verification.
[18] Gadhiya, Y., Gangani, C. M., Sakariya, A. B., & Bhavandla, L. K. The Role of Marketing and Technology in Driving Digital Transformation Across Organizations. Library Progress International, 44 (6), 20-12.
[19] Tirumalasetty, P. (2025). Deep Graph Learning for Autonomous Data Reconciliation Across Heterogeneous Enterprise Systems.
[20] Vamshidhar Reddy Vemula.(2023).Multi-Cloud Security Orchestration Using Deep Reinforcement Learning.