broadleaf
commerce

[scal·a·bil·i·ty]

noun

the capacity to be changed in size or scale; the ability of a computing process to be used or produced in a range of capabilities.

A Study Testing BLC Infrastructure Simulating Real-World Shopping Scenarios

Table of Contents

Meet the Author

JEFF FISCHER

Jeff Fischer is Chief Technology Officer and Board member at Broadleaf Commerce, responsible for Technological Development, Digital Innovation Strategy, Data Protection, and Executive oversight on Technology Implementations. Jeff has 25 years of experience across Digital Commerce, Software Engineering, and Technical Consulting.

Introduction

Broadleaf Commerce (Broadleaf) provides companies with a platform for building high performance commerce solutions. Based on best of breed open source technologies including the Spring Framework, Broadleaf was designed from the ground up to be extensible and scalable for businesses and institutions requiring a mission critical eCommerce solution.

With a robust microservices architecture, Broadleaf embraces the latest thought in software architecture principles and practice. Coupled with our innovations in advanced composable commerce, Broadleaf provides a forward-thinking architecture that can fit into any infrastructure budget.

This paper provides details of scalability tests performed with the Broadleaf Commerce Microservices reference implementation. Testing was completed using a range of node size combinations in a standard kubernetes cluster in the cloud. With the ability to easily scale to thousands of transactions per second across tens of thousands of concurrent users and millions of products, the test results speak for themselves.

Note, for details on configuration and test plan specifics, refer to the appendices at the end of the document.

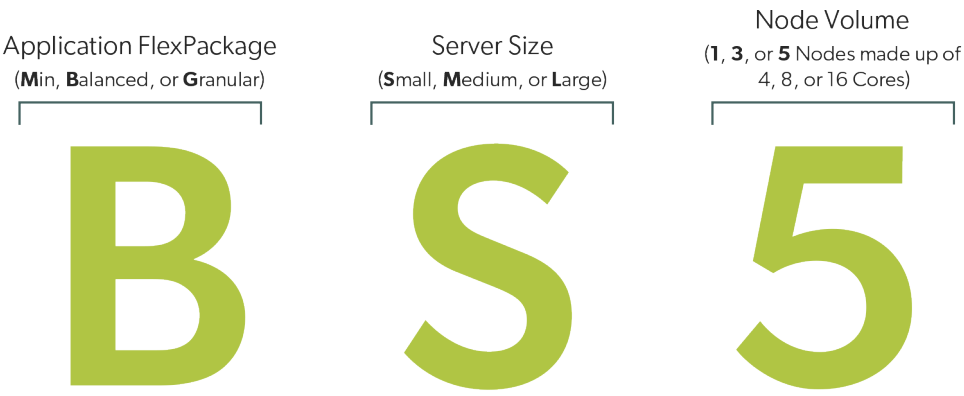
Section 1

Test Methodology

Simulating real-world shopping scenarios with industry average conversion rates

How to read the charts in this study

For this paper, we have employed a naming scheme that combines application and infrastructure sizing into a shortened name for easy reference. Throughout this document, you will see chart references to items such as "BS3", or "GL5". Refer to the chart below to understand how these terms relate to application and infrastructure resources.



Application FlexPackage[1]	Kubernetes Infrastructure	Label
Min	**S**mall; **1** 4-core Node	**MS1**
Balanced	**S**mall; **3** 4-core Nodes	**BS3**
Balanced	**S**mall; **5** 4 -core Nodes	**BS5**
Balanced	**M**edium; **3** 8-core Nodes	**BM3**
Balanced	**M**edium; **5** 8-core Nodes	**BM5**
Balanced	**L**arge; **3** 16-core Nodes	**BL3**
Balanced	**L**arge; **5** 16-core Nodes	**BL5**
Granular	**L**arge; **3** 16-core Nodes	**GL3**
Granular	**L**arge; **5** 16-core Nodes	**GL5**

[1] See the "Considerations" section (specifically "Cost" and "Culture") for more details on flex packages.

Simulating real-world scenarios when testing an e-commerce application is critical in determining not only how efficiently the software performs under normal circumstances, but also how many users it can serve during peak demand. While testing up to a 100% conversion rate to ensure performance on “Black Friday” and “Cyber Monday” behavior was accounted for, most test cases conducted an average 10% add-to-cart action and an aggregated 3% conversion rate.

In all cases, Broadleaf set out to report objective scalability numbers. In isolation, test cases for “home page views” or “orders” have no merit outside of simulated consumer behavior. The ability for a system to handle hundreds of millions of views to a single page without any other variable is a useless statistic in itself. Furthermore, test cases with varying concurrent user numbers hold no value unless tested against concurrent user behavior.

A test was considered “passing” if the Broadleaf framework generally responded to API calls with an average response time of 500 ms, or less. Furthermore, real world usage of microservice APIs often require several calls to fulfill a concept, such as rendering a product detail page. In these cases, we also measured the aggregate response time of all API calls involved to complete the overall user experience and confirmed the sum of timings was generally less than 1 second.

Finally, test cases were given a several minute warm up time followed by a 2 minute ramp-up period before experiencing peak load. Refer to Appendix C for more details on the approach.

Note on nomenclature - For this paper, we have employed a naming scheme that combines application and infrastructure sizing into a shortened name for easy reference. Throughout this document, you will see chart references to items such as “BS3”, or “GL5”. Refer to the chart below to understand how these terms relate to application and infrastructure resources.

Section 2

Test Results

Against high traffic, large product catalogs, and peak season spikes, Broadleaf Commerce proves the ability to handle the most stringent scalability requirements.

1. Throughput

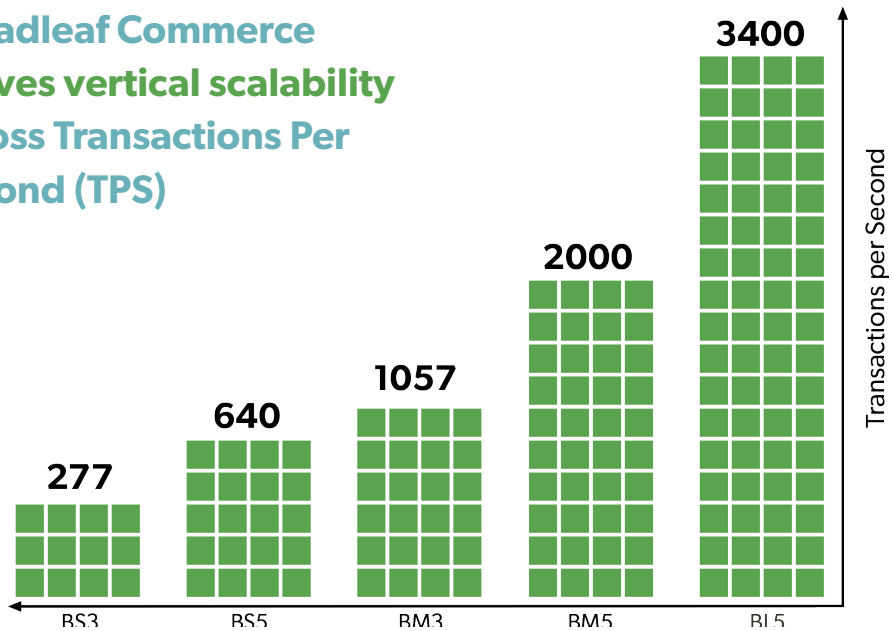
Peak demand is defined based on industry, customer base, and seasonality. Through all major peak eCommerce variables, Broadleaf proves the ability to scale. Across high transaction volume, concurrent users, large catalogs, and high conversion rates, Broadleaf exhibits consistent peak performance.

For test purposes, Broadleaf focuses on throughput as it relates to completed transactions per second (TPS), and completed orders per second (OPS). A transaction in this context refers to an individual microservice API call to complete a unit of functionality as it relates to the overall ecommerce test plan. Refer to Appendix A for more details on the test plans used. By adding replicas of application microservices to a growing kubernetes cluster, we demonstrate fairly linear scale as we seek to increase throughput.

The initial test focuses on the classic heavy browse use case culminating in a 3% conversion rate to completed orders. Such a case involves significant user activity in the areas of search, catalog browsing, and registered customer login and account perusal. A 10K product catalog is used in all cases, unless otherwise specified.

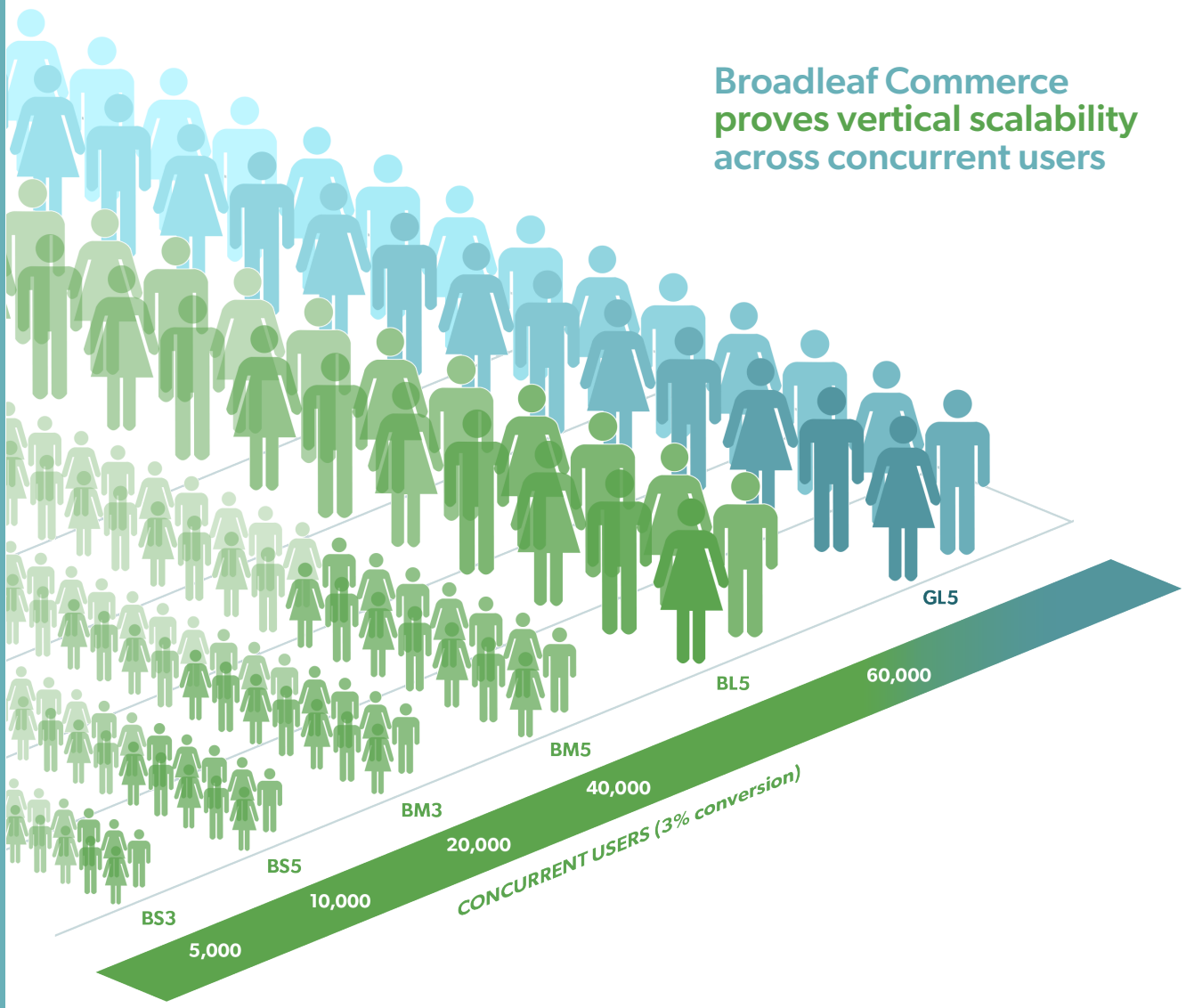
In this round of tests, the Balanced flexpackage throughput was examined through small, medium, and large infrastructure configurations, culminating in 3400 TPS for the system. The Granular flexpackage was also considered at the larger infrastructure sizes and performed comparably. Spoiler alert - for extreme scale, refer to the Conversion Rate results section, where we demonstrate scaling to 100 orders per second using conventional techniques alone.

Broadleaf Commerce proves vertical scalability across Transactions Per Second (TPS)



2. Concurrent Users

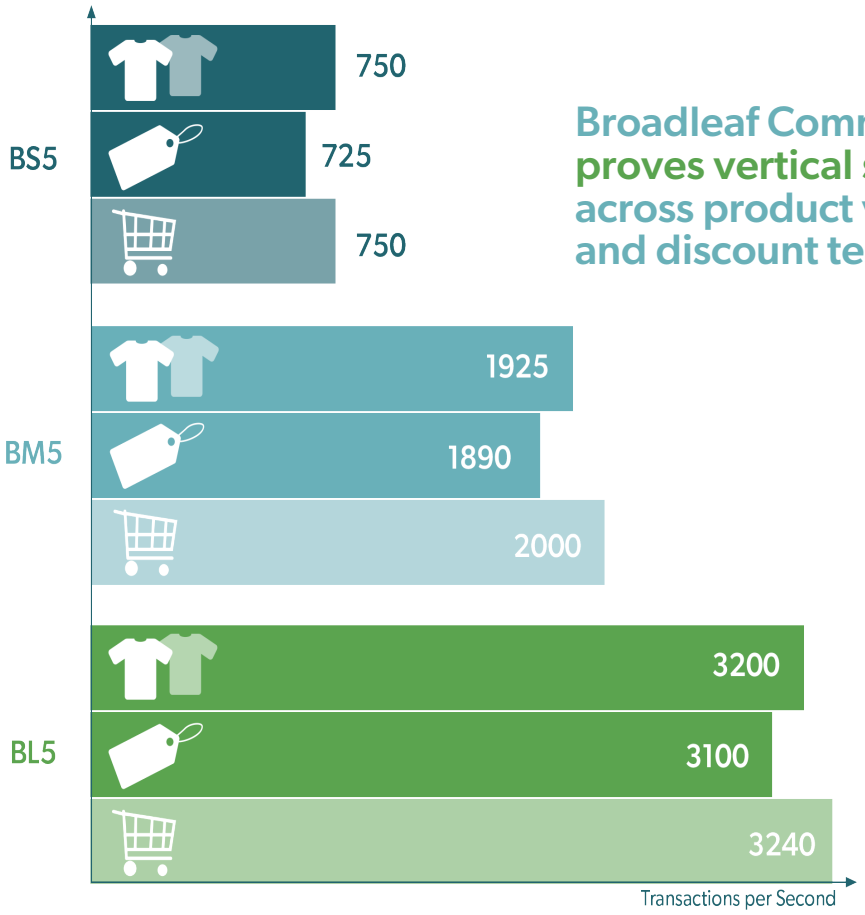
In order to simulate concurrent users in a real world scenario, Broadleaf tested virtual users in gated test plans involving percentage advancement at various stages of the customer journey, coupled with quantity and catalog choice randomization. Then, calculating based on an ecommerce average of 1 page view perminute per user, we were able to estimate concurrent user capacities.



For companies needing to accommodate even more users (e.g., companies that want to be the next Amazon, Facebook, or Twitter), infrastructure build out and serious application performance tuning can be handled with Broadleaf's Professional Services. For most businesses with typical conversion rates, the numbers demonstrated above can handle sites generating billions of dollars in sales.

3. Data Variations

While the 3% conversion rate test plan is interesting on its own, it is important to consider the impact of catalog complexity. In this round of tests, we introduce two factors: discounts and variants/options. Discounts can slow down cart manipulation as it introduces additional calculation required to price items in the cart. Variants (aka Skus) and product options also represent additional catalog complexity for the system to navigate as it addresses inventory and pricing concerns across a larger landscape of customer choice.



Broadleaf Commerce proves vertical scalability across product variations and discount testing

Checkout without Variants or Discounts

Discount Test

100 BOGO offers were introduced into the system - all set to auto apply. This setting indicates that all offers are qualified against the cart contents at the time of add to cart, which is a key computational moment in the cart lifecycle. The effect was a little more noticeable in this case, but overall still not an unfavorable impact at any size tested.

Variants Test

Two different tests were run. First, 3 variants per product were introduced with a basic set of product options for the test to negotiate upon add to cart. When compared to normal execution (no variants), the impact was negligible, within a margin of error.

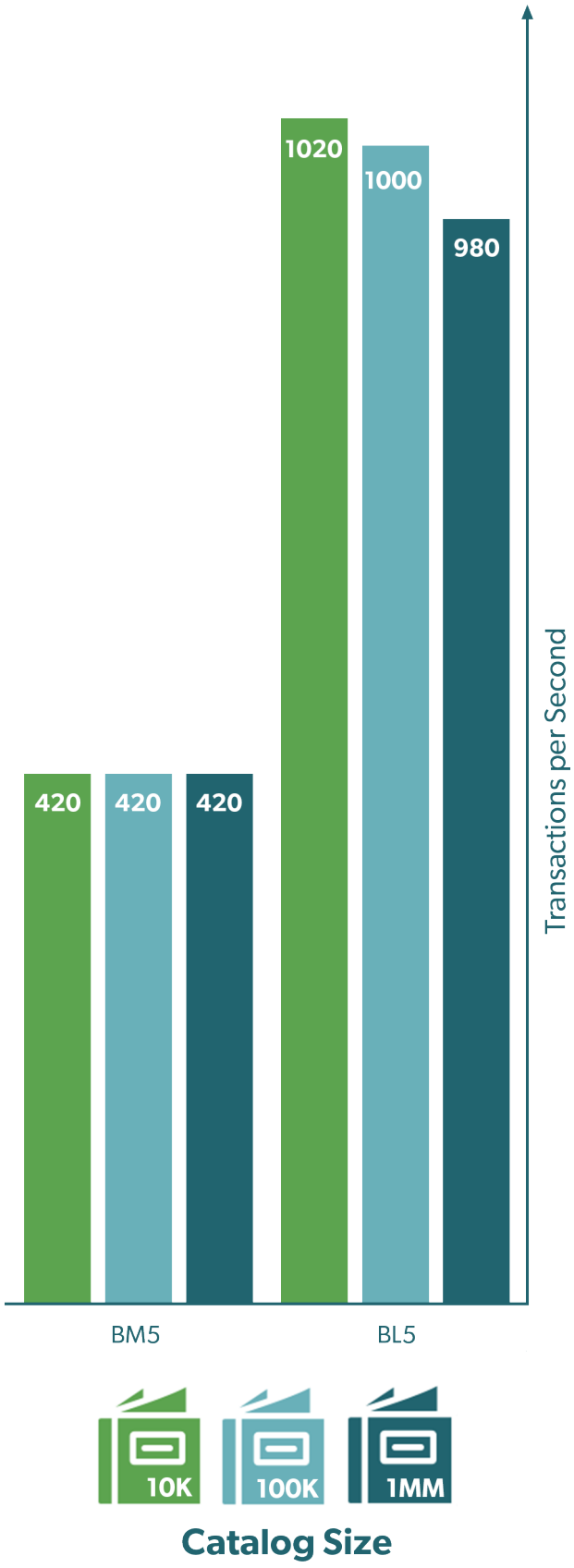
4. Catalog Size

For corporations requiring larger catalog sets, Broadleaf tested an online catalog with 100,000 and 1,000,000 products.

Broadleaf Commerce demonstrated the ability to handle wide variations in catalog size with negligible impact.

This test focuses on the order life cycle journey using a test plan starting with the product detail page and ending in cart checkout. This test plan is similar to the 100% conversion rate test plan detailed later in this document.

Again, we see a negligible impact to performance based on catalog size across all infrastructure combinations.

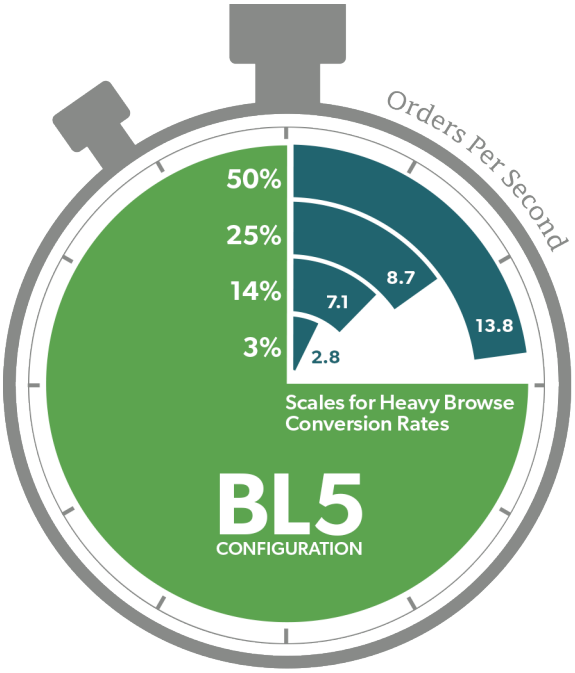


5. Conversion Rates

For corporations with increased conversion rates above industry averages, Broadleaf tested up to 100% conversion. The key metric captured for this test is orders per second (OPS).

At the highest tier tested (BL5) using a heavy browse test plan (similar to the standard 3% conversion test plan), Broadleaf demonstrated a volume of 13.83 OPS at a 50% conversion rate. This translates into about 50K order per hour.

This type of traffic may be common for some retailers during Black Friday style shopping scenarios.



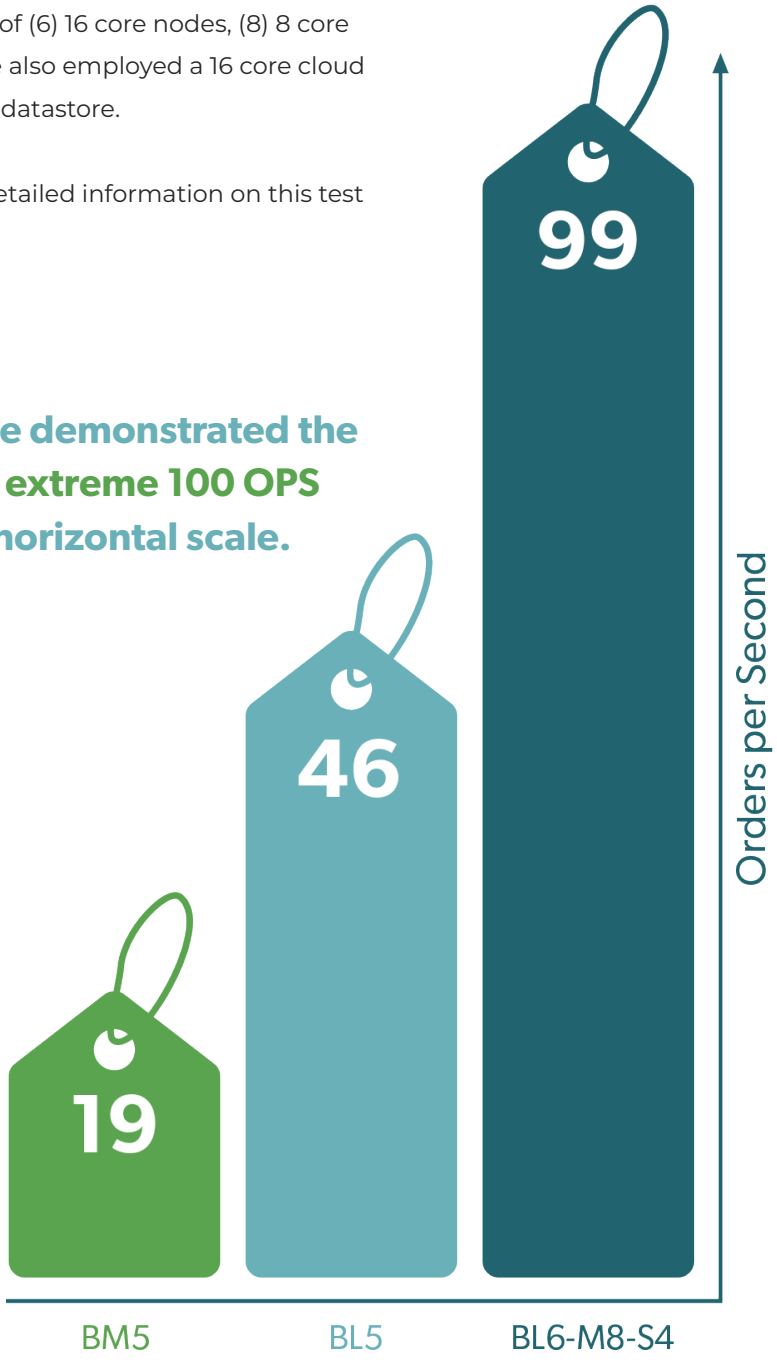
Given an industry standard average order value (AOV) of $128, Broadleaf Commerce can comfortably support billions in revenue on a reasonable hardware budget.

Checkout focused results are often requested as well. This type of test plan lightens the browse requirements, and instead opts for an order focused journey. This solely considers the product detail page, and beyond to checkout and order completion - at 100% conversion rate. This type of flow is common with highly targeted product offerings, usually with a small product count.

We were able to demonstrate the extreme case of 100 OPS using conventional horizontal scale techniques. For this high end case, we utilized a cluster composed of (6) 16 core nodes, (8) 8 core nodes, and (4) 4 core nodes. We also employed a 16 core cloud native database as the backing datastore.

Refer to Appendix A for more detailed information on this test plan.

Broadleaf Commerce demonstrated the ability to scale to an extreme 100 OPS using conventional horizontal scale.



Section 3

Considerations

There are plenty of options that can assist with achieving a Broadleaf installation that is optimized for performance and cost.

Cost

Infrastructure cost is a major consideration when determining the right platform to leverage for your eCommerce solution. The right balance of infrastructure spend weighed against throughput and revenue expectations is important to estimate carefully ahead of time. To further complicate the issue, modern architecture best practices call for a microservice approach, which affords many benefits, but tends to come at a higher infrastructure cost to support the granular service count.

Broadleaf understands these challenges and has come up with an innovative approach to address these issues, without inhibiting future growth. Broadleaf has introduced an infrastructure composability concept entitled FlexPackages.

A FlexPackage is a unit of composition that conserves all of the basic plumbing and components that make up the base of the microservice platform.

Configured on top are the unique components and API that inhabit individual, granular microservices. By putting multiple microservices together into a single FlexPackage, you can conserve greatly on the complexity and quantity of infrastructure needed to support the stack, while at the same time completely honoring bounded context restrictions and design boundaries. The persistence tier is still isolated per microservice, and the endpoint APIs and asynchronous messaging tier are solely used for inter-service communication.

Broadleaf currently ships in the neighborhood of 25 granular microservices. However, we also expose configuration for the “Balanced” flexpackage that combines these microservices into 4 primary components: Cart, Browse, Supporting, and Processing. Doing so, we are able to comfortably realize smaller infrastructure deployments that would not be feasible with that many individual microservices. And, via configuration alone, the microservices can be combined into other combinations, or broken apart completely into the original, granular representations. This is a powerful feature allowing you to model your infrastructure to match your revenue growth, or IT culture.

We call this Advanced Composable Commerce, as the FlexPackage allows you to model multiple vectors of composability.

With microservices, you can choose the features and functions that make sense for your business. With FlexPackages, you can extend that flexibility into the infrastructure domain.

You can also consider savings across multiple environments. For example, with the same codebase, you can package differently for dev, qa, and prod. The possibilities are limitless.

Refer to Appendix D for more information on the default FlexPackages used, and the components they contain.

Culture

IT Culture is another factor, in addition to cost, that can influence decisions regarding FlexPackage choice. Team structures that honor traditional microservice isolation boundaries are likely to favor the most granular representation of each microservice. This type of team tends to operate in a silo and often ships code with a higher frequency - shortening overall time to value. On the other hand, multi-discipline teams may choose to cover multiple bounded contexts and favor shipping less often, or with reduced devops complexity, by reducing microservice count through FlexPackage composition.

There are pros and cons with both extremes, and there are a spectrum of levels between.

The reality is that IT culture doesn't always match up nicely with technical innovation, and the FlexPackage concept can help soften the devops transition.

If organizations choose to adopt more devops complexity, the FlexPackage pattern can help with that transition without requiring code refactoring.

Caching

Broadleaf leverages the Spring Cache abstraction in key flows for performance benefit. The out-of-the-box implementation we employ leverages Apache Ignite. By using Ignite, we get the benefit of a robust cache architecture, including its features around off-heap cache. By storing cache members off-heap, the burden on the JVM to garbage collect frequent evictions is removed. This benefits the overall GC posture of the application and further stabilizes performance.

With default settings, we find that Ignite allocates immediately 256 MB of off-heap memory as a bucket for cache. During our testing, with the caches we provide out-of-the-box, we never exceeded that initial bucket. In fact, we routinely only used about 64 MB of it. Note that each microservice runtime with cache enabled will allocate off-heap memory in this way. While small, this should be accounted for when considering pod scheduling and node capacity.

By default, we configure a basic TTL cache in a non-distributed configuration. This means that each microservice instance maintains its own copy of cache. This is the least blocking configuration and is something we generally favor for performance, at the cost of a delay in eviction in most cases. Ignite is flexible and is designed to be used as a distributed cache if needed. If more real time consistency is required, the cache can be configured in this manner, possibly at some throughput cost.

Auto Scale

Auto scale can be an effective mechanism for meeting occasional demand increase without keeping peak infrastructure available 24/7.

By automatically scaling to need, you can decrease overall cost without sacrificing customer experience.

Kubernetes provides auto scale features at both the pod and node level. As pod utilization meets configured thresholds, kubernetes can spawn new replicas to absorb the increase in demand. Furthermore, as pod count exceeds capacity, kubernetes can spawn additional nodes in the current node pool to create additional capacity for the pod increase. If enabled, kubernetes will continue to do so until it reaches its configured limit. This configuration is enabled at the point where the infrastructure is provisioned. If using Terraform as we did, it is a setting for the resources in the Terraform template file.

There are several factors to consider for auto scale. Broadleaf leverages Java and Spring. There is a startup time cost from initial application launch to the time when the application is ready to receive connections. Compound that with the time kubernetes requires to spawn a new pod. Further compound that with the time kubernetes requires to spawn a new node (this latter point is the most costly of the three). A new node is not always required if there is already enough capacity to handle the new pod Further compound that with the time kubernetes requires to spawn a new node (this latter point is the most costly of the three). A new node is not always required if there is already enough capacity to handle the new pod. Nonetheless, there is a real time-to-effectivity cost that must be taken into account when considering auto scale. You should not expect instant availability.

There are a number of best practices for configuring the kubernetes cluster autoscaler, horizontal pod autoscaler, and vertical pod autoscaler that can be found online. I won't detail them all here. However, it is useful to consider demand scenarios and autoscale expectations. If you have ebb and flow of fairly gradual demand, the system will still provide acceptable customer experience under increased demand during the period in which the auto scale process is enacted. If you experience incredible spikes in traffic, you may outpace the scale-up timeline. In such a case, you should maintain additional hot resources to accommodate the spike, or pre-emptively scale up for a temporary timeframe if the spike can be predicted.

Database

The tests performed in this paper were all executed against a Postgres database (specifically a Google Cloud SQL instance provisioned outside the kubernetes cluster). While Broadleaf also supports Mysql, MariaDB, and Oracle, our reference implementation leverages Postgres.

We found the cloud sql implementation to be highly efficient, roughly equivalent to a similar database installed directly in the cluster. We also found Broadleaf's usage of database resources to be highly efficient, contributing to an overall performance benefit. In general, the connection pool configured for the application was set to a count of 10. The only exception was the 100 OPS test, which used a connection pool size of 20. During test monitoring, we never found the active usage to exceed that threshold, with little or no blocking at acquisition.

We also found that database vertical scale requirements were minimal as the application itself scaled. Most of the test cases required only an 8 or less CPU instance for the database. The exception was the 100 OPS extreme test, which used a 16 core instance. However, at that scale, that is a very acceptable sizing.

For tuning the database itself, throughput levers for cloud native instances tend to be governed by sizing. Factors such as storage size, CPU count, and RAM can affect disk IO quota, network throughput, and max connection count, respectively.

CDN

The load test did not retrieve graphics or other static assets from the application container, nor did it engage the built-in asset server for any managed asset retrieval. Most high volume sites will benefit from having static assets delivered to users via a CDN (Content Delivery Network). CDN solutions offer hi-speed nodes located across the globe with delivery of your assets coming from the nodes closest to a given user. This serves to reduce response times for your application and lessens unnecessary load on your application container.

Third Party Integrations

Typical enterprise eCommerce systems can contain ten or more integrations. Each integration has the potential to negatively affect the scalability of the system. It is important to use best practices for integrating with other systems to ensure that one poorly performing third party integration does not bring down the entire system.

Tuning strategies, including cache and circuit-breaker patterns, can help to avoid hotspots and maintain acceptable customer experience.

Customizations

The load tests reported in this paper were performed against a reference implementation of the Broadleaf Microservice Framework. As such, there was no custom code included in the test outside of what Broadleaf itself provides. Client implementations built on top of the Broadleaf Framework will necessarily include additional customizations, libraries, and integrations that can possibly contribute to performance degradation. For this reason, each new implementation should itself go through a load test analysis to qualify it as meeting performance expectations. Should you need it, Broadleaf provides professional services to help with performance tuning your implementation.

Section 4

Conclusion

The Broadleaf Commerce framework scales across all testing metrics to meet the needs of even the most demanding eCommerce sites.

Well known retailers and businesses depend on Broadleaf Commerce to power their eCommerce solutions, as Broadleaf provides best-in-class eCommerce capabilities at the highest value.

Broadleaf proved real business use cases across multiple scenarios, demonstrating:

- **Thousands of transactions per second**
- **Tens of thousands of concurrent users**
- **Millions of products**
- **Billions of dollars worth of sales**

Furthermore, the provided test results demonstrate that Broadleaf Commerce scales horizontally by adding additional microservice instances. This type of scaling is ideal for cloud based environments, especially those leveraging kubernetes.

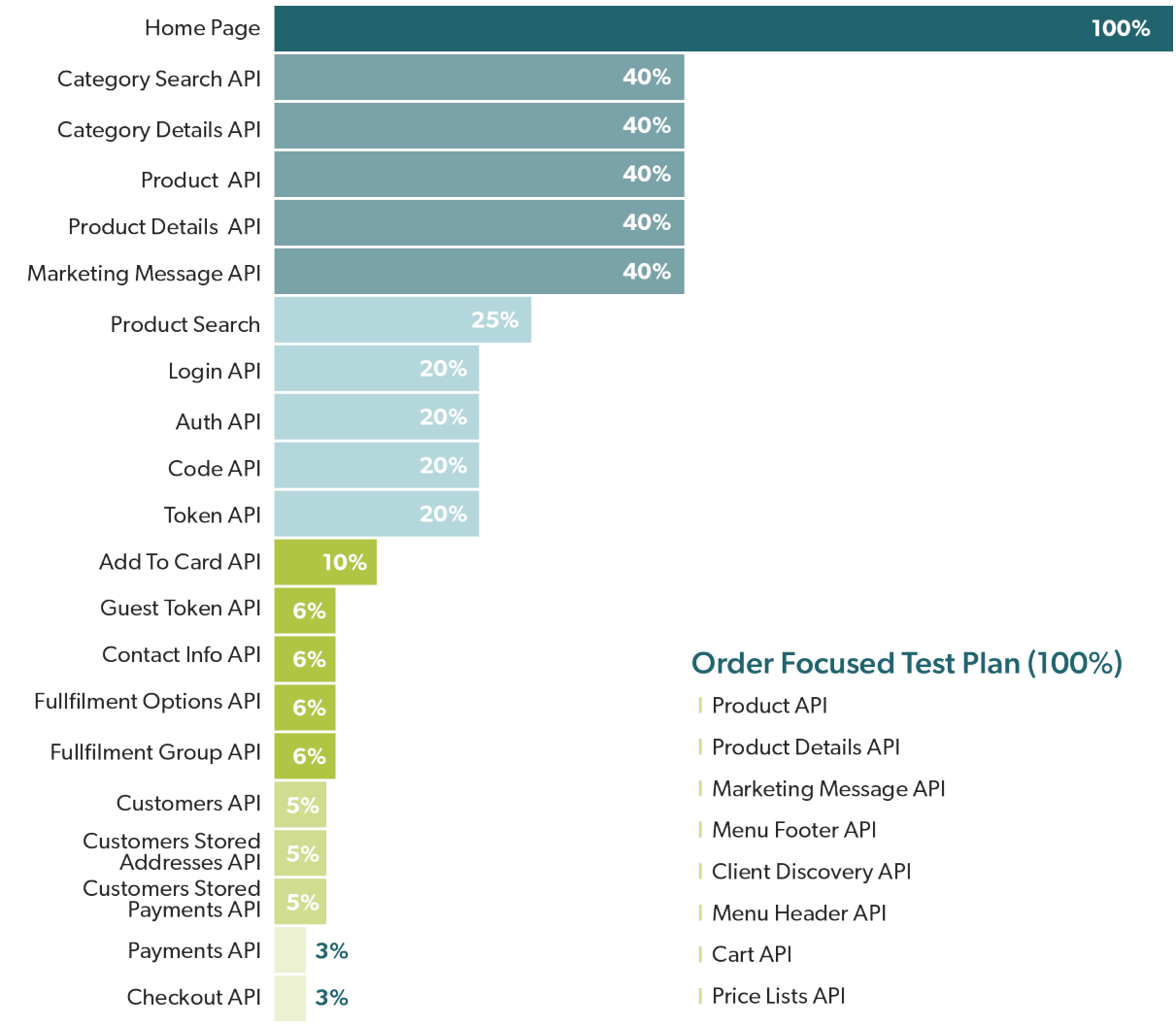
Finally, the results demonstrate that Broadleaf's FlexPackage technology enables deployment of a large microservice architecture on a hardware footprint that can fit into any budget.

For more information on Broadleaf, please visit: www.broadleafcommerce.com

Appendices

Appendix A - Test Plan

Standard, Browse Heavy Test Plan



Additional Considerations Impacting Multiple Factors

- Menu Foooter API
- Client Discovery API
- Menu Header API
- Cart API
- PriceLists API

Order Focused Test Plan (100%)

- Product API
- Product Details API
- Marketing Message API
- Menu Footer API
- Client Discovery API
- Menu Header API
- Cart API
- Price Lists API
- Add To Card API
- Guest Token API
- Contact Info API
- Fulfillment Group API
- Fulmillment Options API
- Payments API
- Checkout API

Appendix B - Throughput Reference

Type	Label	TPS	OPS	Node Specs	Theme
(M)in	S1	20	0.01	1 - 4Core	Browse Heavy, 3% Conversion
(B)alanced	S3	277	0.20	3 - 4Core, 2 Core DB	Browse Heavy, 3% Conversion
(B)alanced	S4	458	0.36	4 - 4Core, 2 Core DB	Browse Heavy, 3% Conversion
(B)alanced	S5	640	0.60	5 - 4Core, 2 Core DB	Browse Heavy, 3% Conversion
(B)alanced	M3	1057	0.81	3 - 8Core, 4 Core DB	Browse Heavy, 3% Conversion
(B)alanced	M5	2000	1.40	5 - 8Core, 4 Core DB	Browse Heavy, 3% Conversion
(G)ranular	L3	1900	1.50	3 - 16Core, 4 Core DB	Browse Heavy, 3% Conversion
(B)alanced	L3	2121	1.65	3 - 16Core, 4 Core DB	Browse Heavy, 3% Conversion
(B)alanced	L5	3400	2.50	5 - 16Core, 8 Core DB	Browse Heavy, 3% Conversion
(G)ranular	L5	3400	2.50	5 - 16Core, 8 Core DB	Browse Heavy, 3% Conversion
(B)alanced	S5	289	0.99	5 - 4Core, 2 Core DB	Browse Heavy, 14% Conversion
(B)alanced	M5	1092	3.35	5 - 8Core, 4 Core DB	Browse Heavy, 14% Conversion
(B)alanced	L5	2389	7.73	5 - 16Core, 8 Core DB	Browse Heavy, 14% Conversion
(B)alanced	M5	791	4.16	5 - 8Core, 4 Core DB	Browse Heavy, 25% Conversion
(B)alanced	L5	1864	9.21	5 - 16Core, 8 Core DB	Browse Heavy, 25% Conversion
(B)alanced	S5	550	0.60	5 - 4Core, 2 Core DB	Browse Heavy, 3% Conversion, Discounts
(B)alanced	M5	1700	1.20	5 - 8Core, 4 Core DB	Browse Heavy, 3% Conversion, Discounts

Appendix B - Throughput Reference

Type	Label	TPS	OPS	Node Specs	Theme
(B)alanced	L5	3000	2.40	5 - 16Core, 8 Core DB	Browse Heavy, 3% Conversion, Discounts
(B)alanced	S5	640	0.60	5 - 4Core, 2 Core DB	Browse Heavy, 3% Conversion, Product Options
(B)alanced	M5	1900	1.40	5 - 8Core, 4 Core DB	Browse Heavy, 3% Conversion, Product Options
(B)alanced	L5	3300	2.40	5 - 16Core, 8 Core DB	Browse Heavy, 3% Conversion, Product Options
(B)alanced	M5	498	4.33	5 - 8Core, 4 Core DB	Browse Heavy, 50% Conversion
(B)alanced	L5	1594	13.83	5 - 16Core, 8 Core DB	Browse Heavy, 50% Conversion
(B)alanced	M5	420	19.00	5 - 8Core, 4 Core DB	Browse Light, Small Cart, 100% Conversion
(B)alanced	L5	1040	48.00	5 - 16Core, 8 Core DB	Browse Light, Small Cart, 100% Conversion
(B)alanced	L6-M8-S4	2100	99.00	6 - 16Core, 8 - 8Core, 4 - 4Core, 16 Core DB	Browse Light, Small Cart, 100% Conversion
(B)alanced	M5	420	19.00	5 - 8Core, 4 Core DB	Browse Light, Small Cart, 100% Conversion, Large Catalog
(B)alanced	L5	1020	46.00	5 - 16Core, 8 Core DB	Browse Light, Small Cart, 100% Conversion, Large Catalog
(B)alanced	M5	420	19.00	5 - 8Core, 4 Core DB	Browse Light, Small Cart, 100% Conversion, XL Catalog
(B)alanced	L5	980	45.00	5 - 16Core ,8 Core DB	Browse Light, Small Cart, 100% Conversion, XL Catalog

Appendix C - Approach

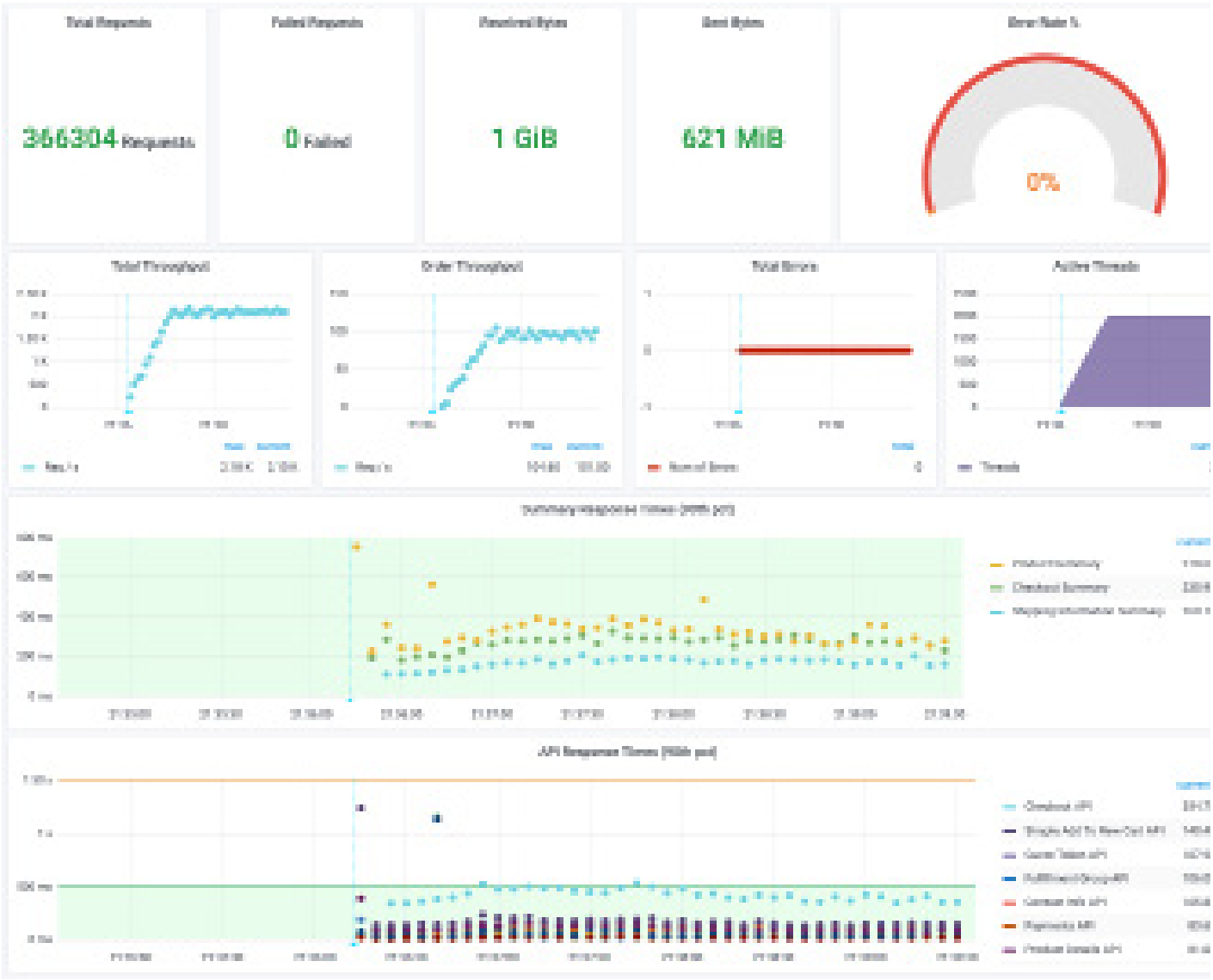
We use a custom JMeter rigging exposed via a web-interface delivered by a Spring Boot application. The rigging is deployed in a specific test node pool that is maintained separately from the main application node pool. In some circumstances, multiple node pools were used for the application pods. This latter case occurred when it was advantageous to segregate pods by type (e.g. a node pool dedicated to replicas of the Browse type). The primary vehicle for attaining this level of separation was to utilize node tainting and toleration configurations to force scheduling segregation.

To minimize cost, Terraform was configured to provision preemptible nodes. Helm Charts were used to deploy the application and support components to the kubernetes cluster. Shell scripts were used to interact with Terraform and Helm to automate the construction, installation, uninstallation, and destruction of the cluster. The overall goal was to minimize cluster uptime to reduce expense.

Monitoring system health at all levels was important during test runs. Distributed systems are complex with many moving parts, each contributing to system health and performance. We leveraged separate Grafana dashboards for JMeter (Figure 1) results and system health (Figure 2). Visualizations in both dashboards can evidence system stress and are useful for determination of where to scale resources and/or capacity. The system health dashboard is the same standard dashboard that ships with the framework.

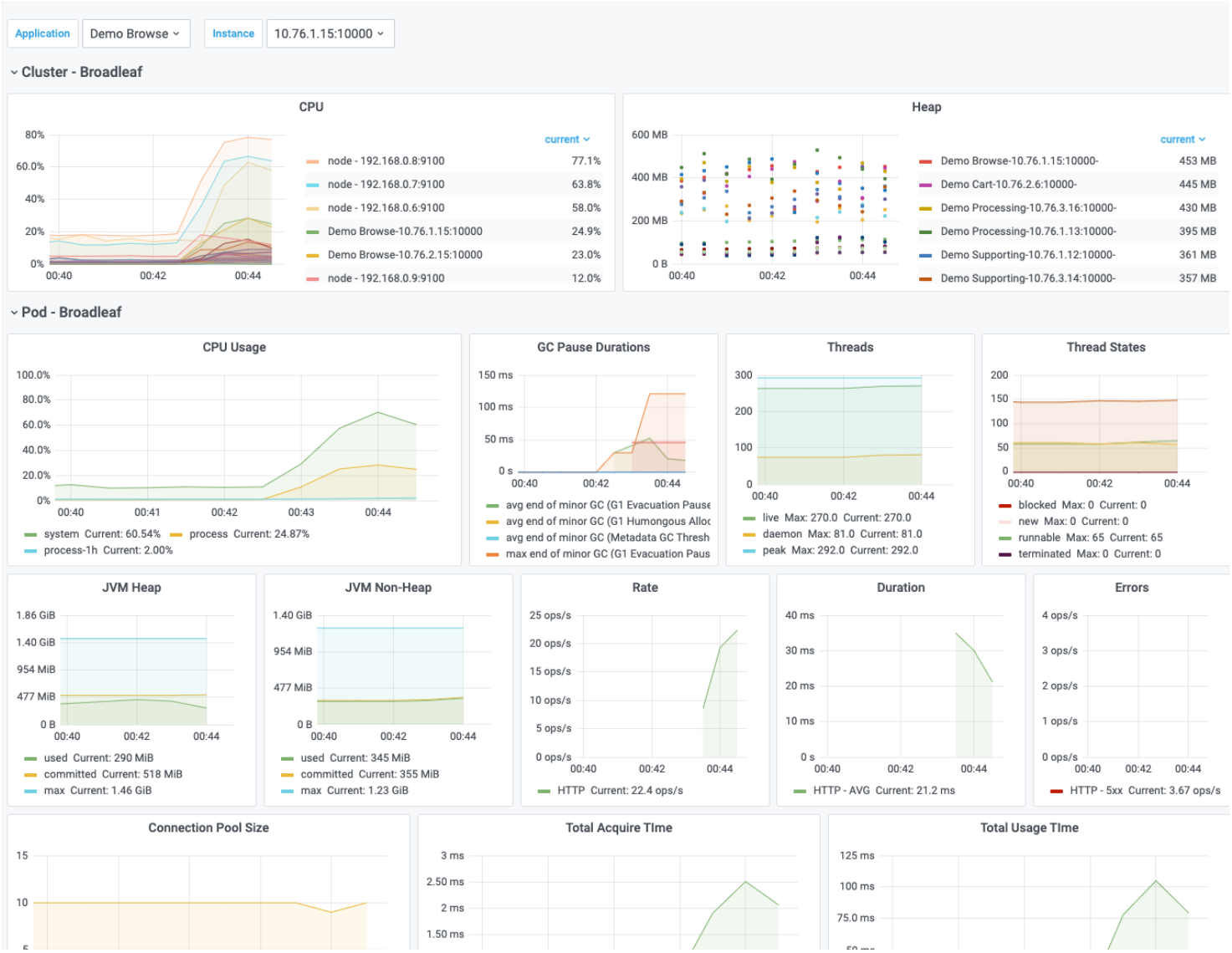
Appendix C - Approach

Figure 1 - JMeter Dashboard



Appendix C - Approach

Figure 2 - System Health Dashboard



Appendix D - Components

Core Components

- Microservices - Provides the core application functionality
- Auth - Provides OAuth related security functionality
- Gateway - Proxy exposed to the client and routes traffic to a microservice
- Database - Data persistence layer for the application
- Kafka - Messaging broker for the application
- Solr - Search services for the application
- Zookeeper - Distributed synchronization for Solr and Kafka
- EFK Stack - APM and centralized log management
- Prometheus - Time series database for system health metrics
- Grafana - System health telemetry

Testing Components

- JMeter - Distributed load test client
- InfluxDB - Time series database for load test results - See Figure 3 for deployment diagram

Granular Microservices

Asset, Catalog, Campaign, Offer, Pricing, Vendor, Catalog Browse, Menu, Personalization, Inventory, Cart, Order, Customer, Cart Operations, Order Operations, Import, Scheduled Job, Admin Navigation, Admin User, Sandbox, Metadata, Tenant, Notification, Search, Indexer

Appendix D - Components

Balanced FlexPackage Contents

Type	Contents
Browse	Asset, Catalog, Campaign, Offer, Pricing, Vendor, Catalog Browse, Menu, Personalization
Cart	Inventory, Cart, Order, Customer, Cart Operations, Order Operations
Processing[4]	Import, Inventory, Scheduled Job, Catalog, Campaign, Offer, Pricing, Customer, Order, Menu, Personalization, Indexer
Supporting	Admin Navigation, Admin User, Sandbox, Metadata, Tenant, Notification, Search

Min FlexPackage Contents

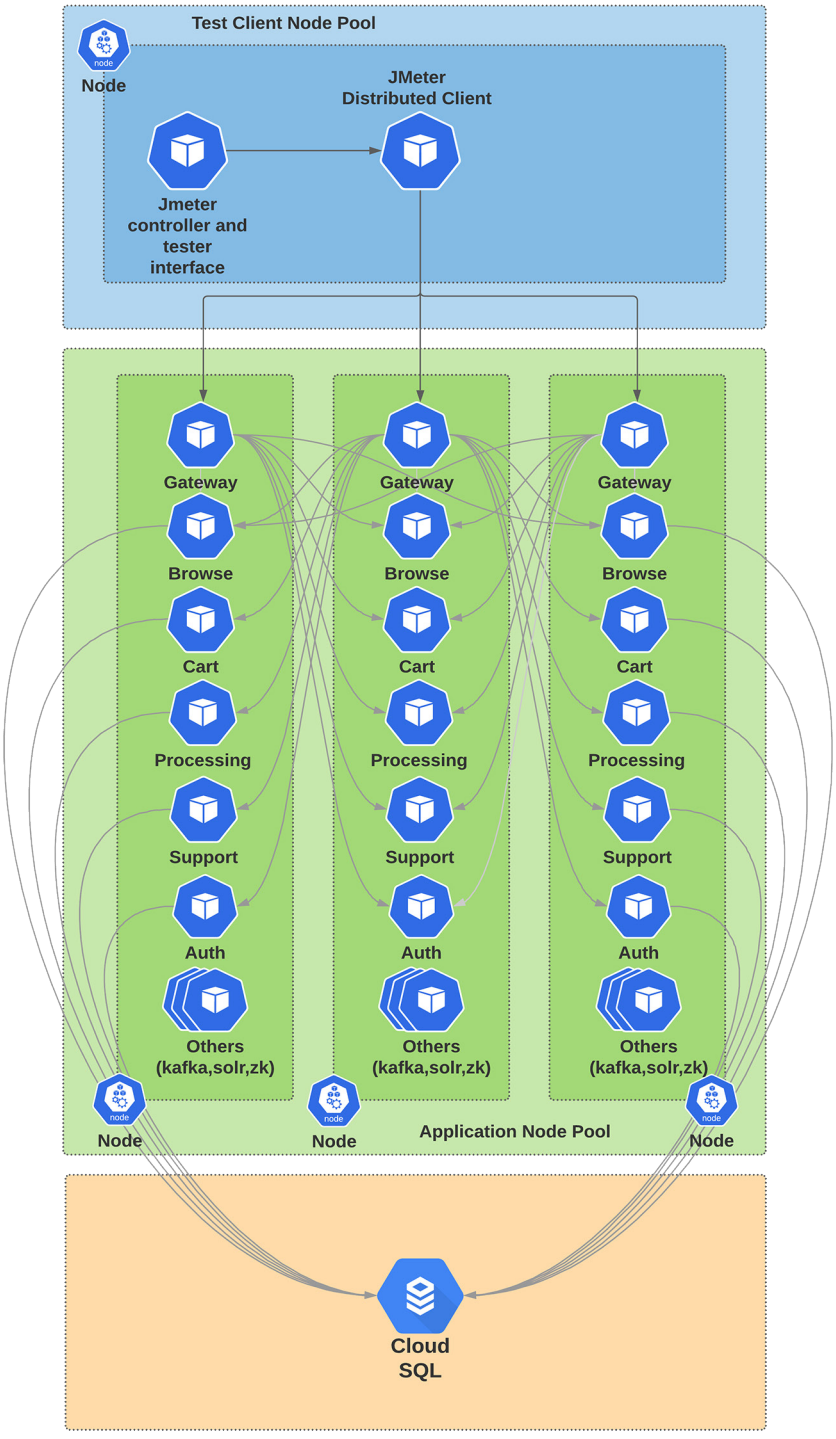
Type	Contents
Min	Asset, Catalog, Campaign, Offer, Pricing, Vendor, Catalog Browse, Menu, Personalization, Inventory, Cart, Order, Customer, Cart Operations, Order Operations, Import, Scheduled Job, Admin Navigation, Admin User, Sandbox, Metadata, Tenant, Notification, Search

[4] There is some redundancy in microservices located in Processing. Processing handles longer lifecycle tasks from the admin and scheduled jobs. That work is isolated here, protecting the utilization of Browse and Cart for customer-facing traffic.

Appendix D - Components

Figure 3 - Sample Deployment Diagram

(Observability components not depicted)



Appendix E - Sample Kubernetes Deployment Plans

BM5 at Browse Heavy 3% Conversion Test
(Other components not shown)

Type	Replicas	Cores Requested (m)	Memory Requested (Mi)
Auth	2	800	1000
Browse	3	2000	2000
Cart	2	2000	2000
Processing	3	700	2000
Supporting	3	700	2000
Gateway	5	1200	500

BL5 at Browse Heavy 3% Conversion Test
(Other components not shown)

Type	Replicas	Cores Requested (m)	Memory Requested (Mi)
Auth	5	1600	1000
Browse	10	2200	2000
Cart	10	1600	2000
Processing	5	1300	2000
Supporting	5	1300	2000
Gateway	10	1000	500

BL6-M8-S4 at Order Focused 100% Conversion Test
(Other components not shown)

Type	Replicas	Cores Requested (m)	Memory Requested (Mi)
Auth	10	1400	1000
Browse	14	2000	2000
Cart	20	3000	2000
Processing	10	1400	2000
Supporting	10	1400	2000
Gateway	14	1000	500

Appendix F - Technical Recommendations

It's worthy to note the kubernetes scheduler on its own does a fair job of organizing pods based on the defined resource requests. However, often you will require more control over how pods are scheduled. Kubernetes provides several advanced configuration mechanisms that should definitely be included in your toolbelt when designing a deployment for your cluster. Specifically, node tainting, toleration, and affinity configurations are invaluable to customizing the deployment to achieve the most efficient and performant installation. We made extensive use of these features during the testing performed in this paper.

If your scale requirements are large enough, it will begin to make sense to segregate pods by type into different node pools. This type of configuration is most clearly evidenced in our BL6-M8-S4 configuration that was used to achieve 100 OPS.

The performance of the system will be most sensitive to adjustments in CPU core request configuration. The system performs well at smaller sizes for smaller throughput requirements. However, the most efficient sizing when using the balanced FlexPackage utilizes cart pod replicas sized at 3000m and browse pod replicas sized at 2000m. Consider these sizes as you explore larger nodes and you have more opportunity to request larger CPU allocations.

When starting a load test, it can be advantageous to use liberal node size and not define pod resource constraints. Then, run load and review kubectl top pod results to see where resources are naturally allocated. Armed with that information as a guideline, setup pod resource requests and limits for future test runs. Add replica count until you reach your throughput goals. Such an approach can take out some of the guesswork when starting to design a load test.

About Broadleaf

Built with the future in mind, Broadleaf Commerce is an enterprise software provider with a proven track record of solving complex commerce challenges. Our API-first approach, and cloud-native microservice architecture gives you the control, flexibility, and performance to innovate quicker and achieve time to value faster.

As the market-leading choice for enterprise organizations requiring tailored, highly scalable commerce systems, we deliver a modular platform that embraces an open philosophy with an extensible and intuitive administrative console.

Broadleaf Commerce was founded in 2009. Over the years, we have earned the trust of leading brands like - O'Reilly Auto Parts, Major League Baseball, ICON Fitness, and Telefonica.

broadleafcommerce.com