Shopware 6.7

Performance Whitepaper

shopware

Performance Whitepaper

Table of contents

shopware

Introduction

In high-demand eCommerce environments, performance is more than just a technical metric - it directly influences user experience, conversion rates, and operational efficiency. With the release of Shopware 6.7, several enhancements have been introduced to improve scalability, reduce resource consumption, and optimize performance under real-world conditions.

This whitepaper presents a focused comparison between Shopware 6.6 and 6.7, offering insights into how each version performs across different load scenarios. The benchmarking covers key aspects, including user interactions (browsing and purchasing), the influence of Varnish caching, and the effect of concurrent API-heavy operations.

The test environment used was consistent, production-grade, and designed to reflect practical usage patterns. By analyzing the outcomes, this paper supports technical leads and system architects in the following:

- Evaluating version-specific performance characteristics
- Understanding infrastructure considerations under load
- Assessing improvements in frontend asset optimization
- Making informed upgrade and scaling decisions

Whether you're planning a migration to 6.7 or assessing the platform's performance profile, this whitepaper provides practical data to support your technical strategy.

Load Testing

Load testing is essential to evaluate how a system behaves under expected traffic volumes, revealing how it scales, and what its maximum operating capacity is.

To objectively assess the performance difference between Shopware 6.6 and 6.7, structured load testing was conducted across multiple scenarios - Browsing-only sessions, Browse and purchase flows, Logged-in fast-buy operations, and API-heavy usage patterns. Through systematic testing, we can:

- Assess how Shopware 6.6 and 6.7 respond to traffic surges
- Measure system behavior under both normal operating conditions and peak loads
- Determine the maximum sustainable throughput of applications
- Identify and analyze performance differences with and without caching layers
- Understand the impact of simultaneous API imports on storefront performance

The following sections describe the results of load testing on Shopware 6.6 and Shopware 6.7 which were performed in May 2025. The goal was to find noticeable improvements in the new version

Setup

Infrastructure

Performance was measured on Hetzner cloud servers with the following hardware specifications:

App Server

- 48 AMD EPYC-Milan Core CPU
- 192 GB memory
- NVME Disk: 900GB

Running a single Shopware application with a RabbitMQ server and two Valkey servers.

Running the key-value server on the same machine reduces the latency. This does not apply to a cluster setup.

Database Server

- 32 AMD EPYC-Milan Core CPU
- 128 GB memory
- 600 GB NVME SSD
- Running MariaDB 11.7

This setup would cost about €600 per month. It is suitable for relatively large-scale infrastructures and is significantly larger than a small or medium-sized business using Shopware would need. Therefore, the results of this whitepaper indicate how Shopware might scale in larger infrastructures and what to expect from it.

Please note that the run tests have 4 different configs in terms of caching and API load. For more information, please refer to our documentation: hosting infrastructure.

Performance Tweaks

Shopware offers performance-oriented configurations that can improve response times in specific environments and load characteristics. For example, some Shops may have time-restricted sales where a high number of not logged-in users are accessing the shop at the same time. In this case, the performance tweaks can help to improve the response times for these users.

These optimizations can significantly enhance performance through:

- PHP configuration adjustments
- Caching strategies
- Infrastructure optimizations
- Database tuning
- Message queue configurations

An overview of all performance-related configurations can be found in our performance tweaks documentation.

The following settings were identified during the load tests and are thus highlighted:

Disable App URL external check

On any Administration load, Shopware tries to request itself to test that the configured APP_URL inside .env is correct.

Test Data

Performance tests assessed a high number of products using data automatically generated and checked for validity and consistency. All performance tests used a base of 0.5 million products.

While the scope of the project did not have a specific high number of prefilled customers or order tables in mind, it did generate a high volume of customers and orders.

Test Setup

System performance must always be understood under certain conditions. To get a realistic overview of the performance of Shopware, the load test must simulate users who visit the store, click through the products and categories, and order items.

Each run was benchmarked against Shopware 6.6 and Shopware 6.7 using identical user loads and infrastructure settings to ensure fairness and comparability.

Test Setup Summary

Component	Details
Test Frameworks	K6 (for load tests), sitespeed.io (for frontend audits)
User Simulation	Virtual Users (Vus) broken down into browsing, guest buying, fast-buying (logged-in), and API importers
Environments	Shopware 6.6 and Shopware 6.7, identical infrastructure for both
Caching Layer	Scenarios toggle Varnish on/off to test frontend cache impact
Background Load	Some scenarios include active API importers simulating stock and price changes
Execution Mode	Benchmark-style (no delay between actions) to stress system throughout
Metrics Collected	RPS, orders/sec, p95/p99 latency for backend; LCP, CLS, JS/CSS payload, and load time for frontend

Scenario Mapping: Coverage Across Load Types

To validate the performance gains introduced in Shopware 6.7, we designed a series of four distinct test scenarios, each simulating a different type of operational load. These scenarios were carefully crafted to mimic real-world ecommerce behavior from regular browsing traffic to enterprise-grade stress under heavy concurrency and API-driven updates.

Load Test Scenarios Overview

Scenario	Caching	API Importers	Max VUs	Purpose
1. Organic Traffic	❌ No	❌ No	662	Measure core backend & storefront performance without caching or imports
2. Flash Sale	✅ Varnish	❌ No	1,324	Simulate high read/buy concurrency, test Varnish impact
3. Enterprise Load	✅ Varnish	✅ 2 Importers	1,324 + 2 APIs	Push backend and frontend under maximum load real-world enterprise setup
4. Frontend Optimization Audit	N/A	N/A	Headless	Validate client-side performance changes (JS/CSS redution, visual metrics)

User Behavior Simulation

Each scenario different user profiles to simulate realistic trafficombinesc mixes:

Type	Scenario	Description	User/System Behavior
Visitor	Browse Only	Simulates a user who only browses the storefront without making a purchase	- Users navigate the homepage, browse categories, search pages, and product detail pages. - No cart activity, no checkout, and no API calls involved.
Guest User	Browse and Buy	Simulates a complete shopping journey of an unregistered user performing a guest checkout.	- Guest users perform a full shopping experience. - Registers as a guest during checkout. - Adds ~10 products to the cart, proceeds to checkout, places an order, and exits.
Registered User	LoggedIn FastBuy	Simulates a registered customer who logs in and quickly places an order with minimal browsing.	- Authenticated users login to their account. - Directly visit a product detail page, add the product to the cart, go to checkout, place an order, and exit.
Backend Service	API Import	Simulated backend system actions to update the catalog via API. Typically used for importing and updating products in bulk.	- Uses a valid bearer token to authenticate. - Performs batch product imports, price updates, and stock adjustments via API calls. - No frontend user interaction.

Scenario 1: Organic Traffic – No Caching, No API Load

This scenario simulates natural customer behavior under no caching or external API load. It reflects a moderate but steady influx of users browsing products and occasionally making purchases typical of an early-stage or mid-sized ecommerce site without performance-boosting infrastructure like Varnish or importers.

User Behavior

- 600 VUs browsing product listings and detail pages
- 50 VUs simulating guest checkouts (browsing, carting, ordering)
- 12 VUs logging in and performing fast buys (minimal clicks to checkout)

Infrastructure

- No Varnish (no frontend caching)
- No API importers (no backend background load)
- Max concurrent users: 662

Organic Traffic Performance Comparison

Metric	Shopware 6.6	Shopeare 6.7	Change
RPS	~940	~940	No change
Orders/sec	0.5	0.6	+20%
p95 latency	162 ms	231 ms	↑
p99 latency	666 ms	1,000 ms	↑

Conclusion

Shopware 6.7 processes more orders under identical uncached traffic, despite slightly higher latency. This points to improved transactional backend handling, even in the absence of performance-enhancing layers.

Scenario 2: Flash Sale – High Traffic with Varnish Caching

This scenario represents a flash sale or advertisement campaign, where hundreds of customers hit the storefront simultaneously, many of them completing purchases. The load is heavy and immediate, pushing the system's caching efficiency to its limits.

User Behavior

- 1,200 VUs browsing actively
- 100 VUs guest buyers, registering during checkout
- 24 VUs logged-in returning customers performing fast buys
- All users operate in benchmark mode (no wait time between actions)
- Conversion rate: 100% for buyer flows

Infrastructure

- Varnish enabled (frontend content cached)
- No API imports (pure frontend/backend load)
- Max concurrent users: 1,324

Comparison Table

Metric	Shopware 6.6	Shopware 6.7	Change
RPS	~1,720	~1,870	+8.7%
Orders/sec	1.9	3.96	+108%
p95 latency	408 ms	296 ms	↓
p99 latency	1,000 ms	425 ms	↓

Conclusion

Varnish makes a huge difference — and Shopware 6.7 amplifies this. Order throughput more than doubles while latency drops by 25–60%, showing excellent frontend cache coordination and backend responsiveness under extreme buyer activity.

Scenario 3: Enterprise Load – Varnish with Continuous API Imports

This is a true enterprise-grade simulation with frontend user traffic and intense backend activity. API importers push constant changes (product stock, price updates), while customers browse and make purchases. This scenario mimics a live, high-volume store with dynamic catalog updates and real-time user activity.

User Behavior

- 1,200 VUs browsing product pages
- 100 VUs buying as guests
- 24 VUs logged-in fast buyers
- 2 concurrent API Importers simulating catalog changes
- Users operate without delay between actions

Infrastructure

- Varnish enabled (frontend caching)
- API importers active (background load)
- Max concurrent users: 1,324 + API

Comparison Table

Metric	Shopware 6.6	Shopware 6.7	Change
RPS	~1,280	~1,780	+39%
Orders/sec	2.29	3.79	+65%
p95 latency	2,000 ms	321 ms	↓ ~85%
p99 latency	3,000 ms	458 ms	↓ ~85%

Conclusion

This is the most demanding scenario — and where Shopware 6.7 truly shines. Latency collapses from multi-second ranges to sub-500ms. The combination of Varnish caching, Valkey optimization, Store-API cache removal, and async improvements results in a backend that scales effortlessly under real-world pressure.

Scenario 4: Frontend Optimization – Sitespeed Audit

This scenario reflects a client-side performance audit using sitespeed.io, focusing on load times, JS/CSS payload, and visual stability. The test is run on default Storefront pages using headless browser metrics (e.g., Largest Contentful Paint, Layout Shift).

User Behavior

- Simulated page visits (no interactions)
- Device profile: Desktop
- Audit scope: Homepage, product page, category page

Comparison Table

Metric	Shopware 6.6	Shopware 6.7	Change
Performance Score	76	78	+2 points
JS Size (uncompressed)	127 KB	95 KB	↓ ~25%
CSS Size (uncompressed)	344 KB	262 KB	↓ ~24%
Page Load Time	122 ms	137 ms	Slight ↑
Largest Contentful Paint	144 ms	144 ms	=
Cumulative Layout Shift	0.099	0.15	Slight ↑

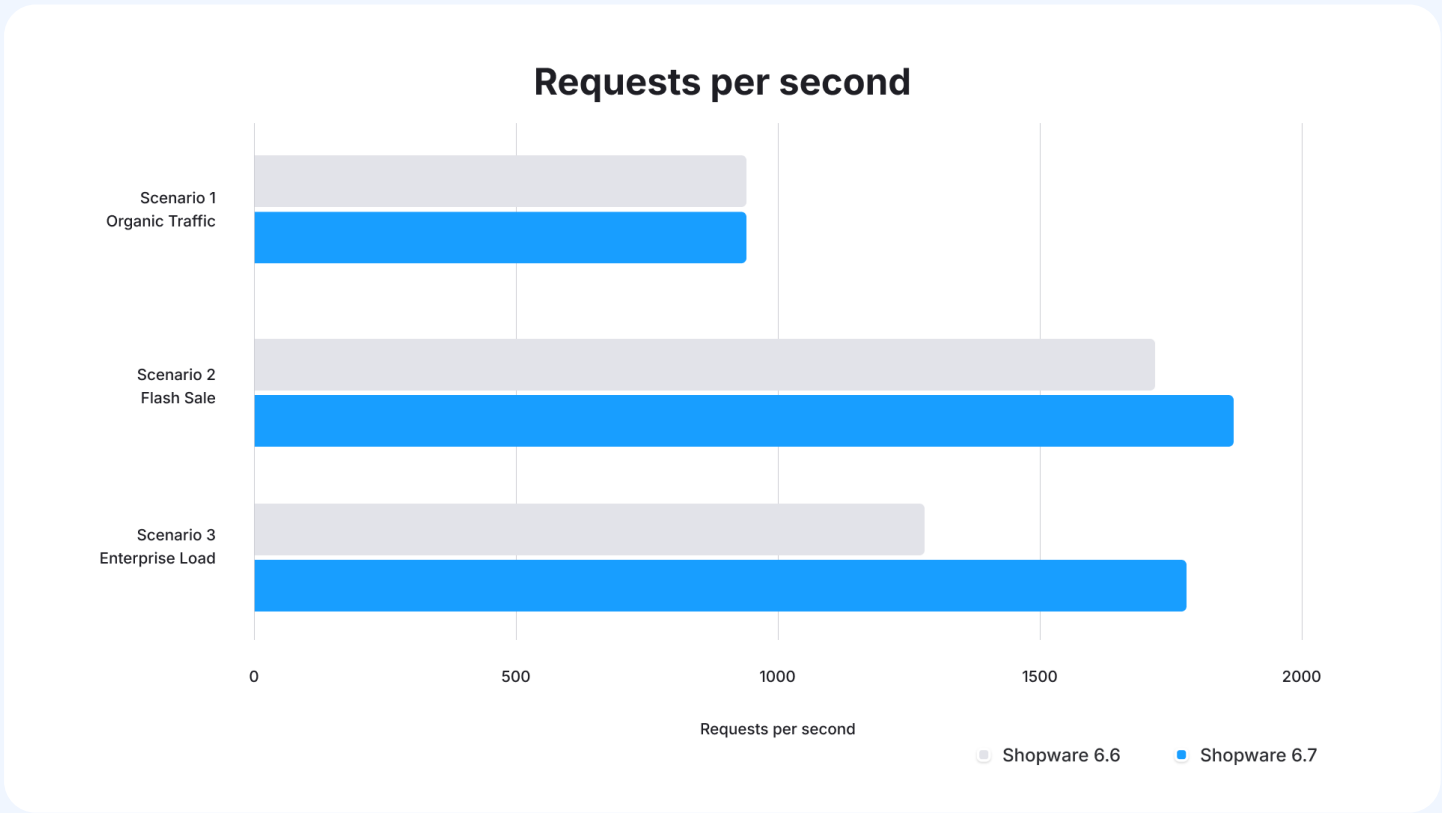
Conclusion

Shopware 6.7 ships with a leaner and more modern frontend, driven by the Meteor design system and Vite. The JS and CSS payloads are smaller by \~25%, helping improve perceived load time. Despite a small uptick in CLS, visual performance remains stable, confirming a more efficient rendering pipeline.

What Each Scenario Proves

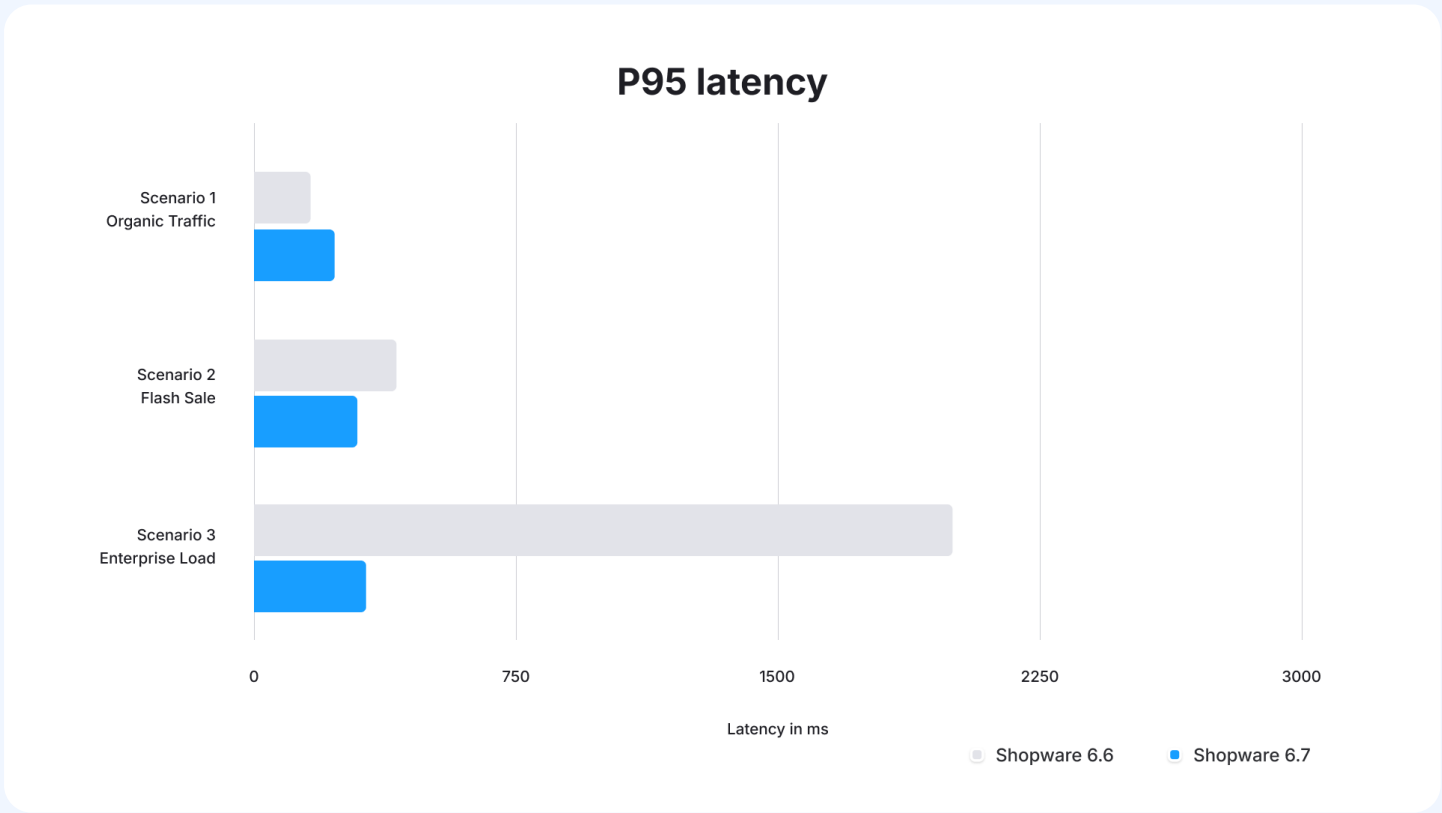
Scenario	Primary Goal	What It Tests
Scenario 1	Baseline reference with minimal infrastructure	Backend efficiency under uncached, moderate user traffic
Scenario 2	Performance under frontend cache during campaign-like spikes	Caching efficiency, order throughput under pressure
Scenario 3	Full-stack resilience with concurrent frontend load and backend write activity	Stress resilience, write/read concurrency, Varnish + API handling
Scenario 4	Evaluate perceived performance on default Storefront	Frontend weight optimization (JS/CSS size), rendering stability (CLS), and visual performance

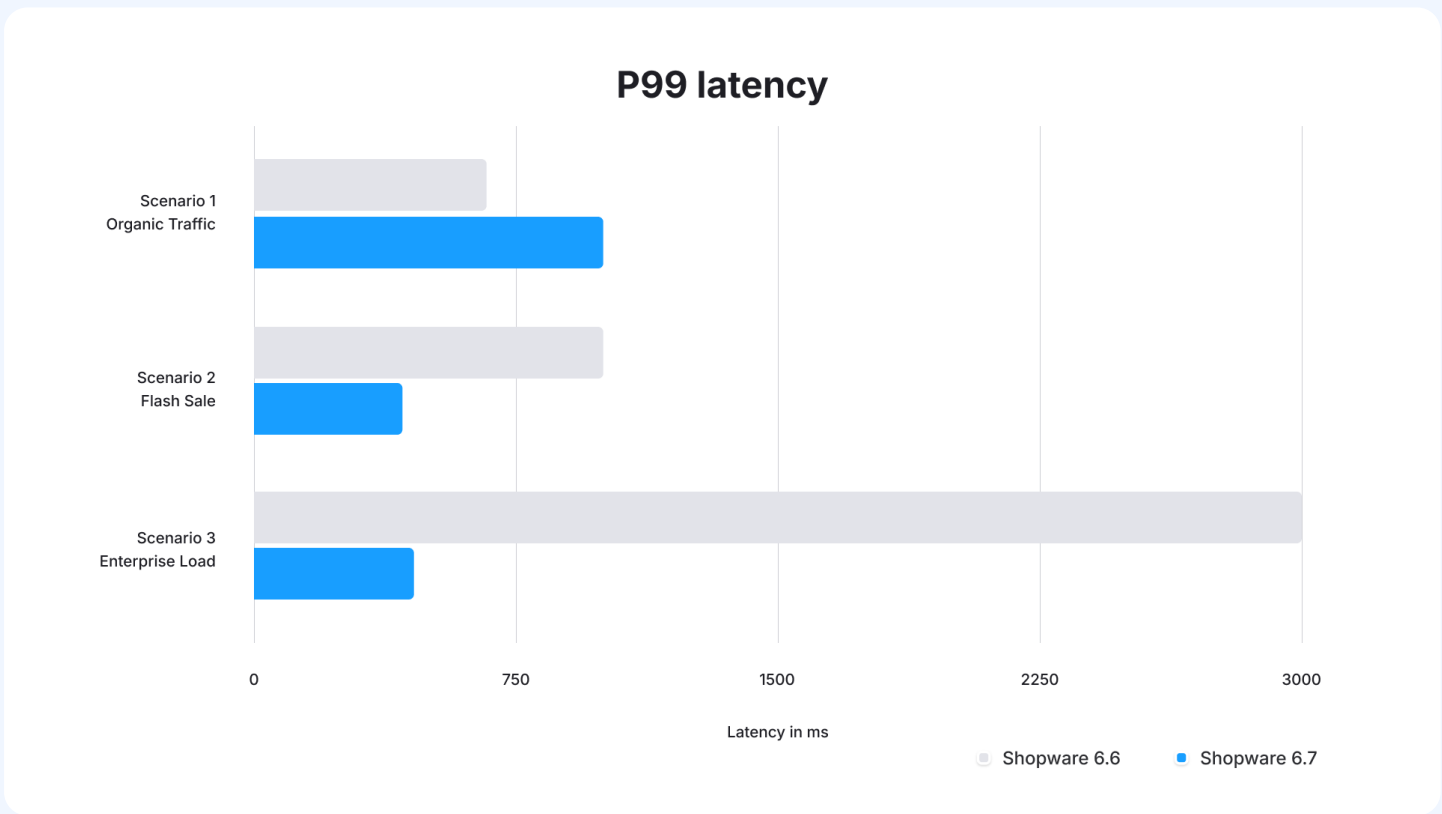
Shopware in Numbers



shopware

Shopware in Numbers





shopware

Shopware in Numbers

Products/SKUs

Generally, there is no hard limitation on products or SKUs in Shopware. However, depending on the server setup, we usually recommend Elasticsearch for catalogs with 120,000–240,000 and more SKUs. With Elasticsearch, we successfully assessed Shopware with 2 million products in our load test (see above).Even more products/SKUs have been tested on development
machines.

Categories

Although Shopware was load-tested with only 500 categories, we know from other tests that 5,000 categories and more are possible.

Concurrent users

Our load test assessed 1,300 concurrent Shopware users. As read database, app servers and (if applicable) HTTP caches can be scaled horizontally, the value of concurrent users should be scalable correspondingly.

Page impressions

The number of page impressions links to the number of users (see above). In our load tests, we generated up to 72,000 page impressions (requests) per minute without HTTP cache.

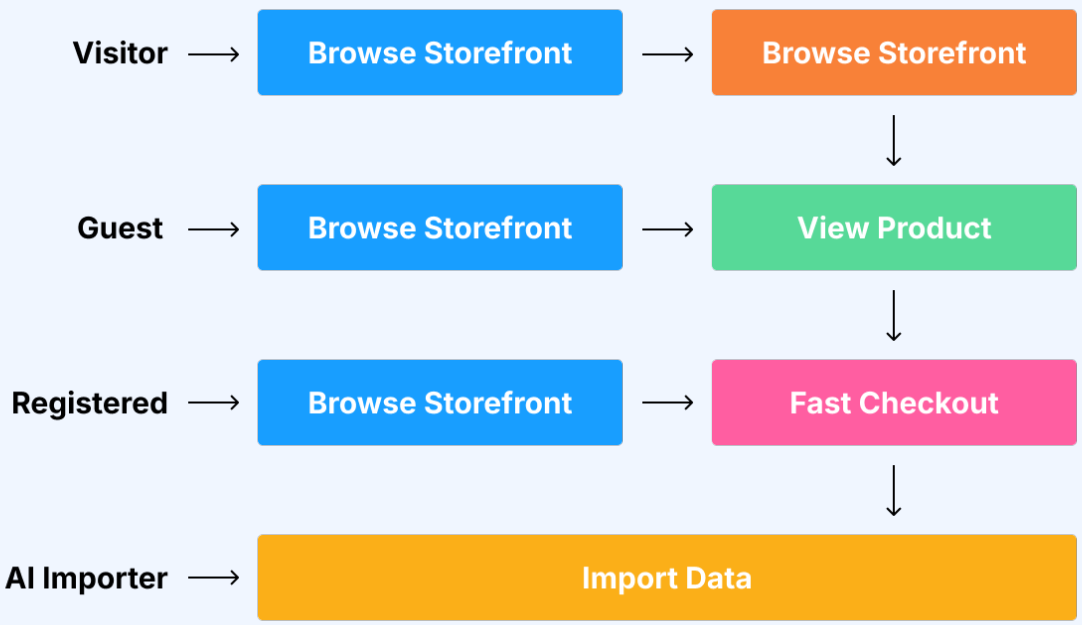
Shopware in Numbers

Registered users

Shopware was tested with up to 560,000 registered customers. The registration of Shopware was tested with 32 registrations per second. Again, depending on the setup, this value should be easy to exceed.

Orders

The number of existing orders barely affects the frontend performance. Shopware 6 was tested with up to 2 million orders. Note that the number of orders might affect the performance of the order admin and promotion modules. During the load tests, up to 500 concurrent users placed orders. In the Advertisement scenario, 240 orders were placed within a minute.



shopware

Key Takeaways

- Shopware 6.7 shows clear backend gains under high load, with latency and throughput improving dramatically.
- Frontend is lighter and faster, thanks to Vite migration and code cleanup.
- Caching strategies (Varnish) multiply performance benefits.
- Shopware 6.7 is well-optimized for scale on modest hardware (\~€500/month setup)
- Backend performance: Up to 40% faster under high API load.
- Frontend performance: \~25% reduction in JS payload size.
- Infrastructure changes Redis replaced by Valkey, Store-API cache removed, Vite integration.
- Frontend metrics: Sitespeed performance score increased from 76 to 78.
- Comparison of Requests per Second (RPS) 6.7 consistently handles higher throughput, especially under API load.
- Order processing speed (Orders/sec) nearly doubles in critical scenarios with Shopware 6.7.
- Significant improvements in p95 Latency under heavy API load reduced from 2s to just over 300ms in 6.7.

Appendix

These configurations were used for performance benchmarks.

PHP

ini
memory_limit=512M
post_max_size=32M
upload_max_filesize=32M
session.save_handler = redis
assert.active=0
date.timezone=Europe/Berlin
opcache.enable_file_override=1
opcache.interned_strings_buffer=20
opcache.preload=/var/www/html/var/cache/opcache-preload.php
opcache.preload_user=nginx
zend.detect_unicode=0
realpath_cache_ttl=3600
redis.clusters.cache_slots = 1

FPM

ini
pm.max_children = 1500
pm.start_servers = 40
pm.min_spare_servers = 300
pm.max_spare_servers = 1500
rlimit_files = 64000

Valkey

conf
appendonly no
save ""
maxmemory 7G
maxmemory-policy volatile-lfu

MariaDB

ini
sql_mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION
log_bin_trust_function_creators=1
binlog_cache_size=64M
key_buffer_size=0
join_buffer_size=1024M
innodb_log_file_size=20G
innodb_buffer_pool_size=100G
innodb_buffer_pool_instances=4
innodb_flush_log_at_trx_commit=2
innodb_flush_method=O_DIRECT
innodb_io_capacity=2000
innodb_io_capacity_max=4000
innodb_read_io_threads=8
innodb_write_io_threads=8
innodb_thread_concurrency=32
group_concat_max_len=320000
default-time-zone=+00:00
max_binlog_size=512M
binlog_expire_logs_seconds=600

max_connections=10000
table_open_cache=4096
table_definition_cache=4096
tmp_table_size=2G
max_heap_table_size=2G