magecom

Case Study

Creating a Highly Loaded Infrastructure Based on Amazon Services to Carry the Capacity During Sales

For an Online Fashion Store

Content

About client

An **online fashion store** that sells women's clothing and accessories within the EU. The product assortment of the store is presented in **17** categories and **1250+** items.

The company actively uses social networks to attract customers. The client regularly organizes promotions and sales.

Region: **France** Industry: **Fashion**

Client in figures

400K+
Followers

120K+
Monthly Visits

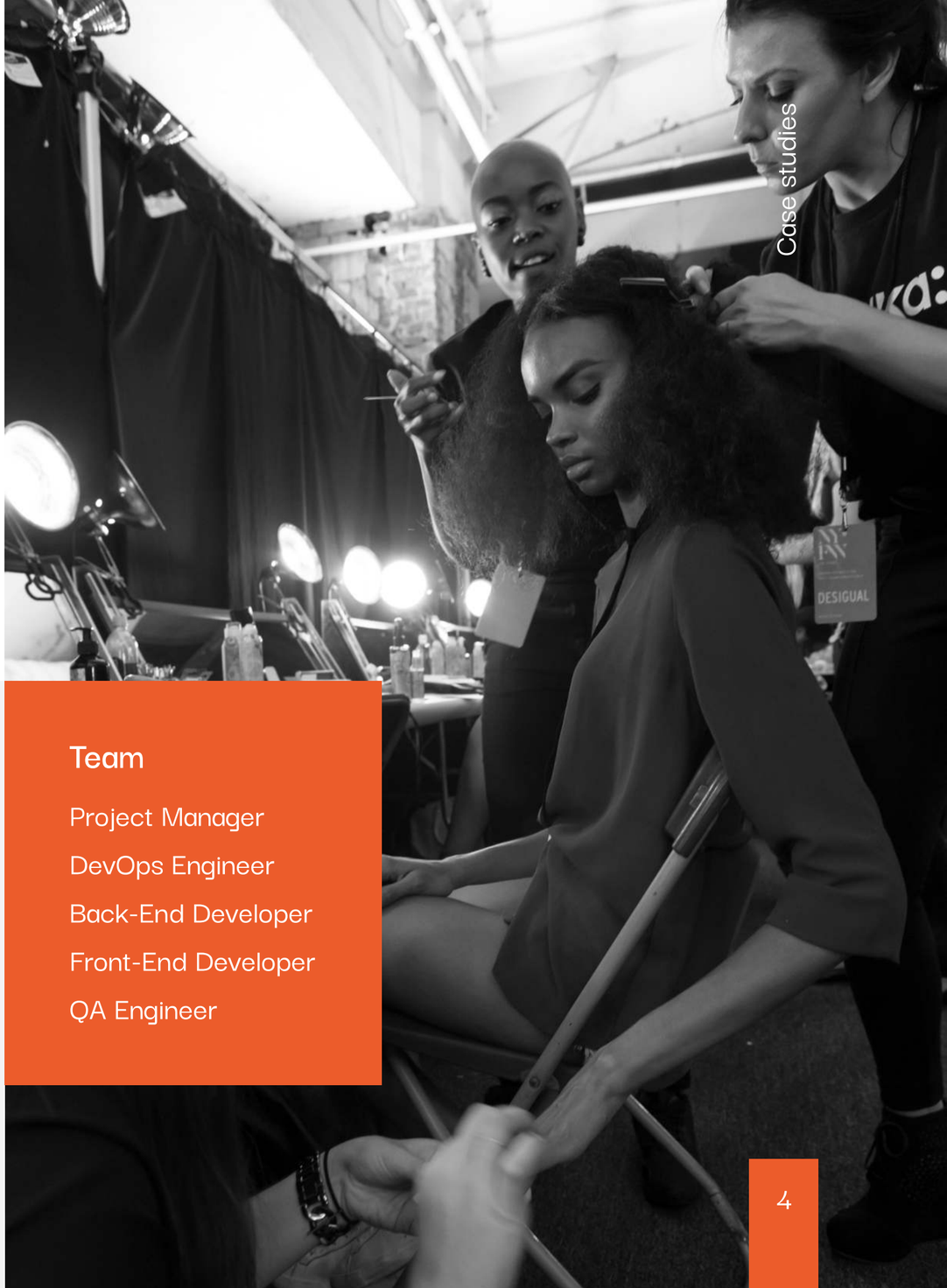
10K+
Simultaneous Users

70%
Of Mobile Traffic

Challenges

The client is in the middle of migrating the project from Prestashop to Magento 2.3 Commerce Edition. At some point, they start to pick a hosting environment, where they consider several options, such as Magento Cloud and a self-hosted solution. Eventually, after quite a lot of back and forth, they land on **Amazon Web Services (AWS)** and ask us for help with the DevOps part.

Also, the client expects a high traffic load during the period of sales and promotions. Based on forecasts for advertising campaigns, they expect that the number of users who visit the site at the same time will reach up to 10,000. We are given the task of creating a server infrastructure, which can handle the load during sales and discounts.



Team

Project Manager

DevOps Engineer

Back-End Developer

Front-End Developer

QA Engineer

Tools

Magento 2 COMMERCE EDITION

amazon web services

fastly

New Relic

APACHE JMeter™

Amazon Web Service

- Amazon Code Deploy
- EC2 Instances
- Amazon autoscaling
- AWS Aurora database, RDS instances
- Redis
- Full-page cache, Object Magento cache
- Elastic Load Balancer
- Elasticsearch
- Bastion
- NFS

Amazon Web Services (AWS) is a subsidiary of Amazon that provides on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. In aggregate, these cloud computing web services offer a set of primitive abstract technical infrastructure and distributed computing building blocks and tools. The AWS technology is implemented at server farms throughout the world, and maintained by the Amazon subsidiary. Fees are based on a combination of usage, the hardware / OS / software / networking features chosen by the subscriber, required availability, redundancy, security, and service options.

SOLUTIONS

01 Planning stage

We always begin working on a project by analyzing the environment and planning our next steps. We also meet with our key specialists to outline future architecture.

During the analysis of this project, it becomes clear that peak loads occasionally occur during the year, which means that we face the task of creating a scalable environment that can adapt to the load both high and low.

One of the critical points of this project is the high traffic that needs, if possible, to be unloaded at the entry point of the infrastructure. It also requires high-performance content delivery to provide a pleasant user experience. To solve this challenge, we choose **Fastly**[1], because it's a configurable content delivery network that can significantly accelerate the delivery of content both cacheable and uncacheable – like product prices and other dynamic and event-driven content.

Apart from that, high-traffic businesses often face the problem of malicious attacks, especially during promotions and sales. Fastly also filters, cuts off the low-quality traffic, and provides powerful protection against DDoS attacks, which is essential for such projects.

[1] **Fastly CDN** makes transmission of the content to the end-users more efficient by automatically storing copies at intermediate locations temporarily. The process of storing these copies is known as "caching." Meanwhile, server locations, in which they are stored, referred to as "caches." Fastly customers specify the origin domains, servers, and applications from where the original content will be fetched. When a request is made, the closest Fastly cache to the user fetches the content from the origin server, stores it in the cache, and sends the response to the end-user.

02 Elastic Load Balancer

Besides, the load also needs to be effectively shared and managed within the architecture; that's why we consider the possibility of using several servers on which Magento can be hosted on. As the infrastructure becomes multi-server then, we decide on a **Load Balancer**[2], which can easily cope with all of the tasks mentioned above.

03 Deployment

Further, we think about the deployment phase, which should be taken into account. The deployment process of the project consists of the Git client's repository and the AWS code deploy with Bitbucket Pipelines, which allows automatically building, testing, and deploying the code, based on a configuration file in the repository. As a consequence, commands can be run with all the advantages of a fresh system inside the containers in the cloud.

[2] **Elastic Load Balancer** automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, IP addresses, and Lambda functions. It can handle the varying load of your application traffic in a single Availability Zone or across multiple Availability Zones.

04 Autoscaling group

Keeping in mind there are several Magento servers managed by an Elastic Load Balancer and the condition that peak loads occur a few times a year, there should be a convenient way to add and remove servers when it's necessary. In fact, Amazon has a suitable technology address that – an autoscaling group, so the more servers are involved in the group, the higher load they can carry.

05 Databases

Also, when working on such a high-loaded project, special attention should be paid to working with databases. They differ from application servers because they can't be included in the autoscaling group that easily. We work with the **Amazon Aurora database**[3], which suits the conditions of the project since it's fully managed by Amazon Relational Database Service (RDS). It automates time-consuming administration tasks like hardware provisioning, database setup, patching, and backups.

For the project, we select two powerful ones – Master and Slave with a synchronization mechanism between each other. The Master database server is where the recording process takes place, and the Slave database server is where the reading is coming from. As the recording is a more time-consuming operation than reading, we plan to separate them to achieve the necessary performance.

[3] **Amazon Aurora** is a MySQL and PostgreSQL-compatible relational database built for the cloud, that combines the performance and availability of traditional enterprise databases with the simplicity and cost-effectiveness of open source databases. It's up to five times faster than standard MySQL databases and three times faster than standard PostgreSQL databases. It provides the security, availability, and reliability of commercial databases.

06 Caching

Another important artifact that is typical for any Magento project is caching. First, to improve the response time and reduce the load on the server, we implement a **full-page cache (FPC)**[4] and object **Magento cache (OMC)**[5]. They're used to display categories, products, CMS pages quickly. Without them, each page may need to run blocks of code and retrieve information from the database, which is usually quite resource-consuming. However, with the caching enabled, a fully-generated page can be read directly from the fast cache memory.

The next step in implementing caching is user sessions. The thing is that logged-in users significantly load the infrastructure due to the Magento specifics of working with open user sessions. The good practice of dealing with this task is to store and work with them separately from the Magento servers. We select **Redis**[6] caching service, which allows inserting and retrieving a massive amount of data into its cache within a short period of time. It can be done easily using mass insertion, a feature supported by Redis.

[4] **Full-page cache (FPC)** is used to display category, product, and CMS pages quickly. Full-page caching improves response time and reduces the load on the server. Without caching, each page might need to run blocks of code and retrieve information from the database. However, with full-page caching enabled, a fully-generated page can be read directly from the cache.

[5] **Object Magento cache (OMC)** is a type of server-side caching. This means the caching is administered at the server level, and isn't controlled by the end user or a system they use for caching. Object caching stores database query results that have been loaded. Then, it serves them up faster the next time they're requested so the database doesn't have to be queried again.

[6] **Redis** is a fast, open-source, in-memory key-value data store for use as a database, cache, message broker, and queue. All Redis data resides in-memory, in contrast to databases that store data on disk or SSDs. By eliminating the need to access disks, in-memory data stores such as Redis avoid time delays and can access data in microseconds. Redis features versatile data structures, high availability, geospatial, Lua scripting, transactions, on-disk persistence, and cluster support, making it simpler to build real-time internet-scale apps.

07 Elasticsearch

An important and integral part of any ecommerce project is providing a fast, personalized search experience, allowing users to find relevant data quickly. The thing is that the default Magento search sometimes requires improvements and doesn't always provide as relevant search results, and it's quite resource-consuming, which pushes us to use third-party systems. To upgrade it, we use **Elasticsearch**[7], because of its proven performance and direct access to the APIs.

08 Bastion

This kind of infrastructure currently needs full security and minimizing the chances of the entrance from the outside. For this reason, we pick **Bastion**[8] with the purpose of facilitating access to a private system from an external network. Besides, it has a multi-factor authentication and provides an extra security layer to prevent unauthorized administrative access to systems.

[7] **Amazon Elasticsearch Service (Amazon ES)** is a managed service that makes it easy to deploy, operate, and scale Elasticsearch clusters in the AWS Cloud. Elasticsearch is a popular open-source search and analytics engine for use cases such as log analytics, real-time application monitoring, and clickstream analysis.
It supports natural language search, auto-completion, faceted search, and location-aware search.

[8] **Bastion** is a special-purpose computer on a network specifically designed and configured to withstand attacks. It generally hosts a single application, for example, a proxy server, and all other services are removed or limited to reduce the threat to the network.

09 Load testing

To understand how the infrastructure is going to perform good enough and be in line with what is required, we usually perform load testing. Our QA engineer performs the procedure, verifies the ultimate capacity, and gives feedback with the metrics.

A suitable tool for performing the load testing is JMeter9, which examines the server layer and discovers the maximum load a website can handle by simulating a group of users sending requests to a target server.

We usually combine the output of JMeter with New Relic10 to make the testing performance perfect. It analyzes how the code performs, indicates the problems on the Magento side, and offers in-depth and robust reporting.

10 Visualization of the planned architecture

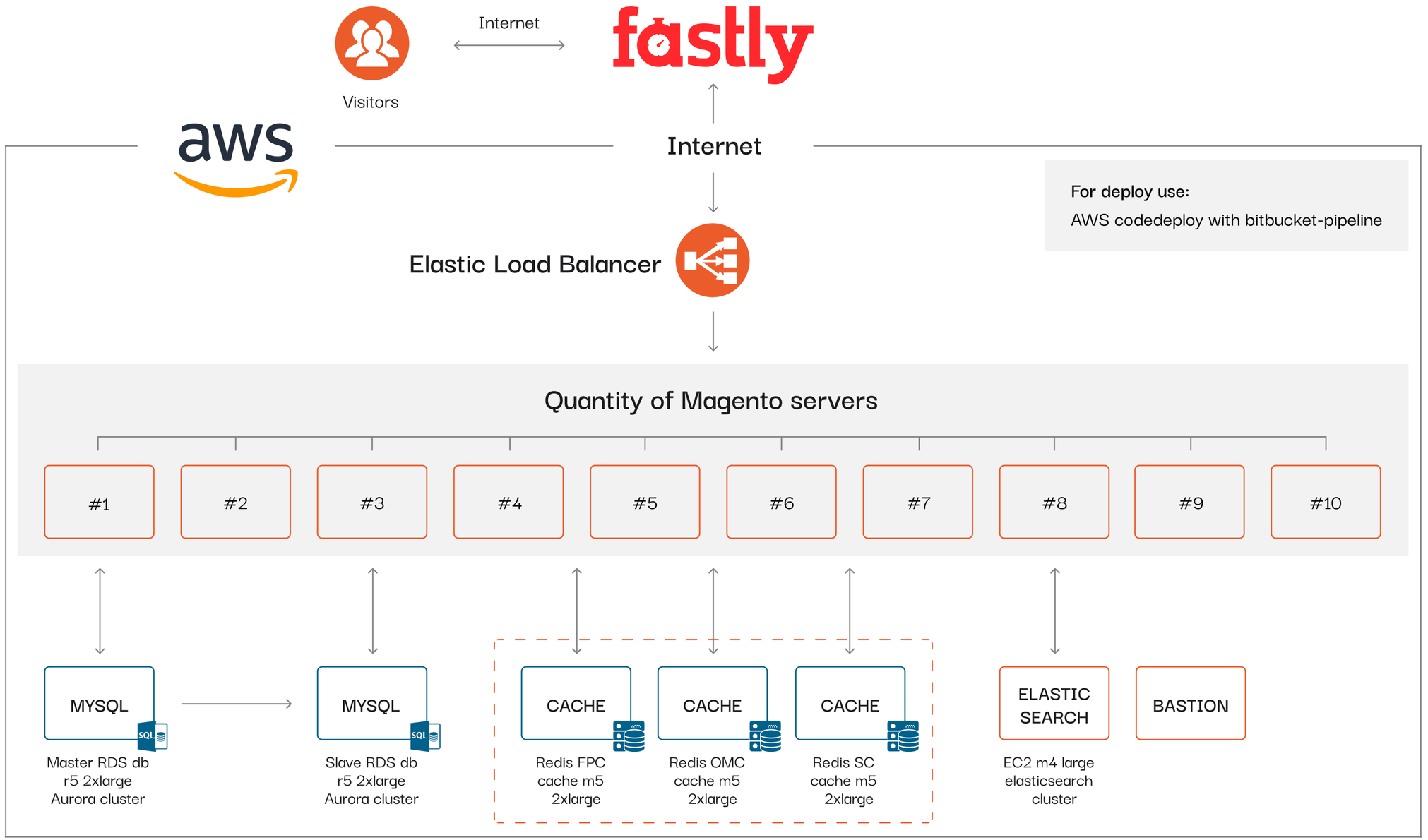
To sum up, we visualize the architecture in the scheme below. We also make calculations of all of the specified tools for the client. After getting approval of the client, we set up the project execution stage.

11 Execution Stage

We start with Fastly CDN, Elastic Load Balancer, the deployment process with Bitbucket Pipelines, Elasticsearch, and Bastion.

[9] **JMeter** can be used as a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications. JMeter can be used as a unit-test tool for JDBC database connections, FTP, LDAP, Webservices, JMS, HTTP, generic TCP connections, and OS native processes.

[10] **New Relic** is a performance management tool that helps analyze and manage application performance, troubleshoot errors, and bottlenecks before they affect your customers' experience.


Visitors
Internet
fastly
aws
Internet
Elastic Load Balancer
For deploy use:
AWS codedeploy with bitbucket-pipeline
Quantity of Magento servers
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10
MYSQL
Master RDS db
r5 2xlarge
Aurora cluster
MYSQL
Slave RDS db
r5 2xlarge
Aurora cluster
CACHE
Redis FPC
cache m5
2xlarge
CACHE
Redis OMC
cache m5
2xlarge
CACHE
Redis SC
cache m5
2xlarge
ELASTIC SEARCH
EC2 m4 large
elasticsearch
cluster
BASTION

Optimization of

12 **the autoscaling group**

The next step of the execution phase is the implementation of the autoscaling group.
We set the following rule: from the moment of promotions and discounts when the average load on the servers reaches 70%, the system provides a new server to balance the overall performance. When the demand goes under 70%, the number of servers reduces one by one, respectively.

At this point, we make a decision to start with 1-2 servers in the autoscaling group on which Magento will be hosted. We do it this way to test the minimum necessary number of servers and optimize the client's costs.

13 **databases**

Next, we move on to the database servers. After conducting the tests, it becomes clear that one database server instead of two, is enough, so the client doesn't have to buy a Slave server and can save money.

14 **storage space**

Another highlight is that we find a way to optimize our infrastructure to save some storage space and increase the speed of content loading in the infrastructure. We implement the **Network File System (NFS)**[11] media library for storing content that doesn't need to be duplicated on each Magento server.

Optimization of

15

caching

Further, we implement Redis, which works well. However, when we start moving on to the implementation of FPC and OMC, which should be taken to separate servers to speed things up, we face a problem. The thing is that based on the plan we choose for the client, the Amazon internal network bandwidth doesn't allow transferring the info from the cache servers to the Magento servers quickly enough. Going to a more expensive plan with Amazon doesn't make financial sense, so we should find a workaround here. After a quick brainstorm with the team, we land on leaving one cache server for the sessions (Redis) and moving FPC and OMC to each Magento server in the autoscaling group to achieve the necessary performance eventually. As a side effect, we save the client's budget by reducing the number of Amazon services without loss of efficiency.

[11] **NFS (Network File System)** is a client/server application that lets a computer user view and optionally store and update files on a remote computer as though they were on the user's own computer. The NFS protocol is one of several distributed file system standards for network-attached storage (NAS).

16 Performing load testing

The large and significant part of the project execution stage is load testing, which is held throughout the entire implementation. It's an iterative process of putting demand on a system and measuring its response. The process of load testing consists of two parts – the server and application layers. The testing algorithm contains the following steps – we perform the load testing, then we get the metrics, which report the capacity that it can carry. After that, we give instructions to the server and application parts and fine-tune the infrastructure.

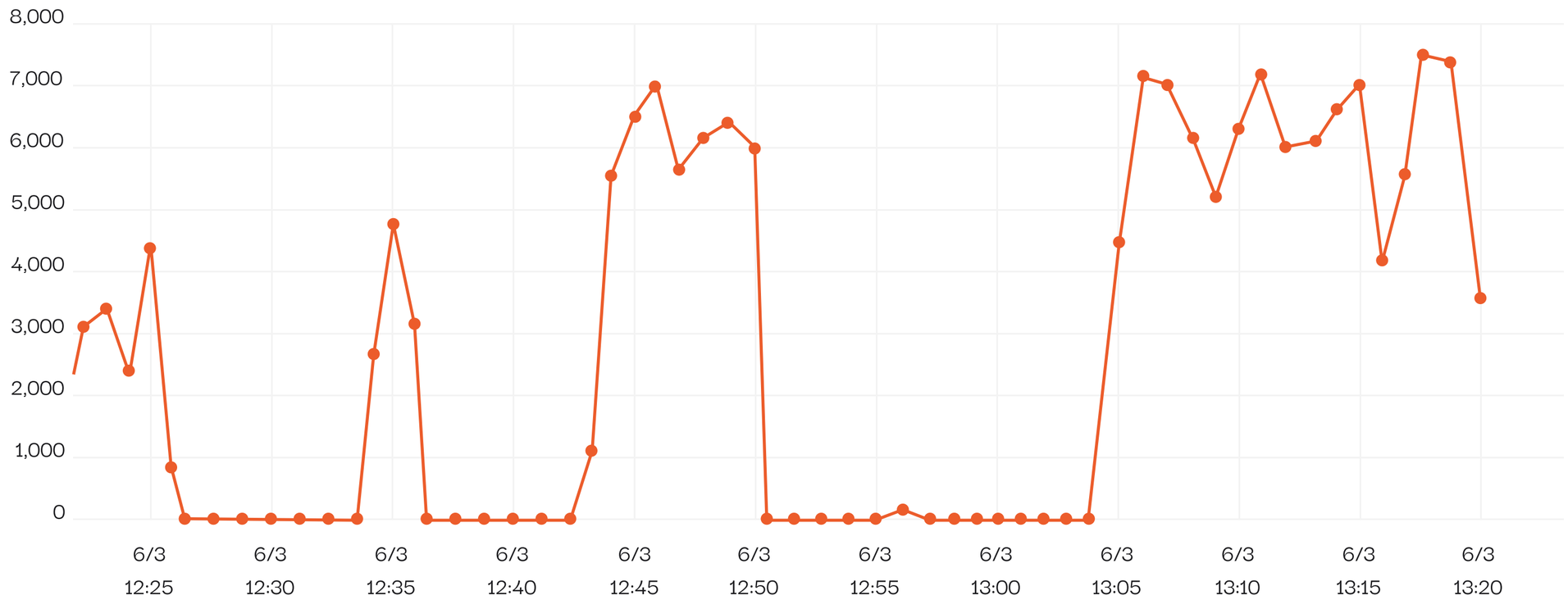
It always makes sense to get intermediate results during the implementation of big tasks. While conducting the tests, we get intermediate results. An example of the dynamics by the number of requests and periods can be found in the New Relic screenshot below.

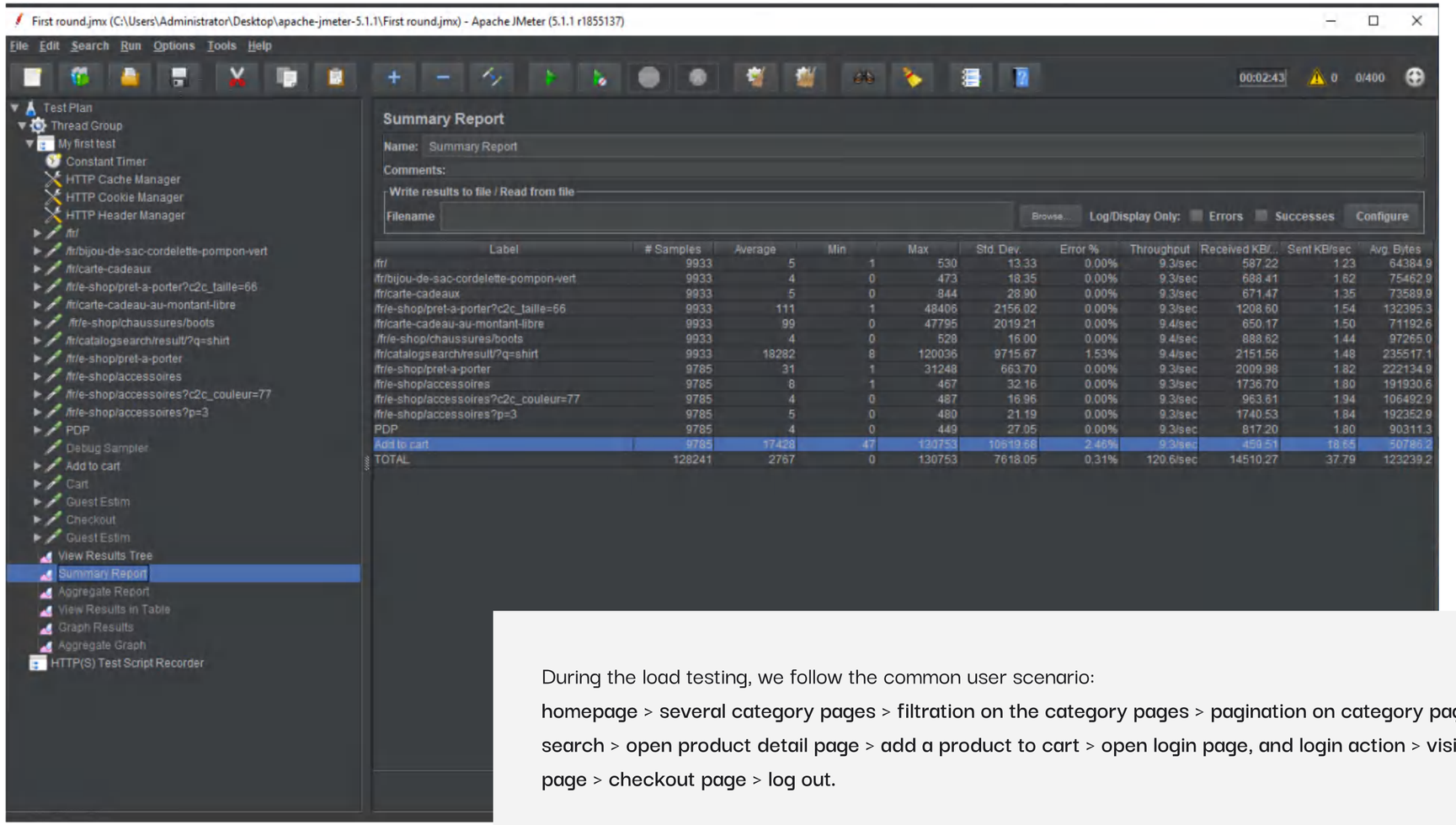
We perform the test on 10 Magento servers in the autoscaling group and reach the point of 7,635 users. It allows us to understand the maximum capacity that the system can carry at a particular moment.

CloudWatch Monitoring Details

Requests (Count)

Time Range: **Last Hour** Statistic: **Sum** Period: **1 Minute**





During the load testing, we follow the common user scenario:
homepage > several category pages > filtration on the category pages > pagination on category pages > search > open product detail page > add a product to cart > open login page, and login action > visit cart page > checkout page > log out.

Then we analyze the summary reports from JMeter, which show all the data for each stage of the scenario. There we can find the bottlenecks and fix them in certain places:

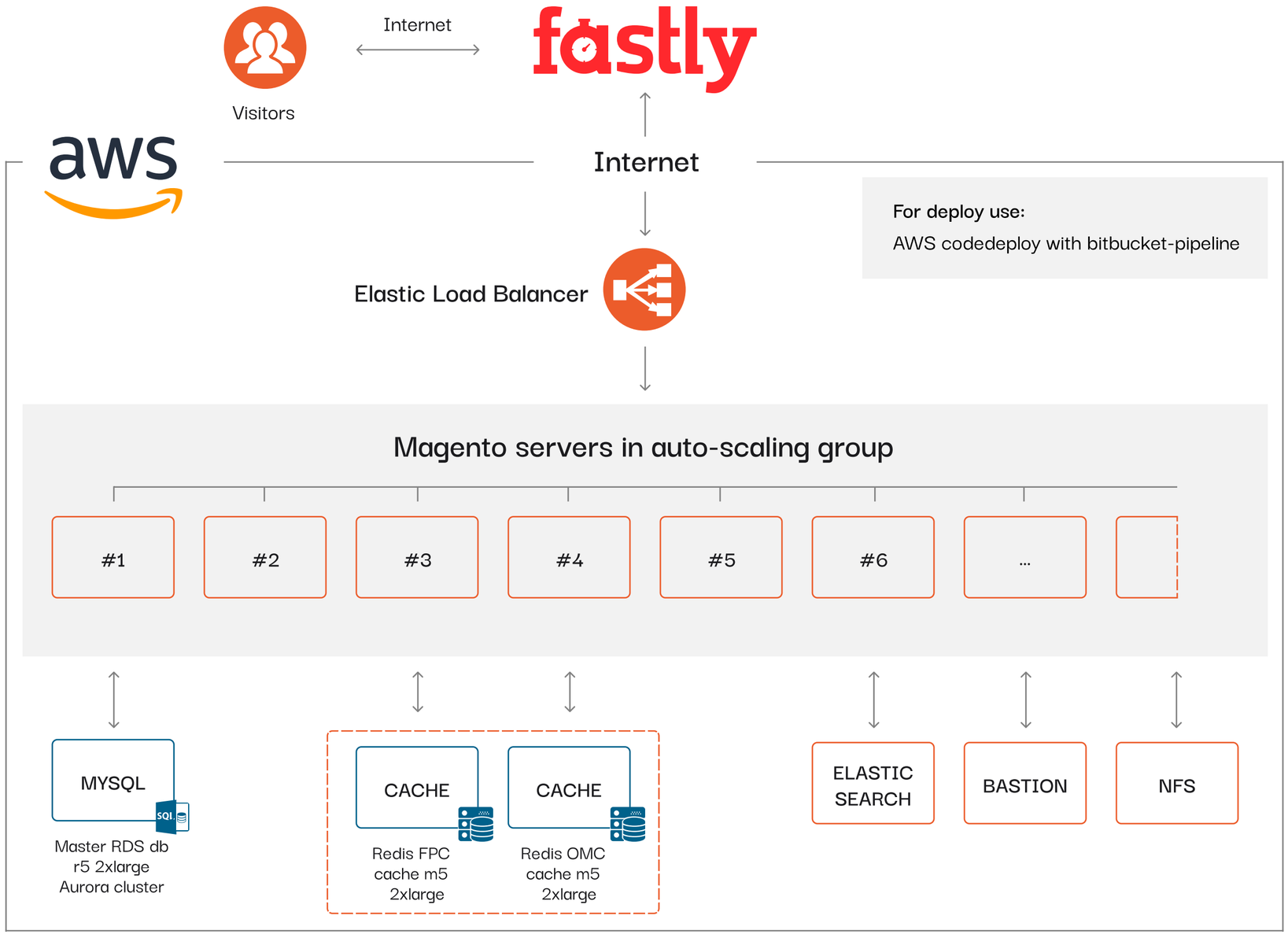
17 ## Optimization results

As a result, we introduce a range of changes into the project architecture:

- land on one database server instead of two
- replace three caching servers with one
- move the static content to the separate server by implementing the NFS media library

18 ## Visualization of the optimized architecture

Eventually, the system turns out to be even more optimal than planned. This is how it looks.


Internet
Visitors
fastly
aws
Internet
For deploy use:
AWS codedeploy with bitbucket-pipeline
Elastic Load Balancer
Magento servers in auto-scaling group
#1
#2
#3
#4
#5
#6
...
MYSQL
Master RDS db
r5 2xlarge
Aurora cluster
CACHE
Redis FPC
cache m5
2xlarge
CACHE
Redis OMC
cache m5
2xlarge
ELASTIC SEARCH
BASTION
NFS

RESULTS

magecom

01 Made it possible for the online store to work well when it’s loaded with **10,000 simultaneous users**

02 Completed the project from start to finish in a month

03 Assisted the client’s team with the extra development hands to work on the AWS configuration and meet their deadline

04 Described all the processes, sent instructions, and trained the client’s team so that they can smoothly work and support the project without our engaging in the future

05 Improved the project infrastructure, while working on it

06 Found a way to optimize the project pricing for the client and decrease the costs

Let’s talk about your business

Choose the most convenient way of communication for you – write an email or contact us in one of the messengers. We’ll discuss your project – provide individual calculations and offer our suggestions on how to upgrade your business.

Website: **magecom.net/contacts/**

Email: **info@magecom.net**

Phone: **+44 74 9143 5563**

magecom

Your Global *Ecommerce* Partner