Case study: a Fortune 500 E-Commerce Company Moves To A Microservice Architecture and uses Traffic Parrot

A Fortune 500 retail e-commerce company software architects purchased Traffic Parrot Enterprise to speed up their software delivery process and reduce the costs of development and testing. In this article, we will explore the details of why they chose Traffic Parrot and how they have used it to help with development and testing of their microservice architecture.

Executive summary:

- Traffic Parrot is a tool used by developers to deliver high quality microservices faster
- Traffic Parrot is designed for autonomous product teams developing microservices, other service virtualization tools do not work well in those environments
- Purchasing Traffic Parrot resulted in faster time to market and lower long term maintenance costs

Value to the IT architects

*'We are building a new platform based on microservices and needed a testing infrastructure that allowed all of our testers to work autonomously, using local machines to record their communication streams between their services and inside our pipelines in order to virtualize a service. Traffic Parrot, while not as fully featured as offerings from HP and IBM, offered the key capabilities that our teams needed such as HTTPS and JMS IBM MQ and a substantially smaller footprint that allowed it to run locally. We have been very pleased by Traffic Parrot's email support and responsiveness to our requests and would recommend them to other teams doing microservice-based architectures." **Chief Architect at a Fortune 500 compan***y

Developers working for the company are building microservices that handle the backend processing of transactions. The e-commerce website is based on the IBM WebSphere eCommerce platform. The applications they work with communicate via HTTP JSON REST APIs, GRPC, ActiveMQ via JMS and IBM MQ via JMS. All microservices are deployed in OpenShift in Docker containers.

One of the challenges with moving to a microservice architecture is that there are many components that need testing. In most cases, microservice architectures will require a Continuous Delivery environment with a Continuous Integration build pipeline for every microservice. That means we will need to test microservices in isolation in automated builds. They will be also developed and tested on developers' desktops or laptops.

This particular company had the following technical challenges:

- Need to create mockups for HTTP, GRPC, JMS ActiveMQ and JMS IBM MQ services
- Need to deploy in Docker to OpenShift (on Red Hat Enterprise Linux Atomic Host)
- Need to test in Bamboo
- Need a graphical/web user interface in the mock
- Need dynamic responses (scripting) in the mock
- Need to choose JMS target queue name based on message content

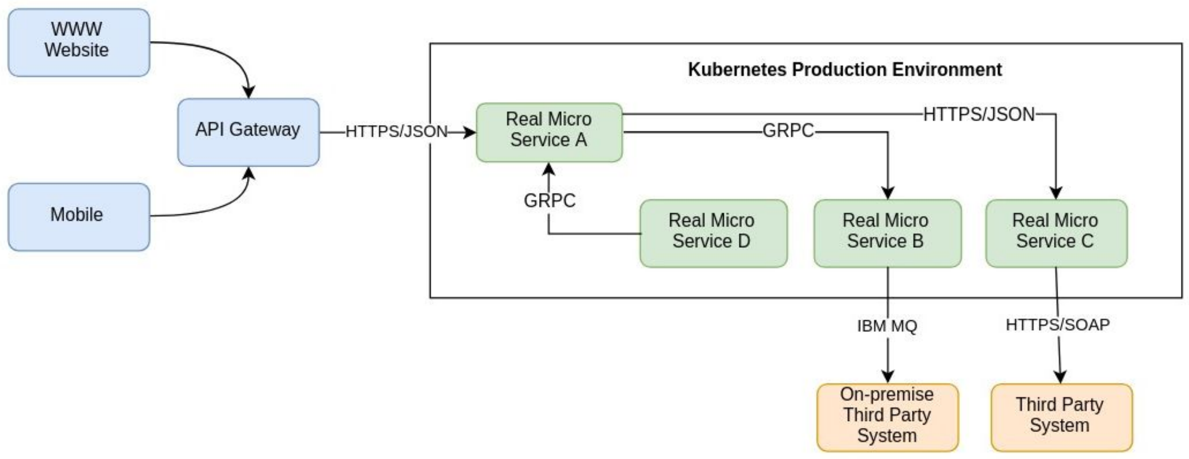

Diagram 1: Production Environment

There are open-source tools on the market that can be deployed using Docker or run in CI. Unfortunately, they come with limited support for different protocols; you would have to use several tools and develop missing protocols in-house. They also often come with no UI or commercial support.

There are other commercial tools that support many protocols and provide advanced dynamic response templating. They also provide richer user interfaces allowing to develop more complex workflows without having to program in Java. Unfortunately, they typically require developers to use thick clients for creating the virtual services, which requires everybody to install them and pay for the licenses. They are also designed to be deployed in a central place managed by one team of administrators rather than running on any laptop or inside an automated CI build. Also, the mock definition artifacts they produce, compared to open source alternatives, are not easy to version control in Git or Subversion.

It was important for the architects to give more autonomy to the development teams and not having a centralized tool to create the virtual services and API mockups. They did not want to create another bottleneck, the centralized administrators team.

They have decided not to build the solution in-house and instead focus on development of the e-commerce platform, and use external tools like Traffic Parrot to reduce the time to market and decrease long term maintenance costs.

They also did not need all the features that the other bigger tools provide.

Traffic Parrot was chosen by the company because it gave them a mix of what the well-established commercial and open source tools provide. It has more functionality and protocols than the open source tools. It also has a more flexible deployment and licensing model than the other commercial tools. The architects wanted a tool that was simple to use and lightweight. Traffic Parrot satisfied all of those requirements.

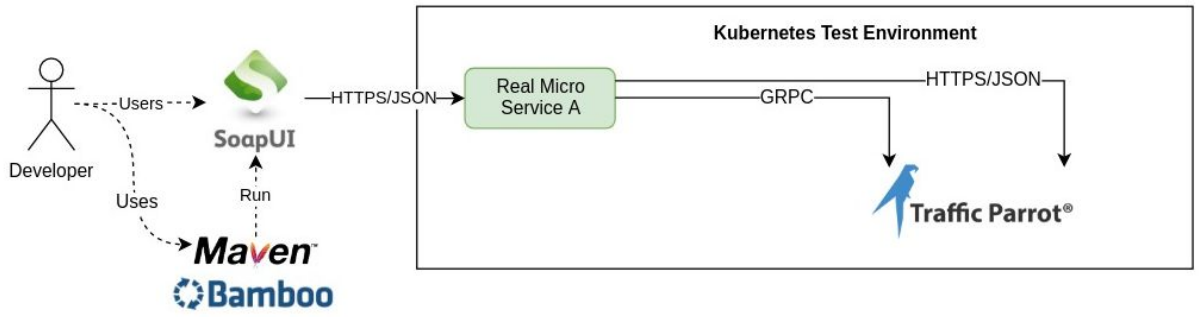

Diagram 2: Testing microservices in isolation

How developers use Traffic Parrot at the company:

- Develop the microservice tests in SoapUI on their laptop
- Start Traffic Parrot on their laptop
- Use the Traffic Parrot Web UI to record traffic (which could be many different protocols) and create mocks by running the SoapUI tests with the microservice that will hit Traffic Parrot in recording mode
- Push the mock definitions (request-response mappings) to Git

How the tests are run in Bamboo at the company:

- Checkout SoapUI tests and Traffic Parrot mocks from Git
- Maven runs Traffic Parrot plugin to start the HTTP, ActiveMQ and IBM MQ mocks
- Maven SoapUI plugin runs the tests that hit the microservice
- The microservice communicates with Traffic Parrot, which is pretending to be the real HTTP, ActiveMQ, and IBM MQ dependencies

Value to the business

The multinational e-commerce marketplace is competitive. Moving to a microservice architecture has resulted in increased productivity of the IT software department, resulting in decreased costs and time to market for both new and existing products.

To make that transition smooth and fast, the company needed tools that are designed to be used in microservice architectures by a team doing Continuous Integration.

The commercial offerings are too heavyweight to be used in the CI environment the company has in place. Also, the licensing costs are too high to justify the value of those tools.

The architects reached out to the Traffic Parrot team for a beta version with JMS support (currently JMS support is not in beta any more). The architects have decided to purchase an unlimited user license for Traffic Parrot. Over a period of 6 weeks, the Traffic Parrot team delivered the missing functionalities as per the requirements of the company architects.

It is difficult to attract and retain development talent. The company has chosen to pay for Traffic Parrot to get guaranteed support and avoid spending internal development time and energy on what would essentially be re-inventing the wheel. They focus their talents on additions to their core offering that increase the value to their end customers.

Summary

If your company is moving to a microservice architecture, you might need more than what the open source tools provide. If you need an API-mocking tool that supports many protocols but is also user-friendly UI and designed to be used with microservices and decentralized autonomous teams that follow CI, Agile and DevOps consider using Traffic Parrot.

Next steps

- Download Traffic Parrot trial
- Contact Traffic Parrot to schedule a demo
- If you have any requirements that are missing in Traffic Parrot, please contact us to discuss potential roadmap acceleration