International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 07 | July 2022 www.irjet.net p-ISSN: 2395-0072

Scale and Load Testing of Micro-Service

Mohan M S[1], Somesh Nandi[2]

Dept. Electronics and Communication Engineering, R.V College of Engineering, Banglore-560050

Abstract - *Representational State Transfer (REST) Application Programming Interface (API) based micro services are more used in cloud applications development due to their inherent benefits. The advantages include scalability, independent development of micro-service. The designed system testing should be validated for functionality and load handling capability. Load testing framework was designed using the Locust framework. Developed framework could generate 2,000 requests per second per container initialized in 4 GB RAM system. This system was used to generate the load for a micro-service. The load was applied to the micro-service to test scaling features. For testing scale the application was deployed from 1 pod to N pod configuration. For each configuration load was applied and response for each configuration was analyzed and mapped. Testing on higher number RPS involved delayed response.*

Key Words: REST, API, autoscaling, containerization

1.INTRODUCTION

Web Platform for providing services and information is common for past decade. Every company wants to host its services online as internet has reached every corner of the world. Building a web platform would guarantee accessibility for every consumer. Designing of a Web Platform for a company to handle millions of customers requires latest technology, which can handle that much amount of load. Scaling a monolithic software application is difficult for large number of users. Hence developing application using microservice architecture ensures scalability [1], load balancing, faster deployment and lower troubleshooting times.

2. MICROSERVICE ARCHITECTURE

Monolithic application were the predominant ones before arrival of micro-service based applications. Earlier an enterprise application was designed as a single software [2]. Monolithic applications gave rise to various issues such as scale problems and dependency between software development teams[3]. Since an enterprise application can be seen as integration between various components. Idea of developing these components/services separately was invented. Micro-service contains various components which are loosely coupled and independently scalable. The micro-services would communicate between them using HTTP REST protocols such as AMQP. To achieve decoupling each service has separate databases. Consistency in the data is maintained using various saga mechanism for multiple services which use same database.

REST Architecture is based on web standards and uses HTTP Protocols. In REST everything is a resource and accessed through REST APIs [4]. Server does not maintain any state of the client. Client needs to maintain the state and based on its state, request is made to the appropriate API [5]. Resources in REST can be anything like JSON, HTML, text. Most used is the JSON response. These are the 4 commonly used HTTP methods.

1.GET – used to access resource, performs read-only operation.

2.POST – provides access to create resource.

3.DELETE – provides access to delete existing resource.

4.PUT – provides access to update or create a resource.

3. SOFTWARE TOOLS

Development of framework for scale testing involves different software dependencies. Software used in this project are Pytest, Locust and Docker.

3.2.1 Pytest

Pytest is python framework for writing functionality and unit tests for an application. It provides lot of customization using which we can achieve application specific test cases. Pytest provides fixtures using which common setup for each test case can be written. Fixtures can also be defined in various levels such as testcase-level, module-level and testrun-level. It also provides dependency decorators using which dependency between test cases can be implemented. The Pytest framework can be used to generate allure report for testcases easily. While running tests in module level all the files starting from python gets executed for pytest tests. For a file all the test function names starting from test or functions using pytest decorator are considered for tests. For writing test the function which needs to be tested should be called inside the test function and its output should be asserted with the expected output. After running the test, Pytest framework would show the list of failed tests with where it has failed. If a statement fails, it also mentions what is the expected assertion. A test can have multiple assert statements or check statements. If one of them fails then the entire test fails. Pytest's debug options are very useful to debug it.

1) Pytest output formatting: Pytest provides robust output formatting with various command-line arguments. The framework provides methods to define what all needs to be shown in the traceback. Using "-v" flag the verbosity of the Pytest tests can be controlled. If "-f" flag is used to generate output, then other jobs like Jenkins can interpret the output files. It also provides various plugins which can be used to automate with other automation jobs.

3.2 Locust

Locust is a python-based load testing framework which can be used to test various features of application by varying the load. The test scenarios can be written in the python language. It can be used to generate distributed system load, with master and slave nodes. A general web-based ui can be generated with locust framework. Load can be scaled by increasing the number of worker nodes which are generating load. The test is conducted by swarming a web application with lot of requests. The system on test can be written in any language. The locust system is completely customizable with various features.

3.2.1 Locust Web UI

Locust has a built-in module to display the test results in web UI. The Web UI has various features such as response time analysis, Exception-analysis, graphical analysis and Error analysis. Web UI can be disabled for the test to print only the test results at the end. It can be configured to modify the number of users, duration on the web UI itself. The tests can be run in headless mode where the test results are shown only at the end of the tests. If the tests are run in multiple machines, then the master node looks after UI maintenance and worker nodes look after load generation. Worker nodes communicate with the master for sending test reports. The primary job of workers is to span users specified by master. Locust test can be run in Docker and Kubernetes environment by creating docker image.

4. TESTING MICROSERVICE

Microservice testing needs to be done at functionality, load handling capability and scale. Functionality verification involves writing test-cases for user transactions. Testing a greater number of users for deployed application involves load testing.

4.1 Test framework for API

Pytest framework is used for automation and testing of the APIs. Pytest provides various features to run series of test in a customized way. The test results are uploaded in the allure report for the summary. Requests python module is used in the test framework to make all the web requests. Initially session is requested, and the web token is created. On reference to which API is being tested, a request body is sent. For given API both positive and negative test cases are written. Written test cases should cover all these responses from the API while testing. For positive test cases the request is generated with proper request body and sent. Then received response body is verified with the expected response. For verifying response status code along with each response parameters are asserted. Some of the APIs are used by admins only for testing these APIs authentication should be generated in a different way.

4.2 Scale and Load Tests

Scale testing and load testing are important tools to identify how the application behaves in deployment. In load testing the application is swarmed with requests and the response time of application is checked. The load is created by creating virtual users on the system which is used for testing and then each user is assigned with task. The task is iteratively performed by the virtual user that has been created. Locust python framework is used for the scale and load testing in the project. Locust library provides user and taskset classes which can be utilized to design the tests. In scale testing the number of Kubernetes pods deployed for application is increased and response will be monitored and verified. The response of the application with different number of pods is analyzed. Based on the analysis scalability of application will be decided. For any load test, application needs to be applied with load. In this case load is web requests. In real world deployment many users log in and do transactions with the application. To imitate that experience in test environment a system is designed to spawn multiple process with each process corresponding to individual user. To obtain credible test results large number of login credentials are stored in the file. This file is used to generate the independent processes in a machine and each user performs set of tasks iteratively. Locust provides user class which is inherited by all the user processes in the framework. Inherited class contains the details of the user login credentials and other required information for the login process.

4.2.1 Virtual users

For any load test, application needs to be applied with load. In this case load is web requests. In real world deployment many users log in and do transactions with the application. To imitate that experience in test environment a system is designed to spawn multiple process with each process corresponding to individual user. To obtain credible test results large number of login credentials are stored in the file. This file is used to generate the independent processes in a machine and each user performs set of tasks iteratively. Locust provides user class which is inherited by all the user processes in the framework. Inherited class contains the details of the user login credentials and other required information for the login process.

4.2.2 Tasksets

Individual process needs to perform set of tasks in iterated manner. Set of tasks contains transactions that regular would do while using notifications-service. All the tasksets which user needs to use should inherit from taskset class. Methods named onStart and onStop are used to create setup and cleanup for the tests. In onStart login and authorization for the user is done. In onStop the session is ended and all the resources which are being utilized are released. Taskset class contains a special class called sequential which is used to ensure tasks are executed in pre-defined order. Locust also provide another class called Fast Sequential which is light weight in number of tasks it supports. But using this class more virtual users can be created per system. Hence more loads can be generated.

4.2.3 Master and Worker load tests: Complex application's real time load cannot be generated in single machine. Multiple machines should work together to generate required amount of load. This configuration of multiple machines in locust is possible with master-worker mechanism only. Master consolidates test results and writes to the web UI where test results are displayed. It also assigns each worker number of users it should spawn. Worker spawns the number of processes it is assigned and sends the report to master. While starting the load testing master needs to given the number of workers and their IP address. While configuring all the nodes IP address should be mapped properly. It should also be ensured that inter-node message delay is minimal.

5. RESULTS

5.1 API Test results

Pytest framework was used to design the test automation for designed APIs. In Pytest automation tests are written as series of tests. Tests are run every time a new code needs to be verified. This ensures that new code modifications have not changed existing functionality. Pytest tests are configured to generate test results as allure report. Allure report indicates which are the tests that have failed. It also displays the traceback of the failures. The test outcomes classified in three categories. those are success, warning and failure. 59 test cases are written for the APIs. These tests include both positive and negative test cases. This test includes verifying of response body along with status codes. Pytest's test results customization was used to write test results as allure reports.

5.2 Locust results

Locust scale and load testing was done for the application. The load testing was done by increasing the number of users and analyzing response from the server. In the figure 1 locust test results are being displayed. It is the result of load testing on single API. There are 120 users who are spawned as virtual processes in the machine. These users are independent processes who are sending requests to the server. In this case all the users are sending request to single API. The test result contains.

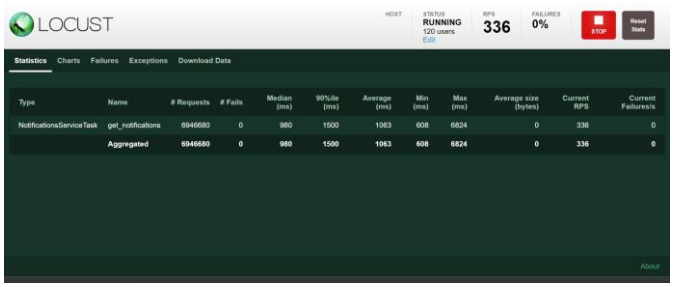

Fig-1: Locust test result display

1. Type contains APIs that are been swarmed by requests.

2. Requests contains total number of requests that have been made to the API.

3. Fails contain the requests which have response code other than 200 HTTP status codes. That is number of requests which the server has failed to respond.

4. Median, Min and Max columns represent how much time the response has taken for being processed. For all request's max, min and average is calculated per API and displayed in the UI.

5. It also shows current API requests per second and failures per second

Web UI also has other tabs for showing error instances. It has charts which shows the response and requests details

with time. It has error tab which shows the error occurrences along with the trace back. It also shows exceptions that occurred along with the trace back of the exception. It also has the option to download data of entire session in excel file.

5.3 New Relic analysis of performance

The App side response along with time taken by each component is recorded by New Relic. New Relic monitors the performance of application and displays it in user friendly UI. Application needs to be integrated with New Relic to see the application's performance. In the figure 2 there is an instance of New Relic graphical representation of performance. It has breakdown of application's response time with various components of app. It is easy to analyse which components needs to be optimized. So while running scale and load test, New Relic data was monitored to see the bottlenecks in the app performance.

Fig -2: New Relic results.

6. CONCLUSION

Developed framework for testing micro-service was used to test notification micro-service. The framework could generate load up to 2000 Requests per second per docker container developed. Detailed framework for testing of functionality and load was mentioned. For testing scale 1 pod to 3 pod configurations were load tested. A system break point was considered when the response time breached 500 milliseconds. Throughput of system for these 3 configurations was tested.

ACKNOWLEDGEMENT

The authors thank Somesh Nandi for the collaboration of preparing this paper and also Hewlett Packard Enterprise for providing opportunity to work in this area.

REFERENCES

[1] Virender Ranga and Anshu Soni. "API Features Individualizing of Web Services: REST and SOAP". In: International Journal of Innovative Technology and Exploring Engineering 8 (Aug. 2019). DOI: 10.35940/ijitee.I1107.0789S19.

[2] Festim Halili and Erenis Ramadani. "Web Services: A Comparison of Soap and Rest Services". In: Modern Applied Science 12 (Feb. 2018), p. 175. DOI: 10.5539/mas.v12n3p175.

[3] Joel L. Fernandes et al. "Performance evaluation of RESTful web services and AMQP protocol". In: 2013

Fifth International Conference on Ubiquitous and Future Networks (ICUFN). 2013, pp. 810–815. DOI: 10.1109/ICUFN.2013.6614932.

[4] Ambar Prajapati."AMQP and beyond".International Conference on Smart Applications, Communications and Networking (SmartNets). 2021, pp. 1–6. DOI: 10.1109/SmartNets50376.2021. 9555419.

[5] Kyrylo Malakhov, Oleksandr Kurgaev, and Vitalii Velychko. "Modern RESTful API DLs and frameworks for RESTful web services API schema modeling, documenting, visualizing". In: International Journal of Advanced Research in Computer and Communication Engineering (Nov. 2018).