UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA DE COMPUTADORES

Curso Académico 2017/2018

Trabajo Fin de Grado

TÍTULO DEL TRABAJO EN MAYÚSCULAS

Autor : Marcos Tenrero Morán

Tutor : Francisco Gortázar

Trabajo Fin de Grado

Título del Trabajo con Letras Capitales para Sustantivos y Adjetivos

Autor : Marcos Tenrero Morán

Tutor : Francisco Gortázar

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Dedicado a

mi familia / mi abuelo / mi abuela

II

Agradecimientos

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Índice general

Índice de figuras

Capítulo 1

Introducción y Motivación

La última era de la informática no ha parado de evolucionar durante las últimas décadas, han surgido y se han puesto en práctica diversas metodologías en el desarrollo del software, desde el modelo de procesos hasta las más actuales como las denominadas metodologías ágiles, formadas por una serie de técnicas y de métodos como Extreme Programming , Kanban o Scrum.

En la actualidad, el uso de las metodologías ágiles está en auge, y esto conlleva la adopción de una nueva manera de concebir la informática, dando prioridad a fases que antes no se tenían en cuenta, como es el caso de la fase de testing.

En el mundo del desarrollo del software, el testing, es quizás, una de las partes más importantes del ciclo, ya que permiten asegurar la estabilidad del producto entre las diferentes iteraciones del desarrollo.

Dentro de esta etapa, existen diversos tipos de pruebas a realizar, pero este trabajo de fin de grado se centra en el testeo y comparación del rendimiento.

En términos de calidad de software, a la hora de realizar un test, se tiene muy en cuenta la pirámide de test, la cual pretende dar unas guías generales a tener en cuenta con respecto a la cantidad de test que se deberían desarrollar para un proyecto, teniendo en cuenta su complejidad de codificación y mantenimiento.

Normalmente se sitúan en la cima esta pirámide los tests de sistema o End-To-End (E2E) ya que el coste de éstos es muy elevado al requerir que todos los componentes de la aplicación se encuentren operativos en el momento de realización de las pruebas y al ser más frágiles. Los test de rendimiento se encuentran justo en esta cima.

El análisis del rendimiento que ofrece una aplicación es crucial para poder comprobar que ésta escala de una manera apropiada con diferentes cargas de trabajo. Normalmente, es un aspecto que se suele olvidar en las etapas iniciales del desarrollo, ya que es un factor que suele preocupar más en aplicaciones que ya tienen una considerable carga de trabajo.

No solo es una etapa relegada debido a que no suele ser un factor preocupante en las primeras iteraciones del desarrollo sino que también resulta complicado el establecer una infraestructura para poder ejecutar, y sobre todo, mantener este tipo de tests.

Este trabajo cubre la realización de una herramienta de ayuda en el proceso del análisis de rendimiento de cualquier aplicación. Pretende ofrecer una manera sencilla de ejecutar estos test de manera automática sin tener que preocuparse de la infraestructura subyacente necesaria.

Normalmente, con los test de rendimiento se pretende simular una gran carga de trabajo sobre la aplicación y esto se traduce en muchas conexiones simultáneas a la misma, y para ello, se necesitan diversos nodos para poder distribuir esta simulación de carga.

La parte mas tediosa de trabajar con diversos nodos es la configuración de éstos para que estén preparados para realizar su cometido.

Por ello, para una efectiva gestión, es necesario poder desplegar y ejecutar estos tests de manera sencilla, y la única manera de hacerlo ágilmente, es contemplar el paradigma de cloud computing, que aporta el concepto de contenedores y un despliegue rápido de infraestructura bajo demanda.

La herramienta que se ha desarrollado durante la realización de este trabajo de fin de grado añade una capa de abstracción a la tecnología de orquestación de contenedores por excelencia, Docker. Lo que se traduce en poder ejecutar cualquier tipo de aplicación que requiera este modelo de despliegue.

Por otra parte, llevo cerca de dos años trabajando en el departamento de automatización y calidad de software en una empresa multinacional de desarrollo de software, estando estos valores y premisas muy presentes en mi vida laboral. En los últimos meses, el análisis de rendimiento ha sido un objetivo muy importante en esta empresa, y mi labor se ha centrado en gran medida en esta tarea, diseñando y aplicando toda la infraestructura necesaria para poner en práctica el proyecto.

Este hecho me ha permitido probar en primera línea los avances sobre mi trabajo de fin de grado en un entorno empresarial real.

Capítulo 2

Objetivos

El propósito general de este trabajo es permitir de una manera sencilla y agnóstica poder ejecutar tests de rendimiento de manera distribuida, con capacidad de crecimiento horizontal en cualquier entorno, incluso en una nube privada y con independencia del sistema operativo ya que en el mundo empresarial, es un requisito bastante demandado.

Existen múltiples herramientas para poder ejecutar tests de rendimiento, incluso de forma distribuida, pero la mayoría requieren una configuración previa nodo a nodo, lo que obliga a una configuración manual de éstos y tener en cuenta la configuración de red entre ellos.

Para poder ofrecer una funcionalidad descrita en los párrafos anteriores, se ha optado por apoyarse en Docker, y en su modo de funcionamiento como clúster, Docker Swarm, lo que ofrece una nube privada creada a partir de las máquinas o nodos que el usuario considere. Permitiendo una gran flexibilidad y haciendo posible el despliegue en diversas arquitecturas.

Las funcionalidades principales que ofrece el framework son las siguientes:

- Interfaz sencilla para el usuario
- Orquestación de contenedores, redes y volúmenes de forma transparente
- Carácter efímero de la infraestructura

Conseguir las funcionalidades descritas conlleva cumplir una serie de objetivos intermedios:

- Construir un Middleware que comunique con la API que expone el demonio de Docker
- Diseñar un plan de acción para la orquestación de contenedores
- Descubrir la configuración de red apropiada para el carácter de la aplicación
- Diseñar una API REST para el consumidor
- Construir un proceso de orquestación de recursos Docker

Además he fijado una serie de requisitos metodológicos para asegurar un desarrollo con la forma mas práctica posible.

- Integración continua desde el comienzo del proyecto
- Gestión ágil del proyecto
- Test-Driven-Development

Capítulo 3

Tecnologías, Herramientas y Metodologías

Aquí me gustaría hablar de los siguientes temas:

1. **Lenguaje Golang:** Carácteristicas básicas del lenguaje, cross-platform, cobertura de test
2. **Git, Travis & Jenkins:** Diferencias y features críticas que me han hecho pasar de Travis a Jenkins
3. **Swagger:**
4. **Goa Design:** Paquete Golang, Design First API, creo que es importante por el carácter de piensa antes de picar, muy similar a TDD
5. **AWS** Toda la CI está desplegada en AWS y tengo pensado montar un cluster Swarm aquí para una futura demo.
6. **Docker**
7. **ElasticSearch** Características básicas
8. **Shared Filesystem:** GlusterFS / Samba . No es Kubernetes, asi que se necesita esta configuración previa para tener todos los ficheros disponibles en todo el cluster. Explicar lo que ofrece y por qué es necesario.
9. **Redes:** Concretamente DNS y algoritmo DNS Round Robin. Es la piedra angular en cuanto a orquestación con arquitectura Master / Workers se refiere
10. **Apache Jmeter:** Forma de ejecución distribuida y arquitectura

3.1. DNS

Domain Name System es un sistema de nombrado de redes IP que se utiliza para resolver nombres en direcciones IP. El servicio de resolución de nombres puede tener diferentes tipos de registros. En la investigación del trabajo sólo se han usado registros de tipo **A**, los cuales traducen un nombre en una o varias direcciones IP, dependiendo de los registros A que tenga almacenados para un mismo nombre de dominio/subdominio.

3.1.1. DNS Round-Robin

DNS Round-Robin es un algoritmo de selección de IP, en el que cada petición que realiza el usuario, obtiene una lista completa de todas las direcciones IP registradas en el servidor de nombres ordenada de tal forma que nunca recibe la misma IP, realizando, de esta manera, labores de balanceador de carga.

Docker implementa este algoritmo de forma alternativa en su DNS interno, y el desarrollador puede elegir si...

Capítulo 4

Descripción informática

4.1. Requisitos

Descripción más detallada de los objetivos explicados en el capítulo de Objetivos

Tareas tablero Kanban del Project de GitHub

4.2. Arquitectura y Análisis

Como el .ºrquestador"se ha centrado en el formato de despliegue Master/Workers, explicación de la forma de ejecución de JMeter en detalle

Explicación de la investigación hasta que di con la forma de orquestar los servicios de manera satisfactoria para nuestro objetivo siguiendo los requisitos de JMeter.

Arquitectura inicial, con algún boceto de arquitectura.

Docker Swarm Architecture + GlusterFS

4.3. Diseño e Implementación

- Diagrama de estados del proceso de orquestación
- Diseño de la API REST, funciones, endpoints...
- Dockerfile JMeter con su script bash
- Dockerfile build Golang + dep dependency manager

- Implementación algoritmo Descubrimiento de contenedores DNSRR

4.4. Integración Continua

Inicialmente, el proyecto, al estar alojado en el servicio de control de versiones de GitHub, en un repositorio de código abierto, se ofrecían diversas opciones SAAS de Integración Continua, entre ellas TravisCI o CircleCI.

Al comenzar el desarrollo, las características que ofrecía TravisCI encajaban perfectamente con los requisitos de Automation Test Queue:

- Pipeline-as-code
- Enfocado a Contenedores
- Integración con GitHub

Se utilizó durante la mitad de desarrollo del proyecto y ofreció muy buenos resultados. Pero llegó un momento en el que era necesario que el host donde se ejecutaban los tests cada vez que realizaba un commit formase parte de un cluster de Docker Swarm, y esto no se podía conseguir con la solución que ofrecía TravisCI.

La limitación de TravisCI era muy importante ya que rompía la línea de Integración Continua, por lo que no quedó mas remedio que optar por una solución mas configurable, Jenkins.

No se optó desde un principio por Jenkins debido a la necesidad de disponer de una máquina con acceso a la red las 24 horas al día y la configuración del mismo.

El despliegue de las máquinas destinadas a la línea de integración continua se realizó sobre máquinas EC2 de Amazon Web Services, todas ellas bajo el mismo grupo de disponibilidad y con un servicio de IP elástica, permitiendo de esta manera que la dirección de acceso a Jenkins fuese estática. Además se incluyó un registro A en el DNS de mi dominio particular para poder acceder con una dirección[1] mas fácil de recordar aún.

[1] http://atq.mtenrero.com:8080

Una parte vital para el correcto funcionamiento de la línea de integración continua fue configurar webhooks en GitHub para apuntar a la instancia de Jenkins además de habilitar la integración propia de Jenkins. Esto permitió que con cada evento en el repositorio, como *commits* o *pull-requests* se enviase una notificación a Jenkins para así poder ejecutar el job especificado y lanzar los tests unitarios sobre todas las ramas del proyecto para asegurar la regresión y un buen funcionamiento del mismo.

Hace apenas dos años, Cloudbees, la organización que es oficialmente responsable del desarrollo de Jenkins, lanzó una característica que llamaron *Jenkins Declarative Pipelines* [2]. Estos pipelines, permiten escribir, de una forma desriptiva, las tareas y procesos a realizar cada vez que el job es ejecutado.

Esto ha permitido describir el workflow de ejecución de manera totalmente agnóstica y compatible con cualquier Jenkins desplegado, siempre que tenga Docker instalado, ya que hace uso del denominado Docker-in-Docker, que permite utilizar el demonio de Docker del host dentro de un contenedor ya existente.

4.5. Pruebas

Hablar de Cobertura de Test, como se ha organizado por paquetes, cómo está integrado con el Pipeline de IC de Jenkins, los triggers que tiene configurado...

Explicar cómo se diseñaron primero los tests básicos, luego la implementación, y después se amplió la cobertura de tests

[2] https://jenkins.io/blog/2017/02/03/declarative-pipeline-ga/

Capítulo 5

Conclusiones y trabajos futuros

Me gustaría hablar de la integración completa con OS Windows, otros tipos de despliegue como en AWS de manera nativa, otros modelos diferentes a la arquitectura Master/Workers...

Quizás de una interfaz web completa con Angular o similar...

Bibliografía

Apéndice A

Manual de usuario

Apéndiceb

Apéndice B

Fragmentos de código

Listing B.1: Jenkisfile CI Docker-in-Docker

```
pipeline {

    agent {
        docker {
            image 'tenrero/golang-dep-alpine:1.10.2'
            reuseNode true
            args '-it -v /var/run/docker.sock:/var/run/docker.sock
               -v $WORKSPACE:/tmp/app'
        }
    }

    stages {
        stage('Prepare Environment') {
            steps {
                sh 'echo $GOPATH'
                sh 'mkdir -p /go/src/github.com/mtenrero/'
                sh 'ln -s /tmp/app
                   /go/src/github.com/mtenrero/ATQ-Director'
                sh 'ls -a /go/src/github.com/mtenrero/ATQ-Director'
                sh 'go get -u github.com/golang/dep/cmd/dep'
                sh 'go get -u github.com/golang/lint/golint'
```

```
                sh 'go get -u github.com/tebeka/go2xunit'
                sh 'go get -u golang.org/x/tools/cmd/cover'
                sh 'go get -u github.com/mattn/goveralls'
            }
        }

        stage('Download Vendor') {
            steps {
                sh 'cd /go/src/github.com/mtenrero/ATQ-Director &&
                    dep ensure'
            }
        }

        stage('Test') {
            steps {
                sh 'cd /go/src/github.com/mtenrero/ATQ-Director &&
                    go test ./... -race -coverprofile=coverage.txt
                    -covermode=atomic'
            }
        }

        stage('Build') {
            steps {
                sh 'cd /go/src/github.com/mtenrero/ATQ-Director &&
                    ./build.sh'
            }
        }
    }

}
```