CACiC 2023
CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN

CRANE: Simplificando el Despliegue de Aplicaciones Contenerizadas en Entornos Locales

Jose Miguel Silva Pavón[1], Franco Bellino[1], Patricia Bazán[3], Alejandra B. Lliteras[2,4], Nicolás del Rio[1]

[1] UNLP, Facultad de Informática, [2] UNLP Facultad de Informática, LIFIA, [3] UNLP, Facultad de Informática, LINTI, [4] CICPBA

js.silva.010@gmail.com, fran85bellino@gmail.com, pbaz@info.unlp.edu.ar, alejandra.lliteras@lifia.info.unlp.edu.ar, ndelrio@info.unlp.edu.ar

Abstract. CRANE es una herramienta diseñada para el despliegue local de aplicaciones en contenedores que busca simplificar las pruebas de entornos distribuidos de forma local. A diferencia de herramientas tradicionales como Minikube. El diseño de CRANE ofrece una solución liviana y de propósito general con capacidades de escalado automático, orientada a estudiantes y desarrolladores que requieran crear y desplegar stacks completos de aplicaciones. CRANE tiene como propósito, también, facilitar la incorporación de habilidades de *devops*, que acelera el proceso de desarrollo de software en un marco de entrega contínua.

En este trabajo, se presenta una API REST, construida para permitir la creación y el despliegue de servicios Docker, así como la monitorización, definición de políticas de escalado y gestión de alertas para la toma de decisiones.

Palabras Claves: Docker, Escalado, Monitorización, Api Rest, Devop, Minikube.

1. Introducción

En la era moderna de la tecnología, los servicios de Plataforma como Servicio (PaaS, por sus siglas en inglés Platform as a Service) han revolucionado el paradigma de desarrollo de software, proporcionando entornos completos de desarrollo y despliegue en la nube. Esta flexibilidad y comodidad han permitido a los desarrolladores centrarse en la lógica de aplicación, liberándose de las complejidades inherentes al manejo de la infraestructura. Sin embargo, cuando las necesidades no implican el uso de un PaaS, y es necesario trabajar con la infraestructura, surgen grandes inconvenientes:

- Migración del entorno a diferentes plataformas, cada una de ellas con diferentes características. Esto implica que haya archivos de configuración diferentes para cada entorno e instalación de dependencias propias donde se vaya a utilizar.
- Dificultad para llevar a cabo pruebas de rendimiento efectivas en etapas tempranas y construir aplicaciones verdaderamente escalables.
- La migración de aplicaciones entre hosts diferentes ha demostrado ser una tarea desafiante debido a las dependencias específicas del sistema, como las versiones de librerías y los sistemas operativos.

A raíz de estos desafíos, los contenedores de Docker han surgido como una solución prometedora, permitiendo el despliegue de aplicaciones en cualquier host con Docker instalado, independientemente de sus características específicas. Sin embargo, esto ha llevado a la necesidad de orquestar, medir y escalar estos contenedores de manera eficiente, un problema que las soluciones actuales como Kubernetes, han podido resolver pero con un alto costo en recursos computacionales. Asimismo, la configuración de estos contenedores frecuentemente requiere la creación de archivos de configuración estáticos que dependen de rutas locales, lo que complica aún más el proceso.

Debido a estas necesidades, surgen nuevos roles, como el de DevOps (Development and Operations) que une roles que anteriormente estaban separados (desarrollo, operaciones de TI, ingeniería de la calidad y seguridad) en un solo rol y así tener una visión completa de todo el sistema a desarrollar.

En este contexto, se propone una innovación en la herramienta CRANE, originalmente presentada por [Arcidiacono et al., 2022], para mejorar el proceso de despliegue y control de servicios.

CRANE es un diseño de solución que se compone de un Front-End y un Back-End. El presente trabajo se basa en la parte Back-End de CRANE.

Esta propuesta abarca los siguientes puntos:

- Una API para la gestión de los contenedores, por lo tanto la interacción entre el usuario y la herramienta es transparente e independiente de donde se ejecuta.
- La medición del estado de los contenedores está integrada en la herramienta, y en caso de que esa medición arroje la necesidad de instanciar contenedores, esta se realiza automáticamente.

Este documento se organiza de la siguiente manera: en la sección 2 se enumeran las metodologías de despliegue de aplicaciones actuales, así como sus características principales. En la sección 3 se analiza el diseño de CRANE y qué lugar ocupa en el despliegue actual de aplicaciones. En la sección 4 se detalla la implementación técnica y que herramientas se utilizaron para su desarrollo. En la sección 5 se explica que futuras implementaciones se pueden realizar a partir de CRANE.

2. Conceptos vinculados al despliegue de aplicaciones

El despliegue de aplicaciones es una fase crítica en el ciclo de desarrollo de software, ya que pone en funcionamiento en entornos de producción, lo desarrollado. En los últimos años ha habido un avance en las técnicas y en las herramientas usadas para que este proceso se pueda realizar de una manera rápida, eficiente y confiable.

A continuación se listan algunas de ellas:

Despliegue Continuo: El despliegue continuo consiste en la automatización del proceso de entrega e implementación de código en producción de forma constante y rápida. Esta metodología se adopta para liberar actualizaciones en producción con mayor frecuencia, lo que permite una “mejora continua” de la aplicación y mayor respuesta.

Contenedores: La tecnología de contenedores popularizada por Docker, ha revolucionado el despliegue de aplicaciones, ya que permite empaquetar todo el entorno de ejecución de la aplicación y sus dependencias en un contenedor aislado. Esto nos garantiza portabilidad y consistencia de las aplicaciones en diferentes entornos y es el tipo en el que se basa CRANE.

Microservicios: Los microservicios son una arquitectura de diseño de software donde las aplicaciones se componen de pequeños servicios independientes que se comunican entre sí, mediante interfaces bien definidas. Esto nos permite poder desplegar cada servicio de manera independiente, lo que facilita la escalabilidad y mantenimiento de la aplicación.

Infraestructura como código (IaC): IaC es una práctica en la que toda la infraestructura necesaria para ejecutar una aplicación se define y se gestiona mediante código. Herramientas como Terraform y Ansible permiten crear y configurar recursos de infraestructura en la nube de forma automatizada.

Monitoreo: El monitoreo de aplicaciones es fundamental para asegurar un despliegue exitoso y la posterior corrección de errores, usando mecanismos de monitoreo en tiempo real.

Servicios en la nube: El uso de servicios en la nube, ha simplificado el despliegue de aplicaciones. Servicios como AWS, AZURE y Google Cloud ofrecen plataformas con herramientas que facilitan el despliegue y gestión de aplicaciones, reduciendo la complejidad.

3. El diseño de CRANE para el proceso de despliegue de aplicaciones

El diseño de CRANE propuesto en este trabajo se enfoca en mejorar el proceso de despliegue de aplicaciones contenerizadas, así como su monitoreo y escalado, haciendo énfasis en la transparencia y facilidad de uso para el usuario. CRANE se basa en la implementación de dos componentes principales: un Back-End para la creación y despliegue de contenedores Docker, y un Front-End destinado al usuario final.

El componente de Back-End, utiliza Python y FastAPI[1] para proporcionar una API que permite la gestión de contenedores Docker. Esto incluye la creación, despliegue y monitoreo de contenedores, así como la definición de políticas de escalado y la gestión de alertas.

El componente de Front-End, a través de la API de CRANE, permite al usuario interactuar con la herramienta de forma transparente e independiente del lugar donde se ejecute. La interacción con el usuario se realiza mediante una interfaz web intuitiva

[1] https://fastapi.tiangolo.com/

y fácil de usar, diseñada para simplificar la experiencia del usuario y reducir la curva de aprendizaje.

Para el monitoreo y escalado de los contenedores, CRANE utiliza Prometheus[2] para recoger métricas de rendimiento y Alert Manager para gestionar alertas basadas en esas métricas. Además, para el balanceo de carga entre los contenedores y el enrutamiento de las solicitudes, se utiliza Traefik[3]. De esta forma, CRANE proporciona un entorno de despliegue completo, incluyendo balanceador, métricas, alertas y más, como se puede ver en la arquitectura mostrada en la Figura 1.

Todo esto se encapsula en un Docker Compose que se genera automáticamente para cada despliegue, lo que permite a CRANE desplegar un stack totalmente configurado de manera automática y sin necesidad de intervención manual por parte del usuario.

4. CRANE como servicio: tecnologías seleccionadas

La primera versión de CRANE se basaba en el uso de un script de consola en bash que empleaba comandos Docker para instanciar contenedores, así como para escalar y desescalar. Sin embargo, esto exigía que el administrador tomara decisiones continuamente sobre cuándo escalar o desescalar una instancia. Para superar esta limitación, se desarrolló la segunda versión de CRANE, capaz de escalar automáticamente mediante métricas de uso, tales como la cantidad de peticiones, tiempos de respuesta, uso de CPU o memoria, en una forma similar a Keda[4] en Kubernetes.

A pesar de esta mejora, la segunda versión tenía el inconveniente de estar vinculada al sistema operativo en el que se usara. Por lo tanto, se desarrolló una tercera versión, con una API REST y una interfaz gráfica para interactuar con Docker mediante mensajes HTTP.

Para este trabajo, se utilizó como punto de partida esta tercera versión de CRANE. La arquitectura se organizó en torno a un Proxy, como se puede ver en la Figura 1, que recibe peticiones y balancea la carga en función de métricas recogidas por Prometheus y Alert Manager, además de actuar de acuerdo a políticas preconfiguradas mediante Open Policy. Todo esto se configuró a través de una API REST creada en Python.

Se optó por utilizar Traefik [Sharma, R. y Mathur, A.,2021] en lugar de Nginx como proxy debido a su configuración y gestión simplificada, así como su capacidad para detectar automáticamente nuevos servicios o contenedores y ajustar su configuración, lo cual simplifica considerablemente la administración de la infraestructura.

[2] https://prometheus.io/

[3] https://doc.traefik.io/traefik/

[4] https://keda.sh/

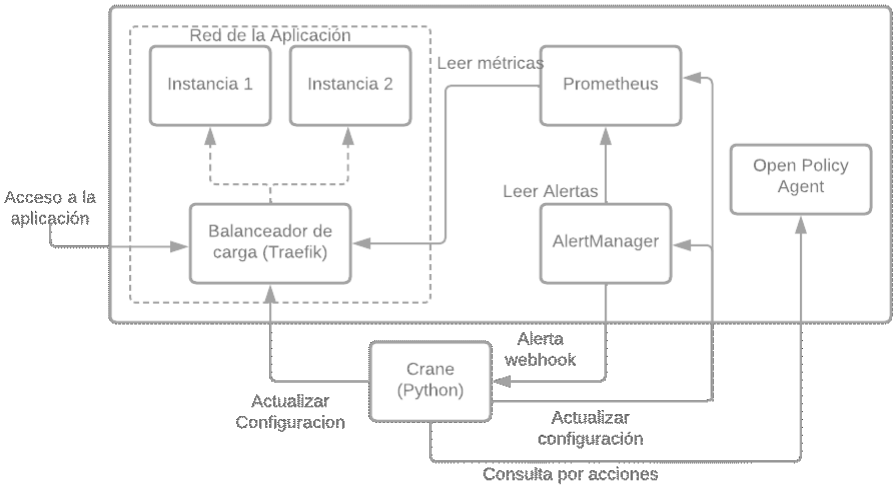

Fig. 1. Tercera versión de la arquitectura CRANE

En el proceso de este trabajo, se pusieron de manifiesto diversos desafíos. El aprendizaje de fastAPI, la comprensión de cómo conectar todas las piezas y la interacción con Docker desde Python ya que presentaron dificultades iniciales. La complejidad de CRANE al generar entornos completos para desplegar servicios exigió un esfuerzo adicional para asegurar que todo encajara de forma automática.

Para solucionar estos inconvenientes se analizó la documentación de fastAPI y Docker, así como de todo su entorno relacionado, recurriendo a la ayuda de la comunidad y experimentando con diferentes configuraciones hasta lograr una integración adecuada. La instanciación de contenedores Whoami y la validación a través de su URL permitieron confirmar que cada vez se devolvía una dirección IP diferente, asegurando que los contenedores fueron instanciados y que la carga se repartía entre ellos.

Las lecciones aprendidas incluyen la importancia de una planificación cuidadosa, la necesidad de un profundo conocimiento en las herramientas utilizadas y la comprensión de que la experimentación y la perseverancia pueden superar los obstáculos técnicos.

Este proyecto permitió una comprensión profunda de la orquestación de contenedores y brindó la posibilidad de desarrollar habilidades valiosas en la automatización y escalado de infraestructuras.

5. Conclusiones y Trabajo Futuro

En este artículo, se presentó una forma de automatizar el despliegue de aplicaciones contenerizadas y posterior gestión de métricas, de forma que sea transparente al usuario. Primero se partió del esquema definido en CRANE y se comenzó a conectar las diferentes partes que componen una simple aplicación. Una vez instanciada la app se

implementó una API Rest que levanta la app, solo haciendo una llamada a un endpoint. Se obtuvo un resumen de lecciones aprendidas, que pueden ser reusadas en personas que se sumen a proyectos de estas características. Como trabajo futuro se pretende implementar una interfaz web usando React que pueda funcionar con CRANE de una manera totalmente visual y disminuir la curva de aprendizaje de la app por parte de los usuarios.

Referencias

1. Arcidiacono, J., Bazán, P., del Río, N., & Lliteras, A. B. (2022). CRANE: A Local Deployment Tool for Containerized Applications. In Conference on Cloud Computing, Big Data & Emerging Topics (pp. 58-71). Springer, Cham.
2. Arcidiacono, J., Lliteras, A. B., & Bazán, P. A. (2020). DEHIA, una plataforma para la generación y ejecución de actividades de recolección de datos con intervención humana aplicada en el Programa E-Basura. In VI Simposio Argentino de Ciencia de Datos y GRANdes DAtos (AGRANDA 2020)-JAIIO 49 (Modalidad virtual).
3. Bullington-McGuire, R. and Dennis, A.K. and Schwartz, M. (2020). Docker for Developers: Develop and run your application with Docker containers using DevOps tools for continuous delivery. Packt Publishing.
4. Burns, B., Beda, J., Hightower, K., & Evenson, L. (2022). Kubernetes: up and running. " O'Reilly Media, Inc.".
5. Ferraiolo, D., Cugini, J., & Kuhn, D. R. (1995, December). Role-based access control (RBAC): Features and motivations. In Proceedings of 11th annual computer security application conference (pp. 241-48).
6. Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT), 2(2), 115-150.
7. Httermann, M. (2012). DevOps for developers. Apress: delivers a practical, thorough introduction to approaches, processes and tools to foster collaboration between software development and operations
8. Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An introduction to docker and analysis of its performance. International Journal of Computer Science and Network Security (IJCSNS), 17(3), 228.
9. Rani, D., & Ranjan, R. K. (2014). A comparative study of SaaS, PaaS and IaaS in cloud computing. International Journal of Advanced Research in Computer Science and Software Engineering, 4(6).
10. Reis, D., Piedade, B., Correia, F. F., Dias, J. P., & Aguiar, A. (2021). Developing docker and docker-compose specifications: A developers' survey. IEEE Access, 10, 2318-2329.
11. Sommerlad, P. (2003, June). Reverse Proxy Patterns. In EuroPLoP (pp. 431-458).
12. Sharma, R., Mathur, A., Sharma, R., & Mathur, A. (2021). Traefik for Microservices. Traefik API Gateway for Microservices: With Java and Python Microservices Deployed in Kubernetes, 159-190.