# Implementing CRANE: a tool for simple deployment of containerized applications in local environments

José Miguel Silva Pavón[1][0009-0004-0240-9448], Franco Bellino[1][0009-0000-0286-4038], Patricia Bazán[3][0000-0001-6720-345X], Alejandra B. Lliteras[2,4][0000-0002-4148-1299], Nicolás del Rio[1][000-0002-0889-0752]

[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 is a tool designed for local deployment of containerized applications to simplify testing of locally distributed environments. CRANE's design offers a lightweight, general-purpose solution with automatic scaling capabilities, oriented to students and developers who need to create and deploy complete application stacks within a controlled environment. CRANE is also intended to facilitate the incorporation of DevOps skills, which accelerates the software development process in a continuous delivery framework.

**Keywords:** API REST, Local Deployment, DevOps, Docker, Scaling, Monitoring, Cloud Infrastructure, Training, Kubernetes.

## 1 Introduction

PaaS (Platform as a Service) services [1] revolutionized the software development paradigm in this digital era by providing environments that come equipped with tools to develop and deploy code directly in the cloud. This flexibility and convenience allow programmers to focus on application logic and abstract from the complexities of infrastructure management. In addition, it allows providing better quality products, as they are considered agnostic to the cloud where they are executed, inherent issues of current technologies.

However, for different reasons, in early stages of development it may be necessary to work with this infrastructure in local environments. In these cases, the migration of the environment to different platforms requires different configuration files, as well as installing system specific dependencies, library versions and considering different operating systems. It also makes effective performance testing more difficult.

As a result of these challenges Docker containers [2] emerge as a solution, allowing the deployment of applications on any host with Docker installed, regardless of their specific characteristics. This leads to the need to orchestrate, measure and scale these containers in a more efficient way, a problem that current solutions such as Kubernetes [3] [4] solve, but at a high cost in computational resources. The configuration of these containers requires the creation of static configuration files that often rely on local paths, further complicating the process.

Due to the aforementioned needs, new roles arise in the work teams such as DevOps (Development and Operations) [5] that unites roles that were previously separated (development, IT operations, quality engineering and security) in a single role and thus have a complete view of the entire system to be developed.

This is why, in this work, we propose the implementation of CRANE [6], which provides a low-cost solution for the deployment and orchestration of containers locally.

This work focuses on the creation of a tool that has been designed to facilitate the deployment and autonomous management of containers in a local environment. This platform allows the fast and efficient deployment of services in containers that are automatically managed by CRANE without the need for manual intervention by the developer, even in situations of low performance or host system errors.

Containers are connected to each other through a network that makes it easy to obtain and analyze metrics in real time. Any drop in container performance is detected by this metrics analysis, and a set of pre-established policies are triggered to respond appropriately to these events.

System management is performed through a REST API, which connects to the Docker client installed on the operating system through a Python library [7]. These tools together, and their organization in the CRANE architecture, allow performing the described operations in an efficient and automated way.

This paper is organized as follows: Section 2 describes the implementation of CRANE as a tool to simplify the deployment of applications in a local environment and also as a DevOps training platform. Section 3 describes the architecture, components and technology stack used in its implementation. Finally, Section 4 presents conclusions, lessons learned and future works.

## 2 CRANE as a tool for simplification and DevOps training

DevOps is a methodology that combines the world of software development (Dev) and IT operations (Ops) to accelerate the software lifecycle and ensure continuous delivery of quality software. Its main objective focuses on automating processes and increasing collaboration between teams, promoting a culture of shared responsibility throughout the life of the application.

This methodology began to be used around 2007, when the software development and IT operations communities became concerned about the traditional software writing model where developers write the code and work independently from the operations team, who are in charge of implementing and supporting it. This is why the term "DevOps" is a combination of the words development and operations because it reflects the process of integrating these disciplines in a continuous and unified process.

A DevOps team consists of developers and IT operators working collaboratively throughout the product lifecycle, with the goal of increasing the speed and quality of the software implementation. It is a new way of working, a cultural shift, that has significant implications for the teams and the organizations they work for.

Under this approach, development and operations teams are no longer separate. In many cases, these teams are integrated into a single entity in which engineers collaborate at all stages of the application lifecycle, ranging from development and assessment to implementation and operations. This has resulted in professionals possessing a diverse set of multidisciplinary skills, promoting closer and more efficient collaboration.

Development and operations teams use tools to automate and accelerate the various processes to increase reliability. These tools help teams address important fundamentals such as continuous integration, continuous delivery, automation and collaboration.

Because of the "continuous nature" of this methodology, practitioners use the infinite loop to show how the phases of the DevOps lifecycle relate to each other. Although they appear to flow in a sequential fashion, the loop symbolizes the need for constant collaboration and iterative improvement throughout the lifecycle.

### 2.1 Design assumptions

CRANE was born from the design assumptions of [6], and evolved in [8] [9] in response to the need to quickly deploy locally a component-based application with multiple instances, test the behavior against critical errors and also migrate that application in different environments. That is, to simulate a cloud infrastructure in a local environment that does not demand a significant amount of hardware resources.

These design assumptions lead to the following question as a developer: *Are you ready to develop and understand applications running in a distributed environment on a cloud infrastructure?*

Developing software in a local environment versus the cloud involves carefully evaluating several critical factors such as cost, scalability, performance and security. On-premises development can give more fine-grained control over the infrastructure and, in some cases, lower upfront costs. On the other hand, opting for the cloud offers benefits such as greater scalability and flexibility, as well as providing access to advanced technologies that may not be available or feasible locally.

CRANE's approach is to integrate the advantages of both approaches, implementing technologies such as virtualization, automation and resource orchestration and create a local environment that mimics the capabilities of the cloud.

This solution gives developers the advantages of the scalability and flexibility of the cloud, while maintaining comprehensive control over the infrastructure and managing costs more cost-effectively.

CRANE is an educational tool that contains an ecosystem to simulate a cloud infrastructure locally, allowing the user to practice basic development and operations (DevOps) concepts such as: 1- Create and interact with containers, 2- Integrate real-time metrics and alerting systems, 3- Implement decision making based on predefined rules, 4- Create multiple instances of an application to test its performance and 5- Reduce costs when testing an application.

# 3 CRANE: architecture, components and implementation of the solution

CRANE is based on the implementation of two components: a Back-End for the creation and deployment of Docker containers [10] and a Front-End for the end user.

The Back-End component uses Python and FastAPI to generate an API that allows the management of Docker services. Among them the creation, deployment and monitoring of containers, as well as the definition of scaling policies and the management of alerts.

The Front-End component, through the CRANE API, allows the user to interact with the tool transparently and independently of the place where it is executed. The interaction with the user is done through an intuitive and easy-to-use web interface, designed to simplify the user experience and reduce the learning curve. This last component is beyond the scope of this paper and will be addressed in future work.

The CRANE Back-End infrastructure is composed of several interconnected components that work together to provide high availability and efficient response to critical events.

The user communicates with CRANE via a REST API and can create containers and adjust scaling rules depending on their own analysis and/or previously defined rules. You have control of exposed application ports, container traffic and can clone application configurations to generate new instances faster.

Being portable and easily accessible, it can be deployed on any system by simply executing a command. At the same time, the generated Docker Compose files are available to be taken to any other machine without having to take CRANE out of the local development environment.

## 3.1 – Technology Stack

This section briefly describes each of the technologies used for the proposed implementation and describes how the components of the solution relate to each other.

Docker [10] is an open platform designed to automate the deployment of containerized applications efficiently. It facilitates the separation of applications from the underlying infrastructure, thus enabling faster software delivery.

A reverse proxy [11] is a type of proxy server that acts as an intermediary between clients and one or more web servers. Unlike a regular proxy that is positioned between the client and the target server, a reverse proxy is positioned between the client and one or more web servers. Traefik [12] is used.

Prometheus [13] is an open-source system monitoring and alerting toolkit originally created at SoundCloud (SoundCloud is a Swedish-founded German headquartered audio streaming service) but now maintained independently of any company. It collects and stores its metrics as time-series data, i.e., metric information is stored with the timestamp at which it was recorded, along with optional key-value pairs called tags.

Alertmanager [14] is a Prometheus component that manages the generated alerts and allows applying certain actions. For this project, it was fundamental because it

allows action to be taken in the event of any variation in the performance or availability of the active services. The way Alertmanager works depends on the rules defined in Prometheus, it is in charge of analyzing if any of the thresholds defined in the file called rules.yaml are met and to act with the configuration provided by the user. In the proposed implementation, a communication through automated messages (webhook) was used, in which Alertmanager notifies Crane by means of a call to the Alert receiving endpoint when it detects any anomaly.

The integration of Prometheus with CRANE brought benefits as it allows to abstract from several complications when defining an alert system, providing the following advantages: 1 - Receive alerts from Prometheus or other monitoring systems that are configured to send alerts through it, 2 - Can discard identical alerts and group similar alerts into single notifications. This helps reduce the noise generated by multiple similar alerts, 3 - Can send alerts to specific recipients based on certain criteria such as alert labels, priorities, or user-defined settings. This allows sending alert notifications to the right operations teams or appropriate communication channels, 4 - Can send alert notifications to different communication channels such as emails, chat systems (Slack, Microsoft Teams), ticketing systems (PagerDuty, JIRA), and other notification services through custom integrations, and 5 - Offers the ability to temporarily mute or suppress certain alerts to avoid notification overload during planned maintenance periods or known incidents.

Open Policy Agent (OPA) [15] is an open-source project that provides a platform for policy authorization and evaluation. OPA is designed to help development teams design and enforce security, access and other types of policies in their applications and services.

OPA is based on the definition of policies in a declarative language called Rego. These policies can address many use cases, such as access control, data validation, network authorization, and are all implemented through an API exposed to the application, which can be queried to verify, among several functions, whether the user has permissions to access a resource.

Fig. 1 shows the architecture of the implemented solution and the way its components dialogue.

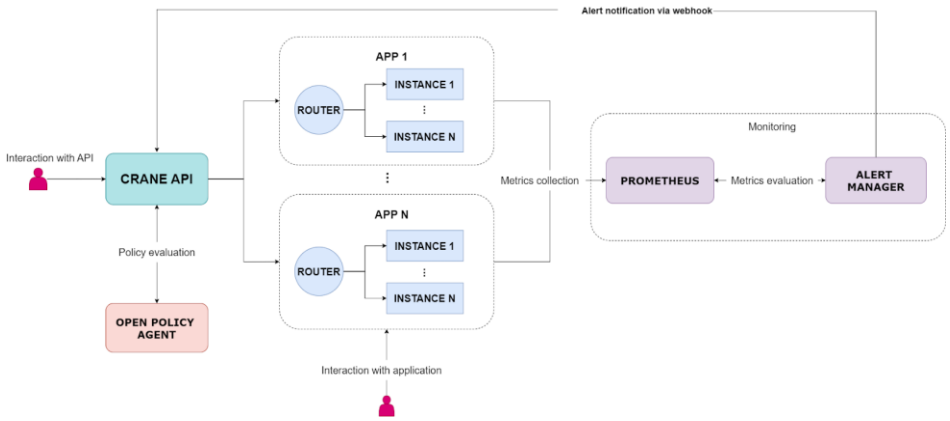


**Fig. 1.** CRANE components and their interaction.

CRANE API is a component written in Python that coordinates and manages several vital functions. It acts as the main link to the Python API, using the Python on Whales library to perform create, read, update and delete (CRUD) operations for containerized applications. In addition, it generates application instances, receives and manages alerts and manages user security.

The application instances (APP), referred to as Instance 1 to N, are interconnected within a virtual private network. This allows secure and isolated communication between the application instances. In turn, the application router communicates with Prometheus using a global network to provide access to the container's network metric.

Prometheus, acts as the monitoring system, responsible for reading and collecting metrics from the application instances. These metrics provide crucial information about the status and performance of the application, essential for automated decision making.

Alertmanager integrates with Prometheus to read alerts generated based on the collected metrics. Alertmanager processes and manages these alerts, and if certain conditions are met, triggers a webhook alert to CRANE.

Webhook alerts are HTTP requests triggered by Alertmanager when specific conditions are detected that require attention. This triggers a process in CRANE to query for possible actions and update the configuration as needed.

Open Policy Agent (OPA), acts as an authorization and policy control layer, which receives queries from CRANE and determines allowable actions according to defined policies. This mechanism ensures that any changes or automated responses are aligned with business rules and security requirements.

### 3.2- Starting CRANE, creating an application and managing alerts

This section presents scenarios of use of the proposed implementation.

The first scenario to be presented consists of starting Crane using the `uvicorn app:app -reload` command.

Within the CRANE API the following processes are executed: a) Authentication and registration: CRANE receives the request and extracts the user_id from the included token, integrating it to the application object, b) Storage: The application is registered in the database, c) Name management: A unique name is generated for the application, combining the name supplied by the user with an auto-incremental ID, to prevent name collisions in the host, d) Container configuration: A docker-compose. yml file is created with the default proxy configuration and the specifications provided by the user. This file is stored temporarily, e) Deployment of containers: Using the Python on Whales library, we proceed to build and deploy the environment with Docker Compose, f) Integration with Prometheus: Once the container is running, the port assigned to the router in the Prometheus network is obtained and the scrape is updated to allow reading metrics from the new application, g) Restarting the monitoring system: The monitoring stack is restarted to apply and refresh the configuration changes and, h) Temporary file management: Finally, if the REMOVE_TEMP_FILES constant is enabled, it is deleted, i.e., if the REMOVE_TEMP_FILES constant is enabled, it is deleted..

Regarding the creation of applications, the user has at his disposal all the configurations that Docker currently accepts, there is no limitation to create applications, as long as Python on Whales is kept updated with the latest Docker features. In addition, once any service is generated, use is made of the CRUD offered by the CRANE API to query, modify and delete the generated applications. In turn, users with administrator role have at their disposal the metrics, alerts and policies configurations to evolve the API and adapt it to their specific needs.

Fig. 2 contains a screenshot showing the instances generated with CRANE in the upper left part. In the lower left part is the terminal with CRANE running. It is also observed when a container is created and the endpoints that are receiving the request (CRANE status). In the right part, the body of the postman that receives the parameters for the creation of the instance is displayed.

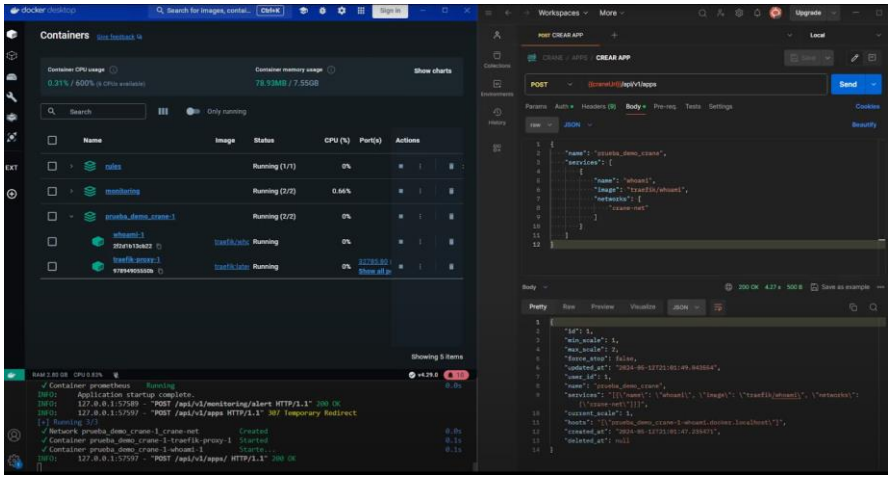

**Fig. 2** - Creation of instances in CRANE

Finally, in case of receiving alerts through the Alertmanager webhook, CRANE proceeds as follows: 1- determines the severity depending on whether the alert indicates FIRING or RESOLVED condition, 2- locates in the database the identifier of the affected service that generated the alert, 3- consults OPA sending alert type, severity and name of the affected service so that OPA evaluates the corresponding policies and returns the action to be executed (escalate, de-escalate, restart container) and 4 - executes the action determined by OPA.

Fig. 3 contains in the upper right part a simulation of high load with Artillery [16]. In the upper left part, the alert activated in red. In the lower right, the CRANE console receiving the HTTP request for new instance creation. And finally in the lower left, the new whoami [1] instance.

[1] Tiny Go webserver that prints OS information and HTTP request to output

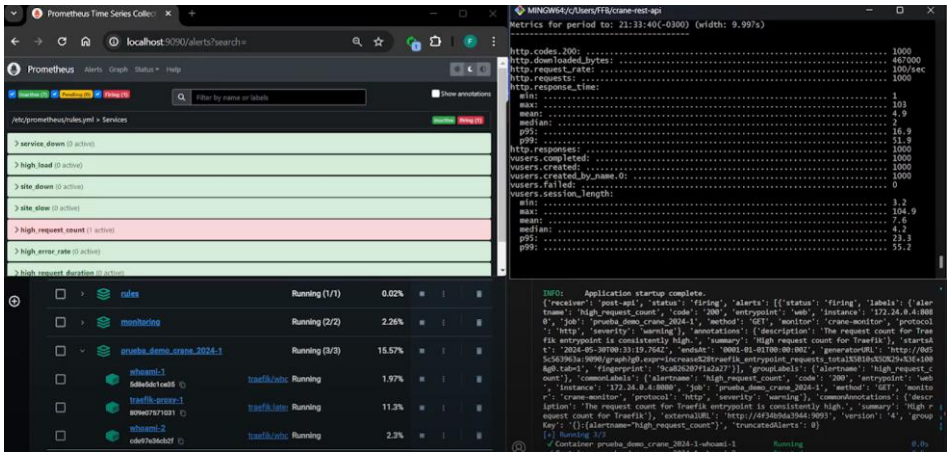

**Fig. 3** -Application scaling due to high demand

## 4- Conclusions, lessons learned and future works

The main objective of this work was to effectively implement the CRANE design proposal [6]. First, the design was technically validated and the technological alternatives for each component were analyzed.

The first prototype was developed in Python following Crane's structure and with it the communication of each part was validated. At the same time, Python FastAPI plugins were included, as well as the interaction of libraries to communicate with Docker.

Then, the integration with Traefik for traffic management, Prometheus for metrics monitoring, Alertmanager for alert management, and Open Policy Agent for the application of scaling and security policies was performed.

Finally, data models were integrated to apply an authentication and authorization layer over the designed API.

Beyond its technical utility, this solution is intended to have significant value in the educational environment. Students could use it to learn about service development and deployment practices, but also for the simulation of DevOps practices in local environments, a process that, as explained throughout this paper, is very complex and costly fundamentally if not analyzed in early time.

Thus, the question *"Are you prepared to develop and understand applications running in a distributed environment on a cloud infrastructure?"* can be answered in terms of using CRANE to learn about relevant aspects in a local environment.

For future work, we propose the creation of a Front-End using React. This Front-End will consume all implemented REST endpoints, from the creation and deployment of Docker services to metering, scaling policy definition and alert management. This will provide users with a complete interface to communicate with all of Crane's core functions.

The purpose of this Front-End is to make CRANE accessible and efficient for a wide range of users, from beginners to experts. It not only aims to make the technical process simpler, but also to foster understanding and learning about containerized service management in a local environment.

The alerting system can also be enhanced by using container metrics, such as memory and CPU usage, for its scaling policies and other parameters, as currently, metrics are taken directly from router traffic.

**Acknowledgements.** To Jose Felipe Arcidiácono who was the person who made the initial design of Crane.

## References

1. 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).
2. 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.
3. Kubernetes - https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/
4. Burns, B., Beda, J., Hightower, K., & Evenson, L. (2022). Kubernetes: up and running. " O'Reilly Media, Inc.".
5. Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. Ieee Software, 33(3), 94-100.
6. 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.
7. Python - https://www.python.org/doc/
8. Silva Pavon, J. M., Bellino, F., Bazán, P. A., Lliteras, A. B., Arcidiacono, J., & Rio, N. D. (2023). Despliegue de aplicaciones contenerizadas: un caso de implementación basado en Crane. In XXV Workshop de Investigadores en Ciencias de la Computación (Junín, 13 y 14 de abril de 2023)
9. Silva Pavón, J. M., Bellino, F., Bazán, P. A., Lliteras, A. B., & Rio, N. D. (2024). CRANE: simplificando el despliegue de aplicaciones contenerizadas en entornos locales. In XXIX Congreso Argentino de Ciencias de la Computación (CACIC) (Luján, 9 al 12 de octubre de 2023).
10. Docker - https://docs.docker.com/
11. Sommerlad, P. (2003, June). Reverse Proxy Patterns. In EuroPLoP (pp. 431-458).
12. Traefik - https://doc.traefik.io/traefik/
13. Prometheus Documentation - https://prometheus.io/docs/introduction/overview/
14. Alertmanager Documentation - https://prometheus.io/docs/alerting/latest/alertmanager/
15. Open Policy Agente Documentation - https://www.openpolicyagent.org/
16. Artillery.io https://www.artillery.io/