Optimización dinámica de microservicios: Implementación y evaluación de un escalador automático

Presentado por: Gastón Rial Saibene
Dirigido por: Miguel Angel Navarro Burgos

Máster en Ingeniería Matemática y Computación
10 de Julio de 2024

Contenido

Introducción

Motivación

La motivación de esta investigación radica en dos aspectos críticos del autoescalado en entornos de microservicios.

1. Garantizar el cumplimiento de los Objetivos de Nivel de Servicio (SLOs) mediante la asignación dinámica y eficiente de recursos.
2. Abordar la necesidad de optimizar el uso de recursos en un contexto económico desafiante, donde los costos de los servicios de infraestructura de TI son elevados.

Introducción

Planteamiento del trabajo

- Propuesta: Diseño e implementación de un escalador automático para orquestar tareas contenerizadas en Kubernetes, tanto en entornos locales como en la nube.
- Objetivos: Evaluar el rendimiento de este escalador, basado en algoritmos de optimización y predicción implementados en Julia, comparándolo con soluciones existentes, extrayendo conclusiones de trabajos previos sobre gestión de recursos en entornos de microservicios.

Contexto y Estado del Arte

Contextualización

- Microservicios
- Computación en la nube
- SLOs - Objetivos de Nivel de Servicio
- Algoritmos genéticos
- Series temporales
- Contenedores
- Kubernetes
- Autoescaladores de microservicios
- Entorno de programación Julia
- Pruebas de carga - k6

Contexto y Estado del Arte

Análisis del estado del arte

- El uso de reglas predefinidas y métricas simples para automatizar la asignación de recursos han demostrado ser limitados en su capacidad para adaptarse a la dinámica de los microservicios.
- La introducción de técnicas basadas en aprendizaje automático ha mejorado la precisión y eficiencia del autoscalado, permitiendo una predicción más precisa de la demanda de recursos y una asignación óptima en tiempo real.
- La complejidad computacional de los modelos de ML son algunos de los desafíos que enfrentan los desarrolladores y administradores de sistemas. Por ejemplo:
  - Trafico variable y fluctuante
  - Complejidad de la topología de los sistemas distribuidos
  - Ajuste en tiempo real de los recursos a bajo costo
  - Garantizar el cumplimiento de los SLOs

Identificación de Requisitos

A grandes razgos, podemos destacar los elementos de la implementación del autoescalador en el siguiente diagrama.

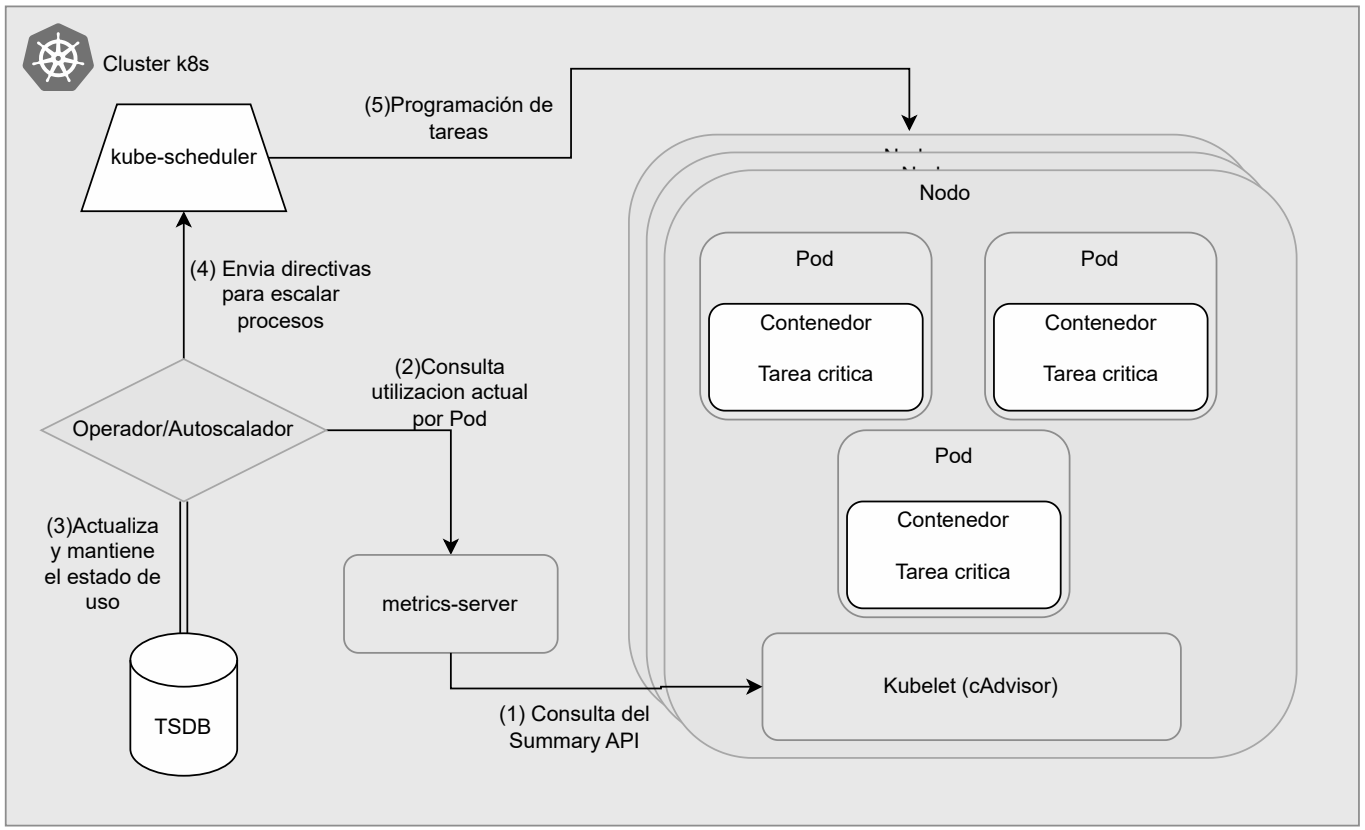


Figura: Descripción general de la implementación

Descripción del sistema

Las funciones claves del autoescalador son:

- **Recolección de datos:** Para cada microservicio, se recogen las métricas de uso de CPU, memoria y red a traves de API REST, y se almacenan en una base de datos de series temporales (TSDB).
- **Predicción:**Se utilizan regresiones lineales para predecir el valor futuro de una serie en función de sus observaciones pasadas.
- **Ajuste de recursos:** Se utiliza un algoritmo de optimización basados en algoritmos genéticos para ajustar el número de replicas de microservicios basándose en las predicciones de demanda.
- **Integración con Kubernetes - clúster -:** Un operador de k8s actualizará el número de replicas de cada microservicio.

Descripción del sistema (cont.)

La aplicación de pruebas a utilizar cuenta con dos microservicios, ‘web-a’ y ‘web-b’. El microservicio ‘web-a’ es ‘stateless’ o sin estado, mientras que el microservicio ‘web-b’ es ‘stateful’ o con estado, puesto que persiste y manteiene sincronizada la información relevante para la aplicación.

Desarrollo del trabajo

Modelo de optimización de recursos

El problema de optimización para un instante de tiempo dado se puede expresar como:

$$
\begin{cases}
\text{minimizar} & -\sum_i x_i \frac{1}{\text{RTT}_i} \\
\text{sujeto a} & \\
& \sum_i x_i \frac{1}{\text{RTT}_i} \leq \sum_i x_i \frac{1}{\text{RTT}_n}, \\
& \sum_i x_i \text{CPU}_i \leq \text{CPU}_{\text{max}}, \\
& \sum_i x_i \text{MEM}_i \leq \text{MEM}_{\text{max}}, \\
& \sum_i x_i \text{RPS}_i \geq \sum_i \text{RPS}_{\text{real}_i} \text{SLO}_i, \\
& 0 \leq \text{SLO}_i \leq 1, \\
& x_i \in \mathbb{N}^+, \quad \forall i \in \{1, \ldots, n\}.
\end{cases}
$$

Desarrollo del trabajo

Modelo de optimización de recursos

Cuadro: Descripción del Modelo de Optimización de Recursos

Término	**Descripción**
x_i	Variable de decisión que representa la cantidad de replicas asignadas al servicio i.
$\overline{\text{RTT}}_i$	Tiempo de ida y vuelta promedio (Round-Trip Time) para el servicio i.
CPU_i	Requerimiento de CPU del servicio i.
MEM_i	Requerimiento de memoria del servicio i.
CPU_{max}	Máximo límite de recursos de CPU disponibles.
MEM_{max}	Máximo límite de recursos de memoria disponibles.
CFT	Throughput (tasa de procesamiento) total considerando los requisitos de cada servicio.
CFT_{min}	Cota inferior de throughput deseado.
RPS_i	Solicitudes por segundo que cada unidad el servicio i puede tolerar antes de sobrecargarse.
$\text{RPS}_{\text{real}\,i}$	Solicitudes por segundo reales para cada unidad del servicio i.
SLO_i	Objetivo de Nivel de Servicio (Service Level Objective) para el servicio i.

Desarrollo del trabajo

Diseño del autoescalador

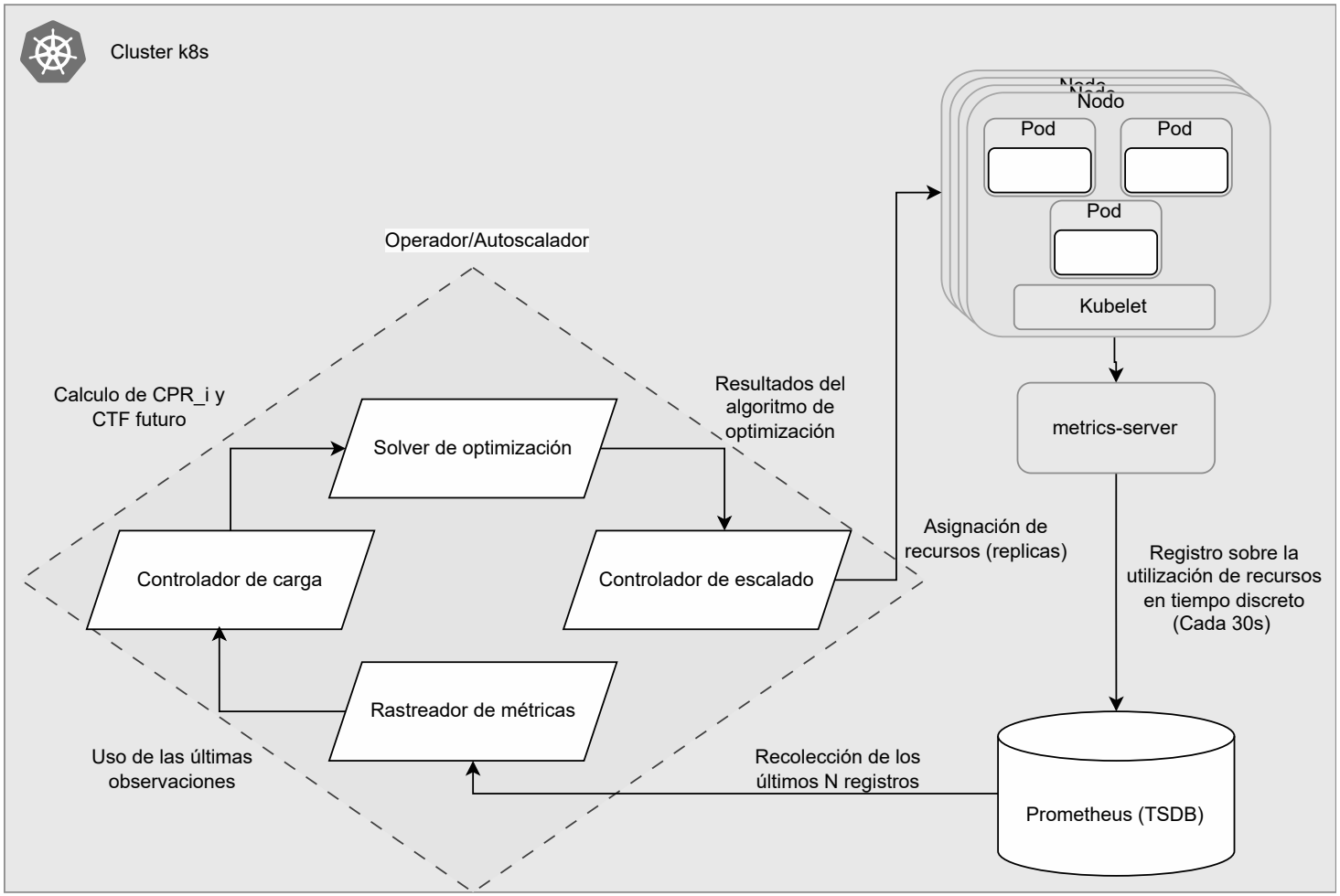


Figura: Arquitectura del autoescalador

Desarrollo del trabajo

Diseño del autoescalador

Cuadro: Valores de recursos y SLOs por unidad de cada microservicio

μ**servicio**	**CPU**	**Memoria**	**RPS**	**SLO**
web-a	200m	512MB	500	0.99
web-b	200m	512MB	300	0.999

Desarrollo del trabajo

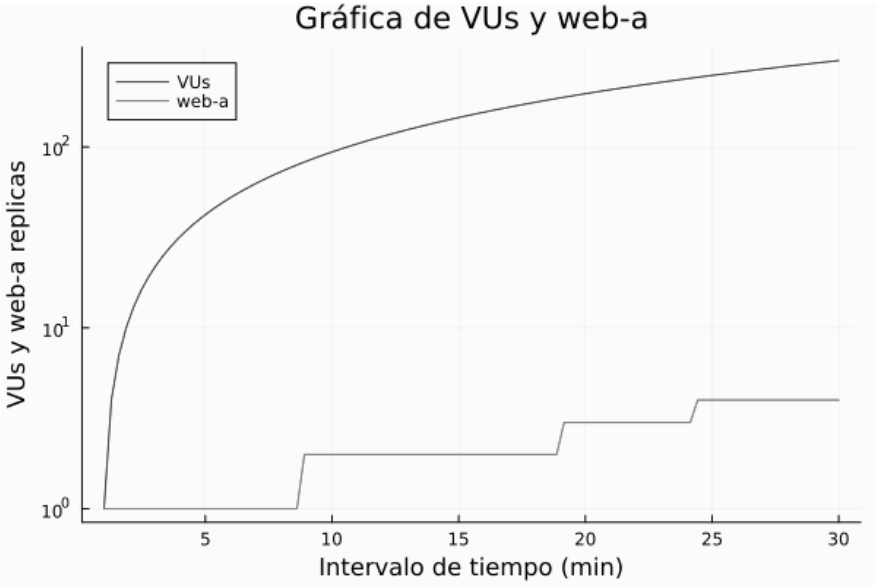
Implementación del autoescalador

Tanto la implementación del rastreador de métricas como la del controlador de escalado son implementaciones de referencia que cumplen con los requisitos del sistema.

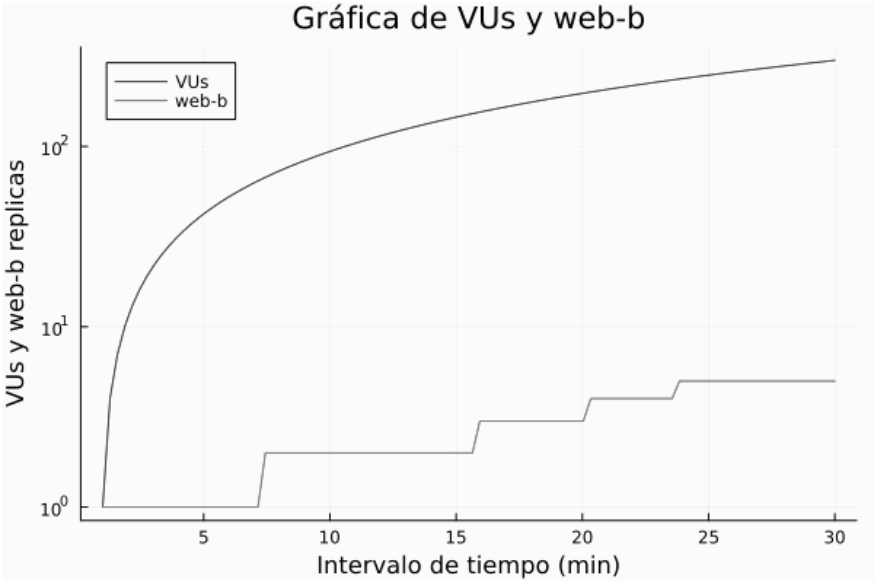
- **Controlador de Carga:** Utilizar regresiones lineales para predecir el uso futuro (CPR). Para ello, se utiliza el RTT promedio y RPS para los microservicios 'web-a' y 'web-b'. LinearRegression.jl implementa la regresión lineal utilizando el algoritmo de mínimos cuadrados.
- **Solver de Optimización** Empleando la biblioteca Evolutionary.jl, se implementa un algoritmo de optimización basado en programación genética (PG) para encontrar la mejor asignación de recursos.

Evaluación y Resultados

Pruebas de carga: Autoescalado de las réplicas



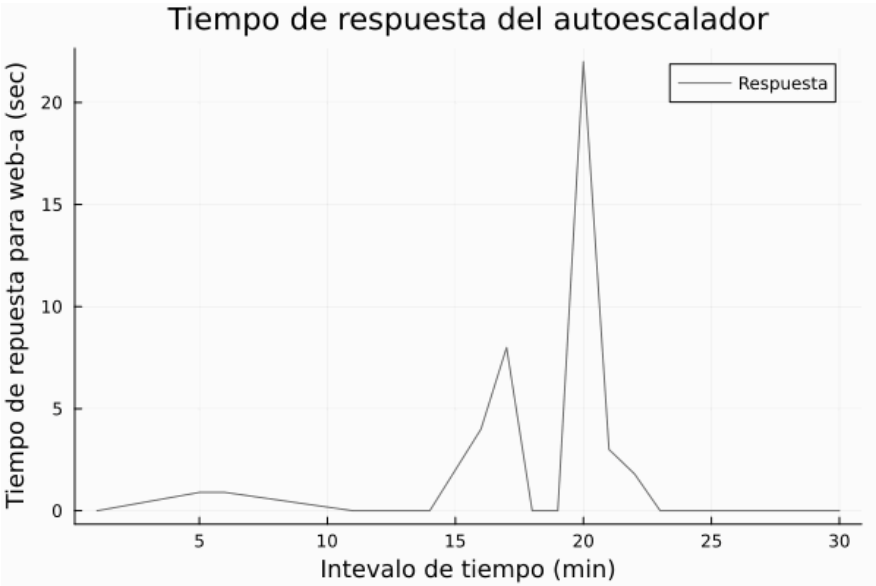
(a) Autoescalado de las réplicas del μservicio web-a



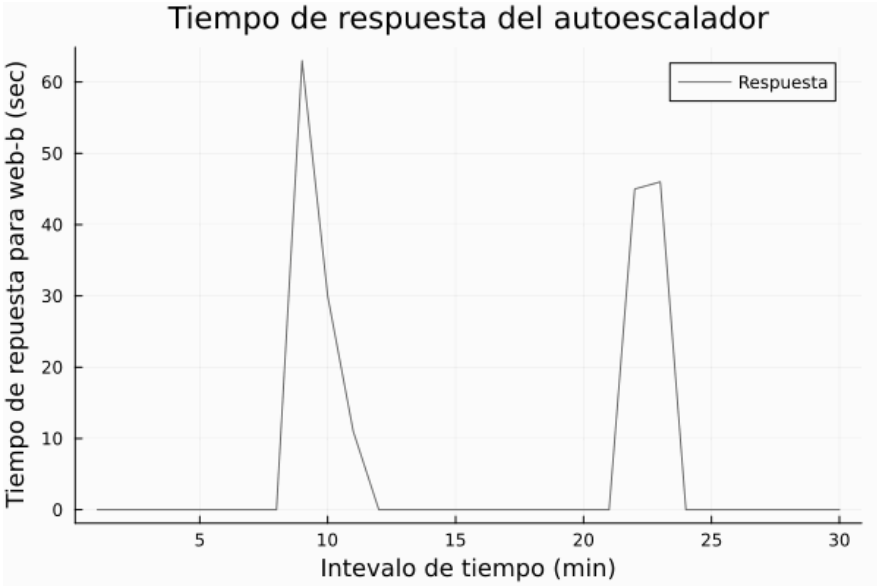
(b) Autoescalado de las réplicas del μservicio web-b

Evaluación y Resultados

Pruebas de carga: Respuesta del autoescalador



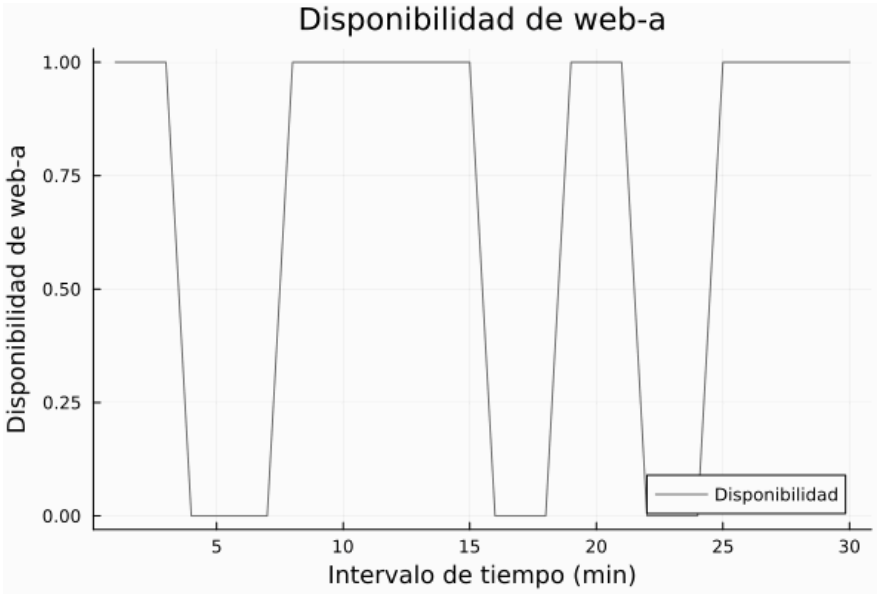
(c) Tiempo de respuesta promedio del autoescalador para el μservicio web-a



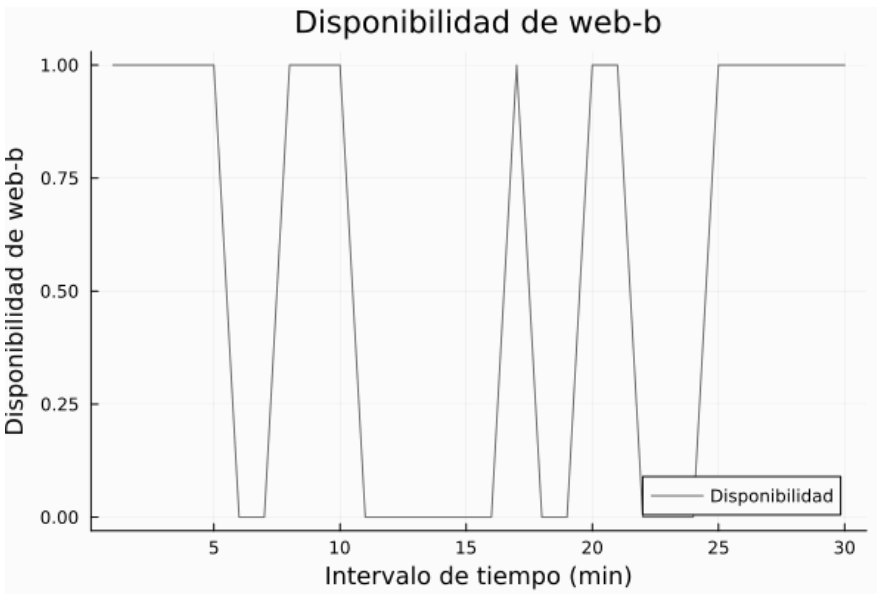
(d) Tiempo de respuesta promedio del autoescalador para el μservicio web-b

Evaluación y Resultados

Pruebas de carga: Disponibilidad de los microservicios



(e) Disponibilidad del μservicio web-a



(f) Disponibilidad del μservicio web-b

Evaluación y Resultados

Pruebas de carga: Cumplimiento de SLOs

Cuadro: Tasa de violación de SLO para los microservicios durante las pruebas de carga

μ**servicio**	**Tasa promedio**
web-a	29 %
web-b	20 %

Conclusiones

Resumen de las conclusiones del trabajo

Punto clave	**Descripción**
Diseño y evaluación del autoescalador	Se diseñó, implementó y evaluó un autoescalador en Julia para gestionar recursos en Kubernetes mediante microservicios, mejorando predicción y asignación de recursos.
Modelo de optimización	El modelo de optimización usa técnicas avanzadas de regresión lineal y algoritmos de predicción para prever la demanda futura y ajustar recursos, manteniendo los SLOs sin uso excesivo de recursos.

Conclusiones

Resumen de las conclusiones del trabajo

Punto clave	**Descripción**
Eficiencia de Julia	Julia se ha demostrado eficiente y adaptable para desarrollar componentes críticos del autoescalador, manejando cálculos numéricos intensivos y permitiendo despliegue escalable en Kubernetes.
Resultados de pruebas	El autoescalador mantiene un equilibrio óptimo entre uso de recursos y cumplimiento de SLOs, mostrando rendimiento competitivo comparado con soluciones existentes como Ursa.

Conclusiones

Resumen de las conclusiones del trabajo

Punto clave	**Descripción**
Limitaciones	A pesar del buen rendimiento, el autoescalador desarrollado no supera a todas las soluciones existentes en todos los aspectos. Cada solución tiene fortalezas y áreas específicas de aplicación.


unir
LA UNIVERSIDAD
EN INTERNET