## Proyecto URL Shortener

Acerca del proyecto:

Programa diseñado para acortar urls, almacenando en MongoDB, las urls pequeñas generadas.

Al introducir las urs pequeñas, se realiza una redirección hacia la url original, contando este evento como una visita.

Se construyo utilizando, las siguientes tecnologías y librerías:

* **NodeJS:**

Está orientado a eventos, puede manejar una gran cantidad de solicitudes simultáneas.

Gran rendimiento, en aplicaciones con alto tráfico de usuarios y eventos.

Es multiplataforma.

Extensa plataforma de paquetes (NPM), de los cuales se utilizaron:

* + - **Express:** Framework para facilitar la creación de aplicaciones web y APIs.
    - **Short-Unique-id:** Librería que genera UUID. Utilizando un diccionario BASE62, con dígitos de 0 al 9 y también con el alfabeto de A a Z (mayúsculas y minúsculas), es posible utilizar un diccionario personalizado. Los UUID generados tienen una longitud de 6 (que se puede parametrizar).

Usando las opciones por defecto, da un total de 56,800,235,584 posibles combinaciones. Con lo cual la probabilidad de generar un duplicado en 1.000.000 de intentos es de 0.00000002, o lo que es igual 1 en 50.000.000

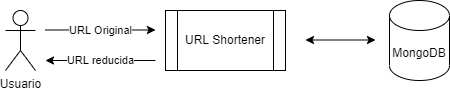
* + - **Moongose:** Librería que permite diseñar una estructura de datos (esquema), donde indicamos los documentos para una colección de datos en MongoDB. Permite aplicar validación, realizar consultas, conversión de datos, etc.
    - **Dotenv:** Modulo para el manejo de variables de entorno.
* **MongoDB:**

Se utilizo esta base de datos no relacional porque:

* Es altamente escalable (crecimiento horizontal), se pueden instalar mayor cantidad de nodos, para ampliar la capacidad.
* Garantizan un alto rendimiento, ya que están diseñadas para trabajar con modelos de datos concretos y patrones de acceso específicos.
* Ofrece mejores tiempos de respuesta en comparación con las bases relacionales.

Funciones incluidas en el proyecto:

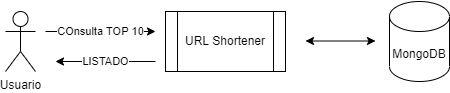
1. Al recibir una URL, emite una url pequeña.



1. Introduciendo la URL pequeña, redirecciona a la URL original (lo cual incrementa, internamente el número de visitas a dicha URL).



1. Se pueden eliminar URL pequeñas que estén almacenadas.
2. Existe una consulta, que permite conocer las 10 URL que recibieron más visitas.



# Lógica de Funcionamiento:

A continuación, se describe brevemente la lógica de la aplicación:

1. Luego de recibir una url, el primer paso que se realiza es la validación de la misma. Utilizando una función que controla que la url cumpla con una expresión regular.

NOTA: Esto se puede reemplazar por una función que valide que la url exista, pero dicha función agregaría tiempo de consulta y verificación. Se podría utilizar el paquete: url-exist  <https://www.npmjs.com/package/url-exist>

1. Una vez que se validó la url original, el siguiente paso es verificar que dicha url, no exista en nuestra base de datos. Si existe se devuelve el registro existente, y sino se procede al siguiente paso.
2. Si la url no existe, procedemos a crear un nuevo código UUID, utilizando la librería short-unique-id <https://shortunique.id/>

NOTA: Debido a que existe la posibilidad, de que se genere un UUID repetido, se realiza una validación con los registros en la Base de Datos, de los UUID generados previamente.

Como alternativa, se podría crear un servicio que cree grandes cantidades de UUIID y los administre, devolviendo siempre un UUID disponible. También se podría llevar un registro de que UUID se utilizaron y cuando se volvieron a disponibilizar, para su uso.

1. Lo que sigue luego de generar el código UUID, es proceder a guardar en la base de Datos MongoDB, nuestro objeto JSON, cuyos campos más importantes son la url original, y la url pequeña.

NOTA: El guardado del objeto JSON se realiza utilizando el paquete Moongose, que permite el modelado de objetos (esquemas) con conexión a MongoDB.

# Consumo local y pruebas de stress:

Se realizaron pruebas de stress utilizando las siguientes herramientas

* **LoadTest** <https://www.npmjs.com/package/loadtest>

Este paquete permite ejecutar Pruebas de cargas en métodos Http.

Instalación:

npm install -g loadtest

Ejecución del Load Test con el comando:

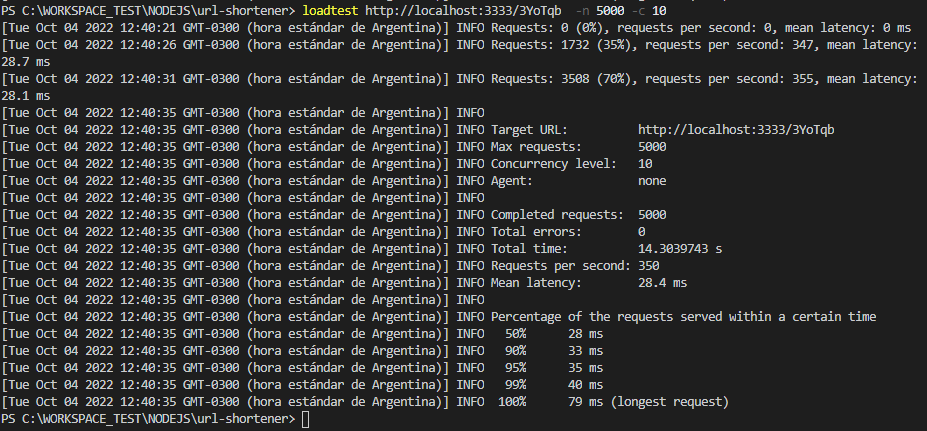
loadtest http://localhost:3333/3YoTqb -t 20 -n 5000 -c 10

-t timelimit

-n requests

-c concurrency

El comando anterior arrojo lo siguiente:



* **PM2** <https://pm2.keymetrics.io/docs/usage/quick-start/>

Es una herramienta de monitoreo (la cual se describe mas abajo en este documento)

Instalación:

npm install pm2@latest -g

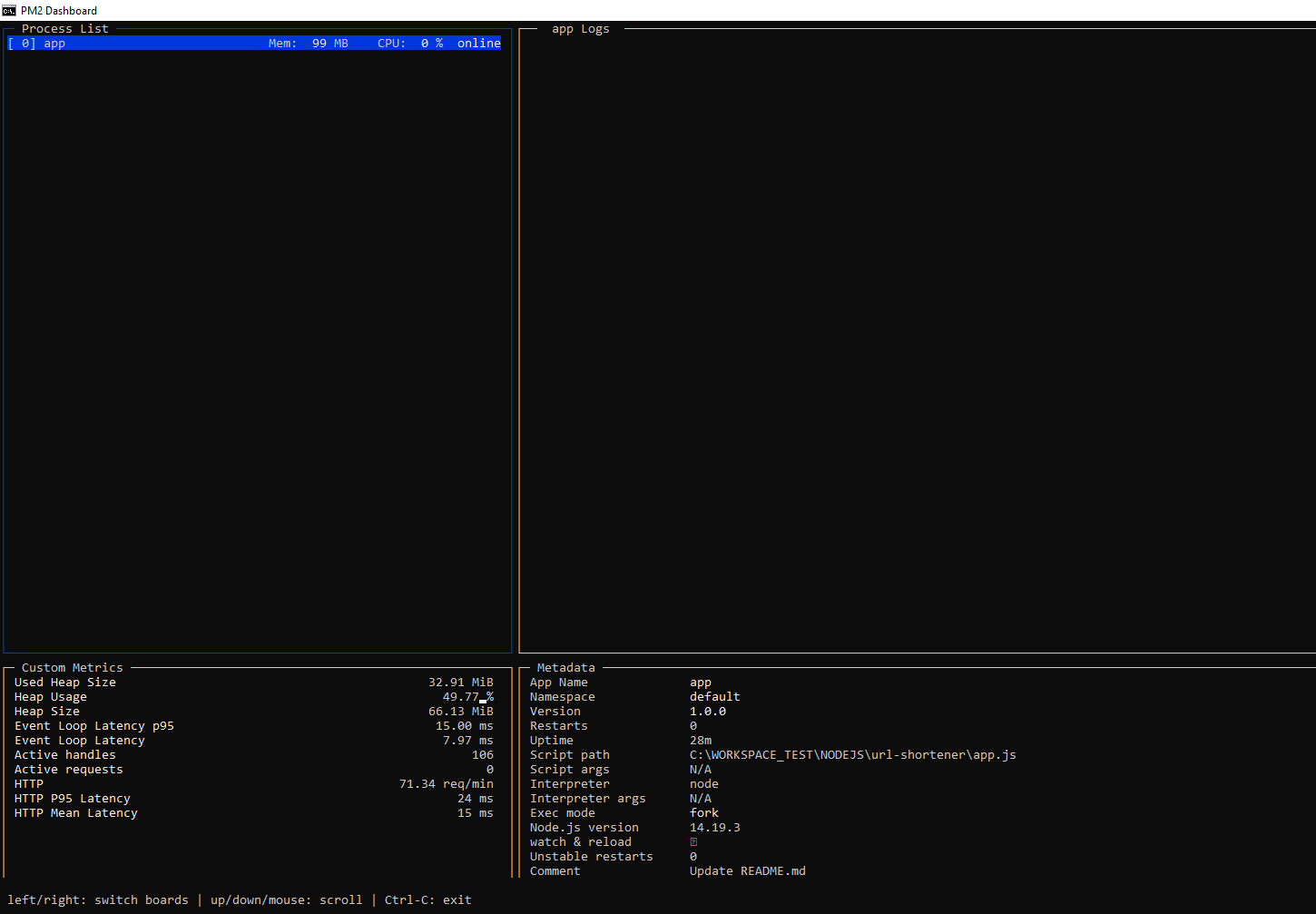
Para iniciar el monitor se ejecuta el siguiente comando:

pm2 start app.js

Luego se inicia el monitor:

pm2 monit

En el cual, al ejecutar el test de carga anterior vemos lo siguiente:



Para finalizar el monitoreo se utiliza:

pm2 stop app.js

* **Express status Monitor:** <https://github.com/RafalWilinski/express-status-monitor>

Es una solución en que permite monitorear en tiempo real, aplicaciones que utilizan NodeJS/Express.

Instalación:

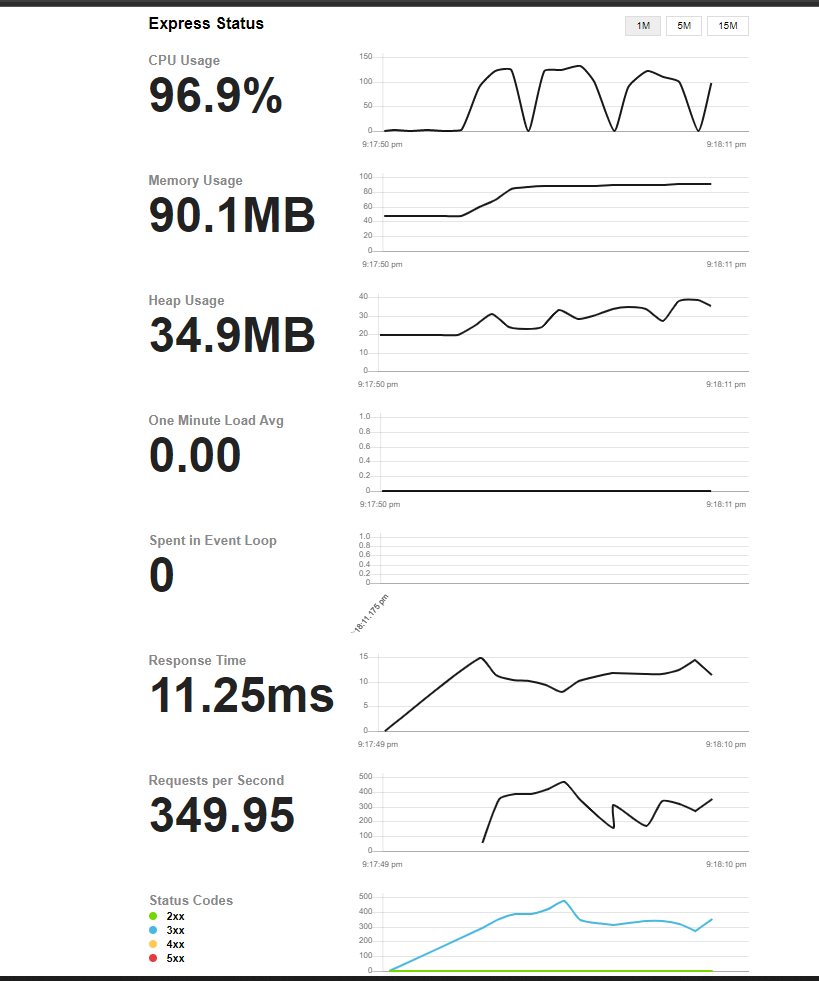
npm install express-status-monitor –save

Luego antes de otro middleware se agrega

app.use(require('express-status-monitor')());

Uso:

Se inicia la app y se va a la dirección localhost:3333/status



# Implementación a gran escala:

Teniendo en cuenta los siguientes requerimientos del desafío:

* **El 90% de todas las solicitudes puedan responderse en menos de 10 ms.**
* **Una solución que pueda escalar hasta, al menos, 50.000 peticiones por segundo**

Como se comentó previamente, se utiliza NodeJS, debido a su gran capacidad para el manejo de eventos, (gestión simultanea de solicitudes).

Pero continuando con el planteo, si consideramos que podemos tener una gran cantidad de usuarios conectados, realizando múltiples consultas de manera simultánea, y sumamos que queremos mantener bajos tiempos de respuesta. Es probable que una sola instancia de la aplicación no sea suficiente.

Por lo cual algunas sugerencias, a tener en cuenta son las siguientes:

Utilizar Cache: Esto permite almacenar consultas reiterativas a base de datos y evitar llamadas recurrentes. Se puede utilizar Varnish [(https://varnish-cache.org/docs/index.html)](https://varnish-cache.org/docs/index.html), que es conocido como un cache proxy HTTP inverso, el mismo se instala delante del servidor HTTP y se configura para almacenar en el cache del servidor una copia del recurso solicitado. Esto mejoraría la velocidad de respuesta, que está pensado para aplicaciones web de contenidos pesados o APIs altamente consumidas. Está orientado exclusivamente a HTTP.

*Es un software Libre y de código abierto, para distintas distribuciones de Linux (en Windows se puede instalar con Cygwin).*

Utilizar un Balanceador de Carga: Si se configura un balanceador de carga, que apunte a múltiples instancias de la aplicación, esto permitiría incrementar la cantidad de consultas, sin impactar negativamente en la performance. Dos ejemplos de balanceadores son: Nginx <https://www.nginx.com/> y HAProxy <http://www.haproxy.org/>.

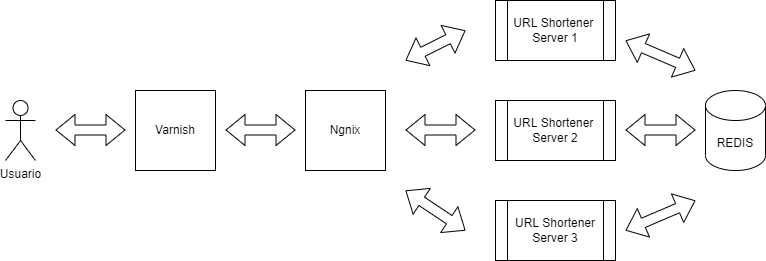
*Nginx: ​ Software libre y de código abierto, licenciado bajo la Licencia BSD simplificada. (existe una versión NGINX Plus).*

*HAProxy: Software gratuito y de código abierto (existe una versión Enterprise).*

Utilizar una base de datos en memoria: Si bien MongoDB, es una gran opción, comparando su performance contra bases relacionales (para este caso en particular de objetos). Es posible mejorar aún más el tiempo de respuesta utilizando una base en memoria como REDIS <https://redis.io/docs/getting-started/>

REDIS, es una base de datos en memoria, pero admite persistencia a almacenamiento persistente. Permite escalabilidad horizontal, tiene alta disponibilidad y también permite realizar cache de consultas, entre varias funcionalidades más.

Ejemplo de la arquitectura planteada:



Nota: Tanto Varnish como Ngnix/HAProxy, soportan balanceo de carga, la diferencia es que Ngnix/HAproxy son **reverse-proxy Load-Balancer** y Varnish es **reverse-proxy cache**.

Ambos trabajan bien juntos, Varnish mejora la performance en las respuestas (devolviendo rápidamente respuestas), mientras que Ngnix/HAproxy asegura un gran balanceo de cargas.

# Métricas:

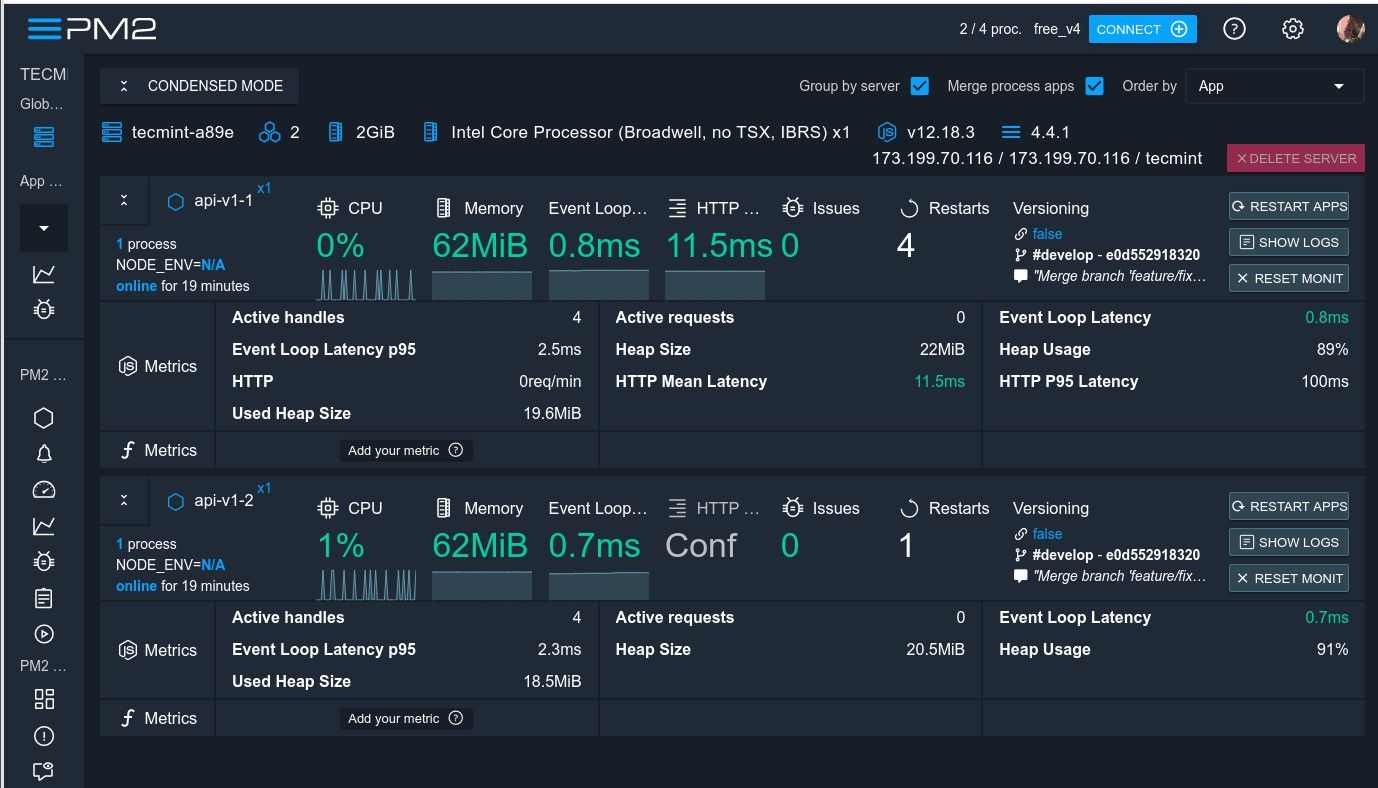
Para poder realizar un seguimiento y tener visibilidad de como esta funcionando nuestra aplicación, se puede utilizar alguno de los siguientes APM (Application Performance Management):

* PM2: es una herramienta de monitoreo, que valida que la aplicación este activa y operando. Es una solución simple para correr y monitorear cargas de trabajo, mediante interfaz web o por la línea de comando.

Se instala mediante NPM.

Es gratuito, mediante suscripción

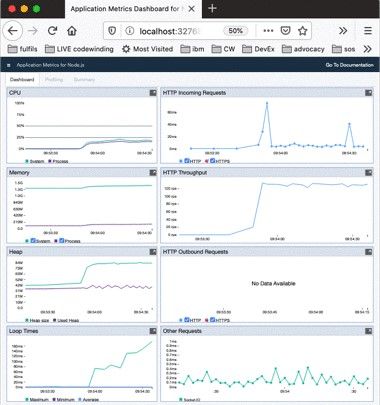
URL: <https://pm2.keymetrics.io/docs/usage/quick-start/>



* APP Metrics: Tiene una gran interfaz web, para monitorear el estado de nuestra app, puede ser utilizada con un middleware, para construir una aplicación de monitoreo.

Es una herramienta gratuita, open-source, hecha por IBM en .NET

URL: <https://github.com/RuntimeTools/appmetrics>



* ATATUS: Sin necesidad de tocar el código fuente de nuestra app, es una herramienta liviana que brinda monitoreo completo, en tiempo real de transacciones DB, request, fallos de API. También tiene Error Tracking, y detecta cuellos de botella en el rendimiento.

Es pago y tiene distintos planes, con diversas funcionalidades.

URL: <https://www.atatus.com/>



# URL del repositorio del proyecto:

<https://github.com/Bila82/url-shortener>