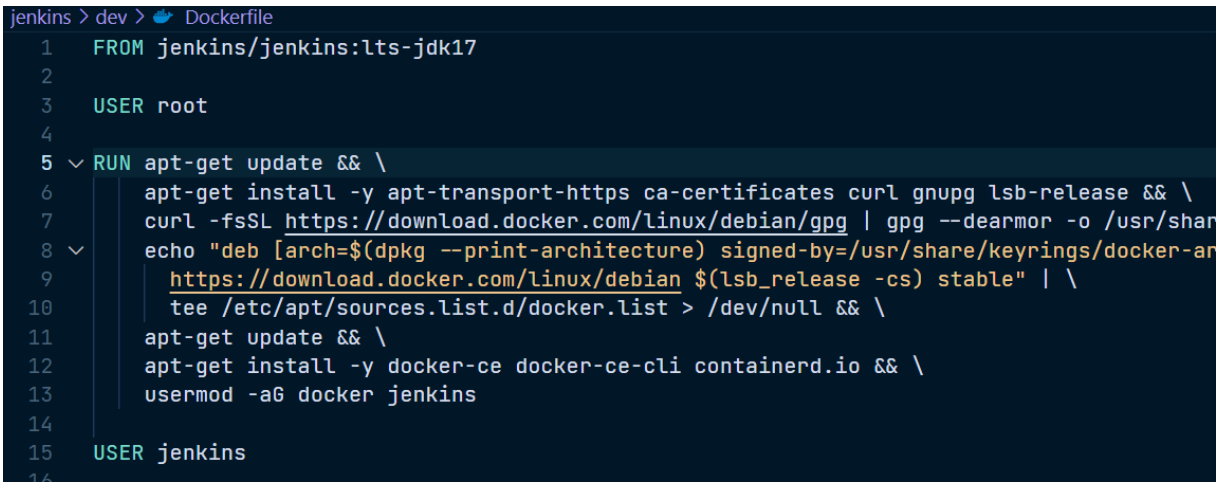
Taller 2: Pruebas y lanzamiento

Fork del repo: https://github.com/Br14nMat/ecommerce-microservice-backend-app

1.

Configuración Jenkins en contenedor:

Para ejecutar los pipelines del ambiente de desarrollo decidí tener un contenedor aislado que tenga Docker instalado para poder buildear las imágenes y poder subirlas a un Dockerhub. Este el Dockerfile con los respectivos permisos del docker.sock para poder ejecutar docker dentro del contenedor:



Esta en el repositorio: jenkins/dev/Dockerfile

Se corre con docker run -d --name jenkins-docker -v /var/run/docker.sock:/var/run/docker.sock -p 8080:8080 -p 50000:50000 jenkins-docker.

Configuración Jenkins local:

Así mismo, por dificultades al intentar correr los pipelines de stage y de prod en un contenedor, decidí correr Jenkins de forma local, descargando el jenkins.war y ejecutándolo con java -jar jenkins.war --httpPort=9090

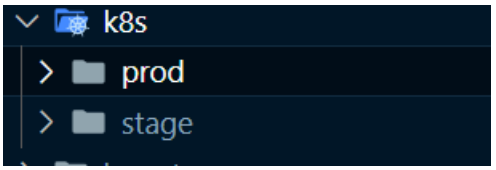
Para la versión de jenkins local que descargue se necesita java 17, entonces cambie las variables de entorno:

export JAVA_HOME="/c/Program Files/Java/jdk-17"

export PATH="$JAVA_HOME/bin:$PATH"

Configuración Kubernetes:

Para kubernetes definí para cada microservicio un manifiesto de deployment y de servicio basado en el compose.yml y el core.yml de la guía. Estos archivos están guardados en la carpeta k8s y están subdivididos en carpetas de stage y prod para separar el deploy de cada ambiente. Además, se ajustaron configuraciones como nombres de servicios, puertos y variables de entorno según el entorno correspondiente



2. Pipelines de ambiente de desarrollo

Para cada uno de los microservicios decidí crear un pipeline diferente y no hice despliegue. Esto para tener la facilidad de construir y subir cada servicio por separado sin necesidad de construir toda la aplicación.

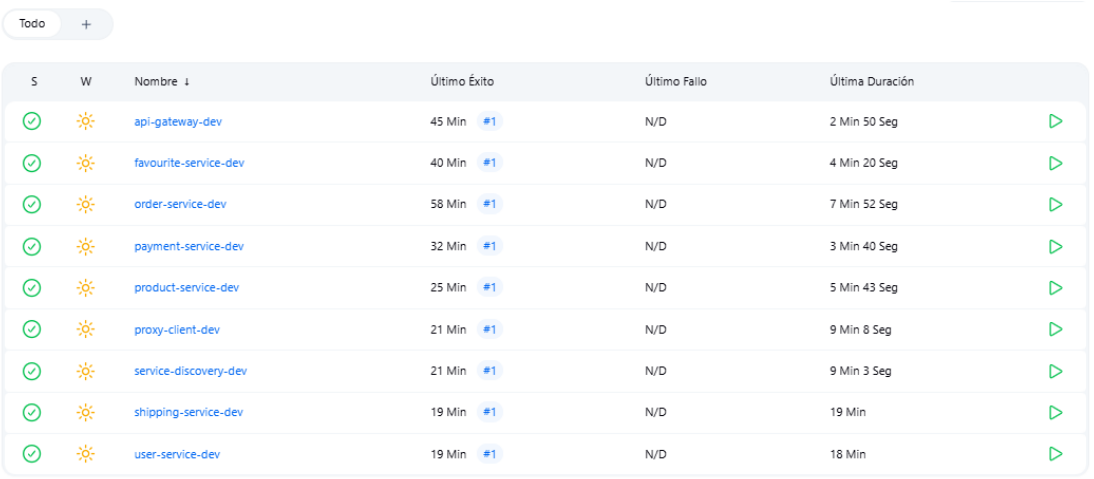
Los pipelines se encuentran en la carpeta jenkins/dev/* y básicamente todos siguen la misma estructura.

Se hace checkout del repositorio, se buildea el servicio con Maven y se ejecutan los test unitarios, se buildea la imagen y se sube a mi repositorio de docker hub con la etiqueta latest.

Ejemplo pipeline desarrollo order-service

```
pipeline{
    agent any
    tools {
        maven "MavenTool"
    }

    stages {
        stage("Checkout git") {
            steps{
                checkout scmGit(branches: [[name: '*/master']], extensions: [], userRemoteConfigs: [[url: '
            }
        }
        stage("Build Order Service") {
            steps {
                dir('order-service') {
                    sh 'mvn clean install'
                }
            }
        }
        stage("Build and Push Docker Image") {
            steps {
                script {
                    withDockerRegistry(
                        credentialsId: 'dockerhub-credentials'
                    ) {
                        sh 'docker build -t br14nmat/order-service:latest -f order-service/Dockerfile .'
                        sh 'docker push br14nmat/order-service:latest'
                    }
                }
            }
        }
```

Ejecución todos los pipelines de desarrollo:

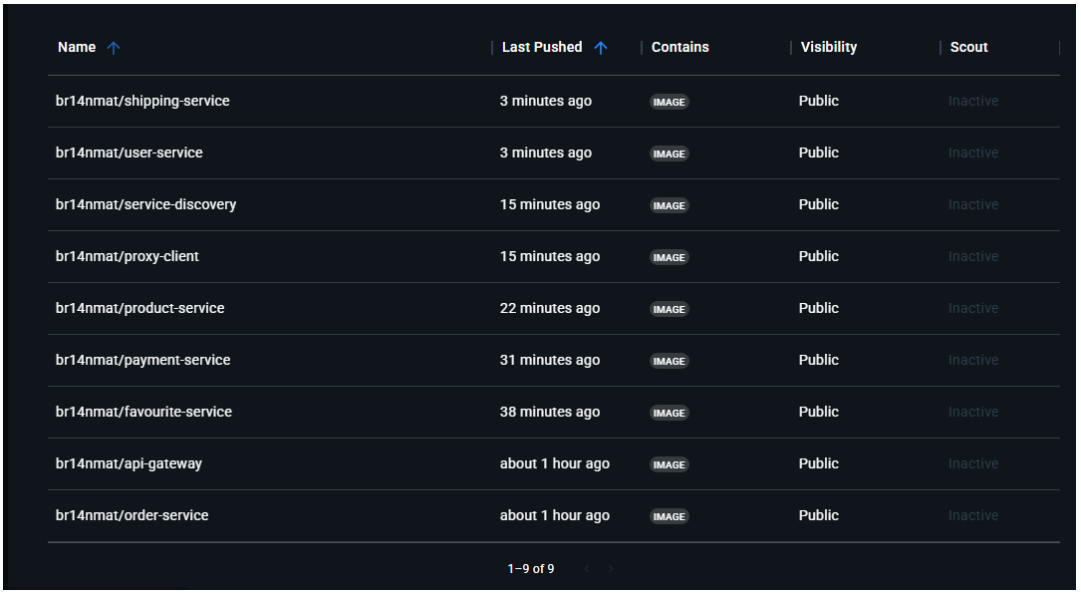


Todo +

S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
✓	☀	api-gateway-dev	45 Min #1	N/D	2 Min 50 Seg
✓	☀	favourite-service-dev	40 Min #1	N/D	4 Min 20 Seg
✓	☀	order-service-dev	58 Min #1	N/D	7 Min 52 Seg
✓	☀	payment-service-dev	32 Min #1	N/D	3 Min 40 Seg
✓	☀	product-service-dev	25 Min #1	N/D	5 Min 43 Seg
✓	☀	proxy-client-dev	21 Min #1	N/D	9 Min 8 Seg
✓	☀	service-discovery-dev	21 Min #1	N/D	9 Min 3 Seg
✓	☀	shipping-service-dev	19 Min #1	N/D	19 Min
✓	☀	user-service-dev	19 Min #1	N/D	18 Min

Docker hub con las imágenes:

Las siguientes imágenes son utilizadas por los manifiesto de kubernetes para hacer los despliegues en stage y prod.



Name ↑	Last Pushed ↑	Contains	Visibility	Scout
br14nmat/shipping-service	3 minutes ago	IMAGE	Public	Inactive
br14nmat/user-service	3 minutes ago	IMAGE	Public	Inactive
br14nmat/service-discovery	15 minutes ago	IMAGE	Public	Inactive
br14nmat/proxy-client	15 minutes ago	IMAGE	Public	Inactive
br14nmat/product-service	22 minutes ago	IMAGE	Public	Inactive
br14nmat/payment-service	31 minutes ago	IMAGE	Public	Inactive
br14nmat/favourite-service	38 minutes ago	IMAGE	Public	Inactive
br14nmat/api-gateway	about 1 hour ago	IMAGE	Public	Inactive
br14nmat/order-service	about 1 hour ago	IMAGE	Public	Inactive

1–9 of 9

3. Tipos de pruebas

Para algunos de los microservicios, debe definí pruebas unitarias, integración, E2E y de rendimiento con locust.

- Pruebas unitarias para los microservicios de product-service y payment-service

- Pruebas de integración para los microservicios product-service y payment-service. En estas pruebas, se levanta el contexto completo de Spring Boot con datos reales definidos para el entorno de pruebas. Por un lado, se prueba el microservicio de productos de forma aislada y por

otro lado, se verifica la integración entre payment-service y order-service.

- Pruebas e2e para los microservicios de user-service, product-service y order-service realizados con Newman utilizando los archivos json que exporta Postman.

- Pruebas de rendimiento y estrés utilizando Locust para los microservicios de user-service, product-service y order-service. Los resultados se guardan en archivos html en locust/locust_output/*/locust_report.html

4. Pipeline ambiente de stage (jenkins/stage/Jenkinsfile)

Para este ambiente se definieron los siguientes pasos:

1. Checkout git
2. Correr las pruebas unitarias y de integración
3. Login con azure
4. Obtener las credenciales del aks y configurar el kubelogin
5. Deploy de los core services
6. Deploy de los microservicios restantes
7. Hacer port forward a los servicios: product, user y order
8. Hacer pruebas e2e con Newman para los servicios anteriores

Resultados pruebas unitarias y de integración:

```
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.47 s - in com.selimhorri.app.unit.service.ProductServiceTest
2025-05-29 18:13:39.795  INFO [PRODUCT-SERVICE,,] 11972 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for
persistence unit 'default'
2025-05-29 18:13:39.796  INFO [PRODUCT-SERVICE,,] 11972 --- [ionShutdownHook] .SchemaDropperImpl$DelayedDropActionImpl : HHH000477: Starting delayed evictData
of schema as part of SessionFactory shut-down'
2025-05-29 18:13:39.796 DEBUG [PRODUCT-SERVICE,,] 11972 --- [ionShutdownHook] org.hibernate.SQL                        : drop table if exists categories
CASCADE
Hibernate: drop table if exists categories CASCADE
2025-05-29 18:13:39.812 DEBUG [PRODUCT-SERVICE,,] 11972 --- [ionShutdownHook] org.hibernate.SQL                        : drop table if exists products CASCADE
Hibernate: drop table if exists products CASCADE
2025-05-29 18:13:39.999  INFO [PRODUCT-SERVICE,,] 11972 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2025-05-29 18:13:40.010  INFO [PRODUCT-SERVICE,,] 11972 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
2025-05-29 18:13:40.017 DEBUG [PRODUCT-SERVICE,,] 11972 --- [ionShutdownHook] o.s.w.c.s.GenericWebApplicationContext   : Closing
org.springframework.web.context.support.GenericWebApplicationContext@587e0acf, started on Thu May 29 18:13:32 COT 2025
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 36, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:32 min
[INFO] Finished at: 2025-05-29T18:13:40-05:00
[INFO] ------------------------------------------------------------------------
[Pipeline] }
[Pipeline] // dir
[Pipeline] dir
```

```
2025-05-29 18:15:12.018  INFO [PAYMENT-SERVICE,,] 18832 --- [ionShutdownHook] com.netflix.discovery.DiscoveryClient    : Completed shut down of DiscoveryClient
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:25 min
[INFO] Finished at: 2025-05-29T18:15:12-05:00
[INFO] ------------------------------------------------------------------------
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // withEnv
```

Login azure y kubelogin:

```
[Pipeline] sh
+ az account set --subscription ****
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Kubelogin step)
[Pipeline] tool
[Pipeline] envVarsForTool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ az aks get-credentials --resource-group taller2 --name taller2brian --overwrite-existing
WARNING: Merged "taller2brian" as current context in C:\Users\Windows\.kube\config
+ kubelogin convert-kubeconfig -l azurecli
[Pipeline] }
[Pipeline] // withEnv
```

Deploy core services:

```
+ kubectl apply -n stage -f k8s/stage/zipkin-deployment.yaml
deployment.apps/zipkin unchanged
service/zipkin unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=zipkin --timeout=300s
pod/zipkin-86479766fb-kvb27 condition met
+ kubectl apply -n stage -f k8s/stage/cloud-config-container-deployment.yaml
deployment.apps/cloud-config-container unchanged
service/cloud-config-container unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=cloud-config-container --timeout=300s
pod/cloud-config-container-5fdfd67849-5ttvd condition met
+ kubectl apply -n stage -f k8s/stage/service-discovery-container-deployment.yaml
deployment.apps/service-discovery-container unchanged
service/service-discovery-container unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=service-discovery-container --timeout=300s
pod/service-discovery-container-8465654fd7-2jhqt condition met
+ kubectl apply -n stage -f k8s/stage/api-gateway-container-deployment.yaml
deployment.apps/api-gateway-container unchanged
service/api-gateway-container unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=api-gateway-container --timeout=300s
pod/api-gateway-container-579dd9d9df-fkk8c condition met
[Pipeline] }
```

Deploy remaining services:

```
+ kubectl apply -n stage -f k8s/stage/product-service-container-deployment.yaml
deployment.apps/product-service-container unchanged
service/product-service-container unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=product-service-container --timeout=300s
pod/product-service-container-78c456d65d-ltxb2 condition met
+ kubectl apply -n stage -f k8s/stage/payment-service-container-deployment.yaml
deployment.apps/payment-service-container unchanged
service/payment-service-container unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=payment-service-container --timeout=300s
pod/payment-service-container-7b67b5775d-hpnrb condition met
+ kubectl apply -n stage -f k8s/stage/order-service-container-deployment.yaml
deployment.apps/order-service-container unchanged
service/order-service-container unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=order-service-container --timeout=300s
pod/order-service-container-6cf5b5c5-pvk5r condition met
+ kubectl apply -n stage -f k8s/stage/user-service-container-deployment.yaml
deployment.apps/user-service-container unchanged
service/shipping-service-container unchanged
service/user-service-container unchanged
+ kubectl wait -n stage --for=condition=ready pod -l io.kompose.service=user-service-container --timeout=300s
pod/user-service-container-5bd87446dd-szwjt condition met
```

Port forward:

```
+ kubectl port-forward service/product-service-container 8500:8500 -n stage
+ echo 11132
+ kubectl port-forward service/order-service-container 8300:8300 -n stage
+ echo 11133
+ kubectl port-forward service/user-service-container 8700:8700 -n stage
+ sleep 10
Forwarding from 127.0.0.1:8700 -> 8700
Forwarding from [::1]:8700 -> 8700
Forwarding from 127.0.0.1:8300 -> 8300
Forwarding from [::1]:8300 -> 8300
Forwarding from 127.0.0.1:8500 -> 8500
Forwarding from [::1]:8500 -> 8500
[Pipeline] }
```

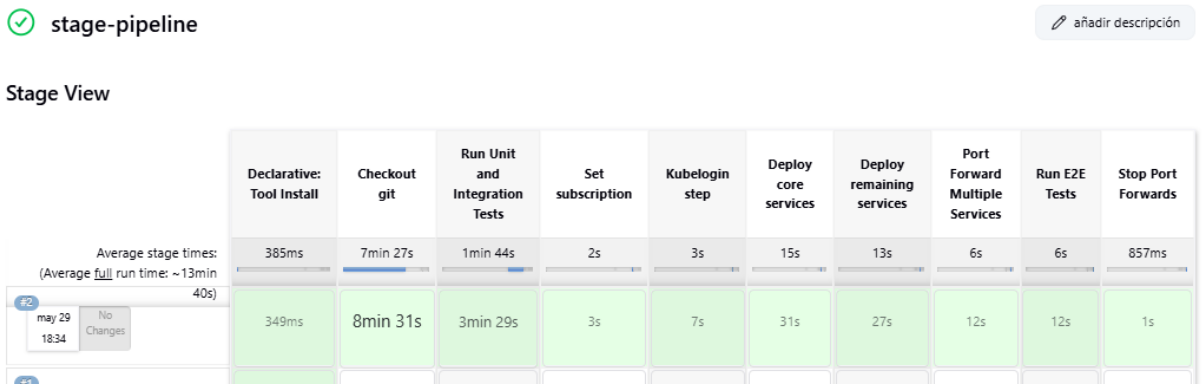
Pruebas e2e con Newman:

```
e2e-order

â†’ get all orders
  GET http://localhost:8300/order-service/api/orders [200 OK, 188B, 818ms]

â†’ save cart
  POST http://localhost:8300/order-service/api/carts [200 OK, 345B, 141ms]
```

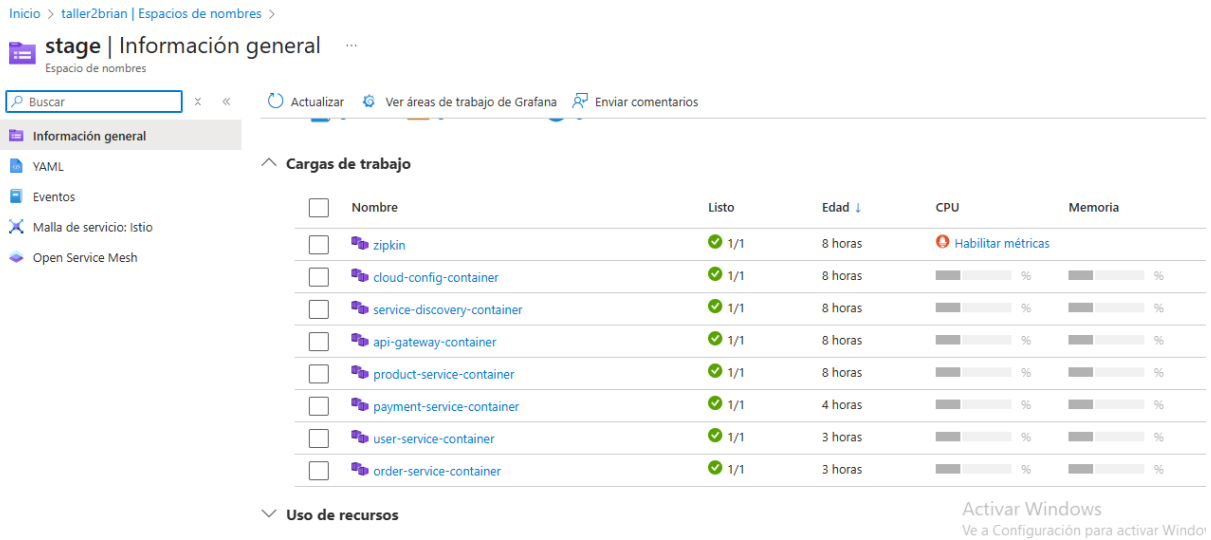
```
e2e

â†’ get all
  GET http://localhost:8500/product-service/api/products [200 OK, 33.43kB, 1249ms]

â†’ save
  POST http://localhost:8500/product-service/api/products [200 OK, 417B, 212ms]

â†’ update
  PUT http://localhost:8500/product-service/api/products [200 OK, 417B, 100ms]
```

```
e2e-user

â†³ get all users
  GET http://localhost:8700/user-service/api/users [200 OK, 188B, 406ms]

â†³ get all address
  GET http://localhost:8700/user-service/api/address [200 OK, 188B, 107ms]

â†³ get all credentials
  GET http://localhost:8700/user-service/api/credentials [200 OK, 188B, 117ms]

â†³ get all tokens
  GET http://localhost:8700/user-service/api/verificationTokens [200 OK, 188B, 124ms]
```

Pipeline con todos sus pasos completados:



Ambiente en el AKS con los servicios desplegados:



4. Pipeline ambiente de prod (jenkins/prod/Jenkinsfile)

Para este ambiente se definieron los siguientes pasos:

1. Checkout git
2. Correr las pruebas unitarias y de integración
3. Login con azure
4. Obtener las credenciales del aks y configurar el kubelogin
5. Deploy de los core services

6. Deploy de los microservicios restantes
7. Hacer port forward a los servicios: product, user y order
8. Hacer pruebas e2e con Newman para los servicios anteriores
9. Hacer pruebas de rendimiento y estrés con locust a los servicios anteriores. Guardar los resultados en archivos html en la ruta locust/locust_output/*/locust_report.html
10. Generar el release notes con la fecha, los commits y la versión. Estos se guardan en la carpeta release_notes

Los resultados son básicamente los mismos del pipeline anterior agregando las pruebas de rendimiento/estrés y los reléase notes

Pruebas de rendimiento/estrés: (estos son los resultados arrojados por consola, pero el análisis posterior se hace con los resultados y las graficas de los reportes en html)

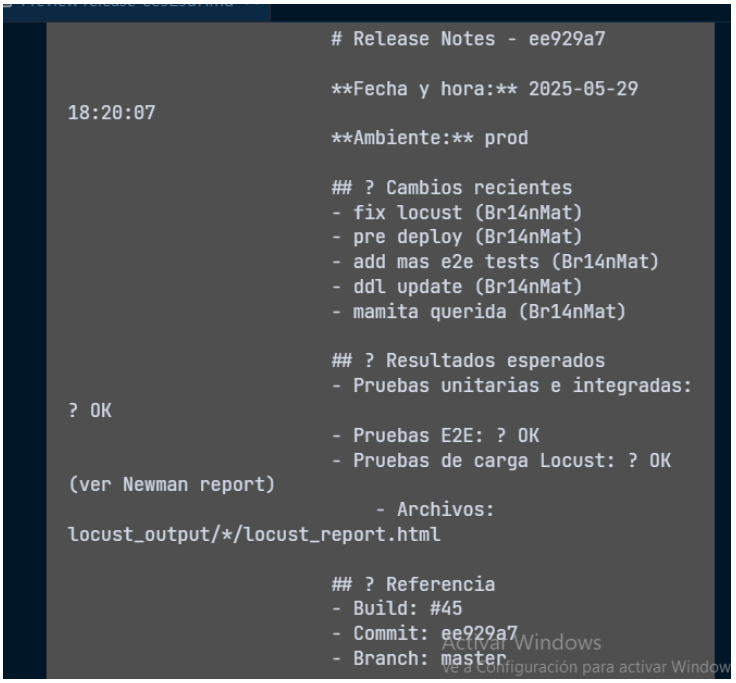
```
Type     Name                                                                  # reqs      # fails |    Avg     Min     Max    Med |   req/s
failures/s
--------|----------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|--------
-
GET      /product-service/api/products                                            78    0(0.00%) |   1413     271    6532    620 |    1.35
0.00
POST     /product-service/api/products                                            43    0(0.00%) |   1413      97    6403    320 |    0.74
0.00
PUT      /product-service/api/products/1                                          42    0(0.00%) |   1695     104    6453    800 |    0.72
0.00
--------|----------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|--------
-
         Aggregated                                                              163    0(0.00%) |   1486      97    6532    600 |    2.81
0.00

Response time percentiles (approximated)
Type     Name                                                                      50%    66%    75%    80%    90%    95%    98%    99%  99.9%
99.99%   100% # reqs
--------|----------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|--
--|------|------
GET      /product-service/api/products                                             620    880   1800   2100   4100   6000   6500   6500   6500
6500   6500     78
POST     /product-service/api/products                                             320   1000   3400   3600   4000   4300   6400   6400   6400
6400   6400     43
PUT      /product-service/api/products/1                                           810   2100   3200   3600   4100   4800   6500   6500   6500
6500   6500     42
--------|----------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|--
```



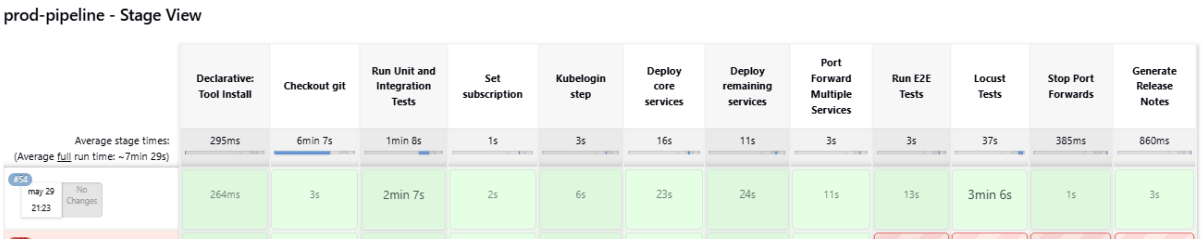
```
         Aggregated                                                             272    0(0.00%) |    174      91    1015    120 |    4.66
0.00

Response time percentiles (approximated)
Type     Name                                                                      50%    66%    75%    80%    90%    95%    98%    99%  99.9%
99.99%   100% # reqs
--------|----------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|----
--|------|------
GET      /user-service/api/address                                                 120    130    150    180    310    440    570    680    680
680    680     77
GET      /user-service/api/credentials                                             110    120    140    190    380    600    760    970    970
970    970     54
GET      /user-service/api/users                                                   120    170    200    210    390    420    510    690    690
690    690     76
GET      /user-service/api/verificationTokens                                      110    130    150    220    410    600    750   1000   1000
1000   1000     65
--------|----------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|----
--|------|------
         Aggregated                                                                120    130    170    200    360    450    680    760   1000
1000   1000    272
```

```
Type     Name                                                                          # reqs      # fails |    Avg     Min     Max    Med |   req/s
failures/s
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|---------
-
POST     /order-service/api/carts                                                        83     0(0.00%) |    157      96     503    120 |    1.10
0.00
GET      /order-service/api/orders                                                      170     0(0.00%) |    145      90     655    110 |    3.10
0.00
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|---------
-
         Aggregated                                                                     253     0(0.00%) |    149      90     655    110 |    4.20
0.00

Type     Name                                                                          # reqs      # fails |    Avg     Min     Max    Med |   req/s
failures/s
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|---------
-
POST     /order-service/api/carts                                                        85     0(0.00%) |    156      96     503    120 |    1.00
0.00
GET      /order-service/api/orders                                                      177     0(0.00%) |    143      90     655    110 |    3.80
0.00
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|---------
-
         Aggregated                                                                     262     0(0.00%) |    148      90     655    110 |    4.80
0.00
```

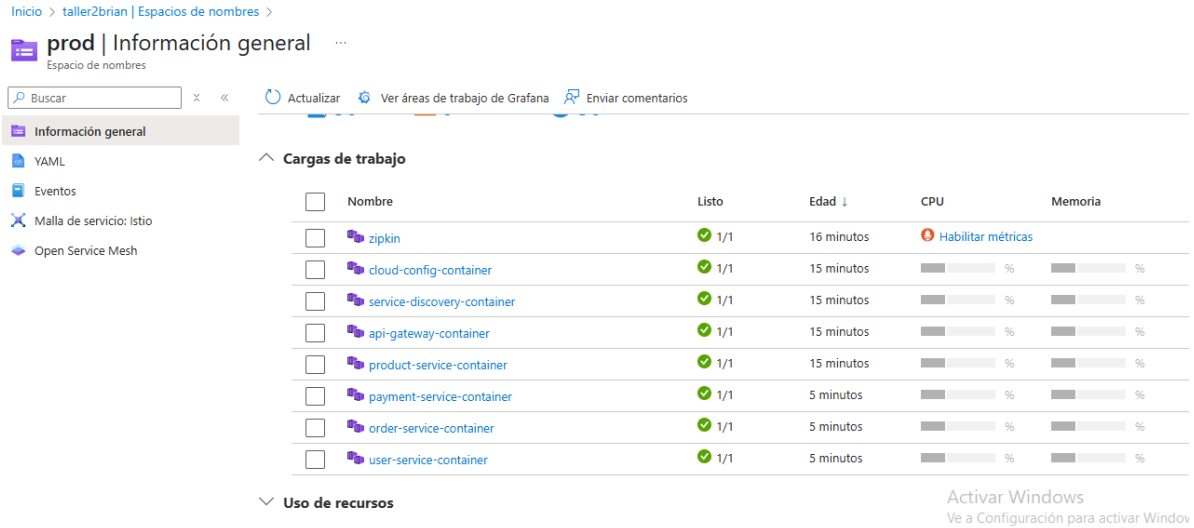
Release notes: carpeta release_notes



Pipeline con todos sus pasos completados:

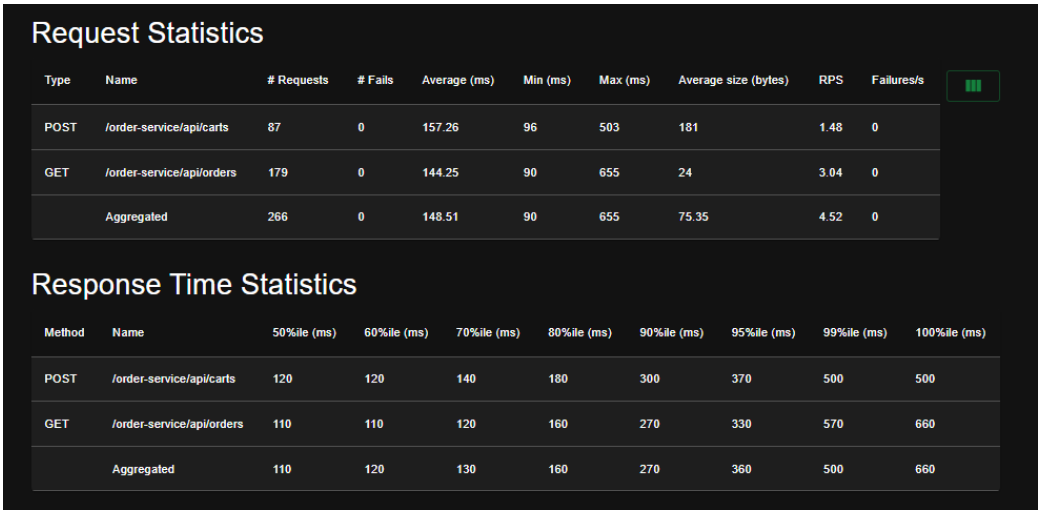


Ambiente prod en el AKS con los servicios desplegados:



5. Análisis pruebas de estrés y rendimiento:

order-service

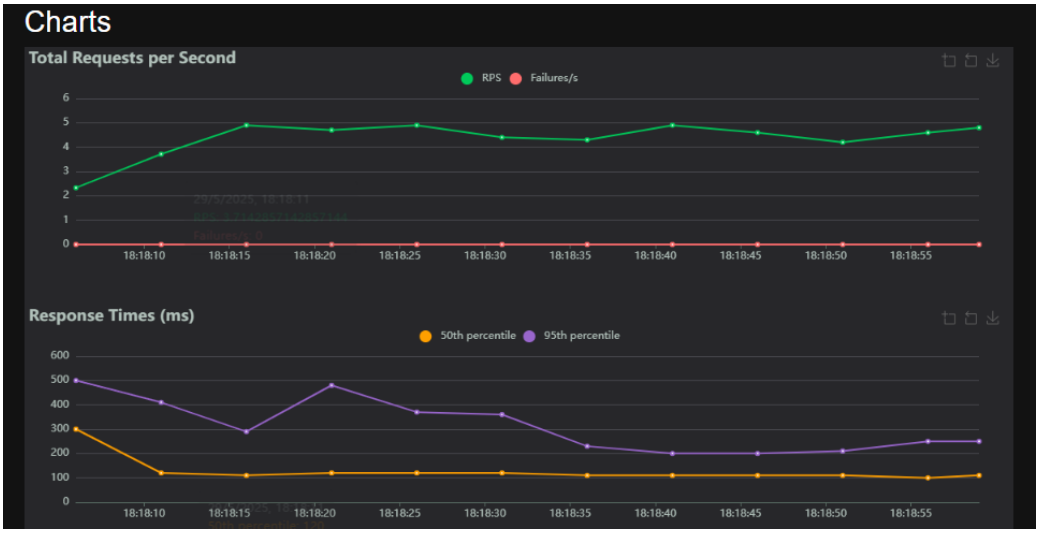


Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/order-service/api/carts	87	0	157.26	96	503	181	1.48	0
GET	/order-service/api/orders	179	0	144.25	90	655	24	3.04	0
	Aggregated	266	0	148.51	90	655	75.35	4.52	0

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/order-service/api/carts	120	120	140	180	300	370	500	500
GET	/order-service/api/orders	110	110	120	160	270	330	570	660
	Aggregated	110	120	130	160	270	360	500	660



Las pruebas de carga realizadas con Locust durante 58 segundos contra el servicio de órdenes (http://localhost:8300/order-service) muestran un rendimiento general estable con un throughput de 4.52 peticiones por segundo (RPS) y sin errores registrados.

Métricas Clave

Throughput (RPS)

- **Total RPS**: 4.52 peticiones/segundo (266 peticiones en 58 segundos)
  - GET /orders: 3.04 RPS (179 peticiones)
  - POST /carts: 1.48 RPS (87 peticiones)
- **Distribución**: Las peticiones GET son más del doble de frecuentes que las POST

Tiempos de Respuesta

- **Promedio general**: 148.51 ms
  - GET /orders: 144.25 ms
  - POST /carts: 157.26 ms
- **Percentiles agregados**:
  - 50%ile: 110 ms (la mitad de las peticiones se completan en ≤110ms)
  - 95%ile: 360 ms (95% se completan en ≤360ms)
  - Máximo: 660 ms

Tasa de Errores

- **0 fallos** en todas las peticiones (266 exitosas)

product-service

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/product-service/api/products	78	0	1413.56	272	6533	39901.33	1.35	0
POST	/product-service/api/products	43	0	1413.08	98	6404	254	0.74	0
PUT	/product-service/api/products/1	42	0	1695.75	104	6453	236	0.72	0
	Aggregated	163	0	1486.15	98	6533	19221.71	2.81	0

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/product-service/api/products	620	740	1100	2100	4100	6000	6500	6500
POST	/product-service/api/products	320	450	1500	3600	4000	4300	6400	6400
PUT	/product-service/api/products/1	810	1500	2400	3600	4100	4800	6500	6500
	Aggregated	600	810	1700	3400	4100	5600	6500	6500

Las pruebas de carga realizadas con Locust durante ~1 minuto contra el servicio de productos (http://localhost:8300/product-service) muestran un rendimiento degradado, con un throughput de 2.81 RPS y tiempos de respuesta elevados (promedio: 1,486 ms). Aunque no hubo errores, los percentiles altos (95%ile: 5,600 ms) indican problemas de latencia significativos.

Métricas Clave

Throughput (RPS)

- Total RPS: 2.81 peticiones/segundo (163 peticiones en ~58 segundos).
  - GET /products: 1.35 RPS (78 peticiones).
  - POST /products: 0.74 RPS (43 peticiones).
  - PUT /products/1: 0.72 RPS (42 peticiones).
- Distribución: Las peticiones GET son casi el doble de frecuentes que las POST/PUT.

Tiempos de Respuesta

- Promedio general: 1,486 ms (casi 1.5 segundos).
  - GET /products: 1,413 ms.
  - POST /products: 1,413 ms.
  - PUT /products/1: 1,695 ms (el más lento).
- Percentiles agregados:
  - 50%ile: 600 ms (la mitad de las peticiones tardan ≤600ms).
  - 95%ile: 5,600 ms (5% superan los 5.6 segundos).
  - Máximo: 6,533 ms (pico en un GET).

Tasa de Errores

- 0 fallos en todas las peticiones (163 exitosas).

user-service

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/user-service/api/address	77	0	159.97	94	675	24	1.32	0
GET	/user-service/api/credentials	54	0	179.7	92	973	24	0.92	0
GET	/user-service/api/users	76	0	173.74	92	688	24	1.3	0
GET	/user-service/api/verificationTokens	65	0	189.11	92	1015	24	1.11	0
	Aggregated	272	0	174.7	92	1015	24	4.66	0

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/user-service/api/address	120	130	140	180	310	440	680	680
GET	/user-service/api/credentials	110	120	120	190	380	600	970	970
GET	/user-service/api/users	120	130	190	210	390	420	690	690
GET	/user-service/api/verificationTokens	110	120	140	220	410	600	1000	1000
	Aggregated	120	120	140	200	360	450	760	1000

Las pruebas de carga realizadas con Locust durante ~1 minuto contra el servicio de usuarios (http://localhost:8300/user-service) muestran un rendimiento estable y eficiente, con un throughput de 4.66 RPS y tiempos de respuesta aceptables (promedio: 174.7 ms). No se registraron errores, y los percentiles altos (95%ile: 450 ms) indican un comportamiento consistente bajo carga.

Métricas Clave

Throughput (RPS)

- Total RPS: 4.66 peticiones/segundo (272 peticiones en ~58 segundos).
    - GET /address: 1.32 RPS (77 peticiones).
    - GET /credentials: 0.92 RPS (54 peticiones).
    - GET /users: 1.30 RPS (76 peticiones).
    - GET /verificationTokens: 1.11 RPS (65 peticiones).
- Distribución: Las peticiones están equilibradas, con /address y /users como las más frecuentes.

Tiempos de Respuesta

- Promedio general: 174.7 ms (óptimo para APIs).
    - GET /address: 159.97 ms (el más rápido).
    - GET /credentials: 179.7 ms.
    - GET /users: 173.74 ms.
    - GET /verificationTokens: 189.11 ms (el más lento, pero dentro de lo esperado).
- Percentiles agregados:
    - 50%ile: 120 ms (la mitad de las peticiones tardan ≤120ms).
    - 95%ile: 450 ms (solo el 5% supera los 450ms).
    - Máximo: 1,015 ms (pico en /verificationTokens).

Tasa de Errores

- 0 fallos en todas las peticiones (272 exitosas).

Reporte de de los resultados: debe entregar un documento que contenga la siguiente
información para cada uno de los pipelines:

- Configuración: Texto de la configuración de los pipelines, con pantallazos de configuración relevante en los mismos.
- Resultado: pantallazos de la ejecución exitosa de los pipelines con los detalles y resultados relevantes.
- Análisis: interpretación de los resultados de las pruebas, especialmente las de rendimiento, con métricas clave como tiempo de respuesta, throughput y tasa de errores.
- Release Notes: documentación de las versiones desplegadas en cada ambiente.

Adicionalmente, un zip con las pruebas implementadas.