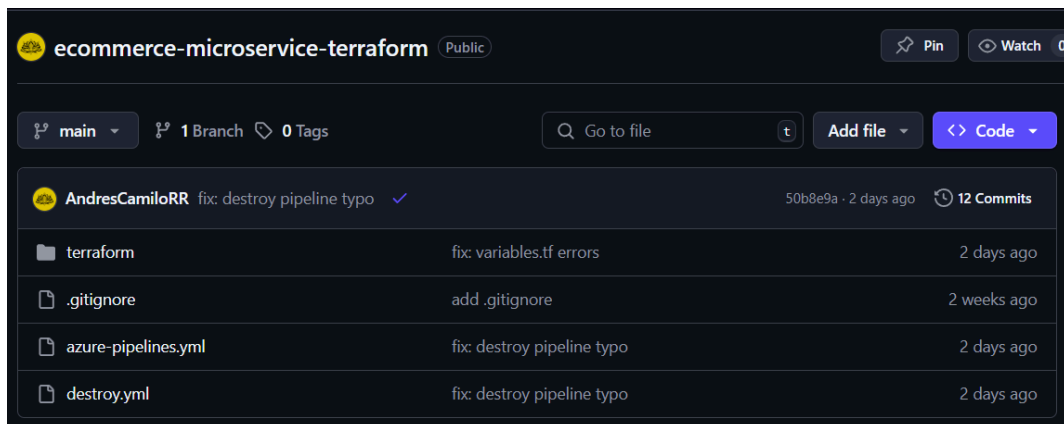


## Reporte taller 3

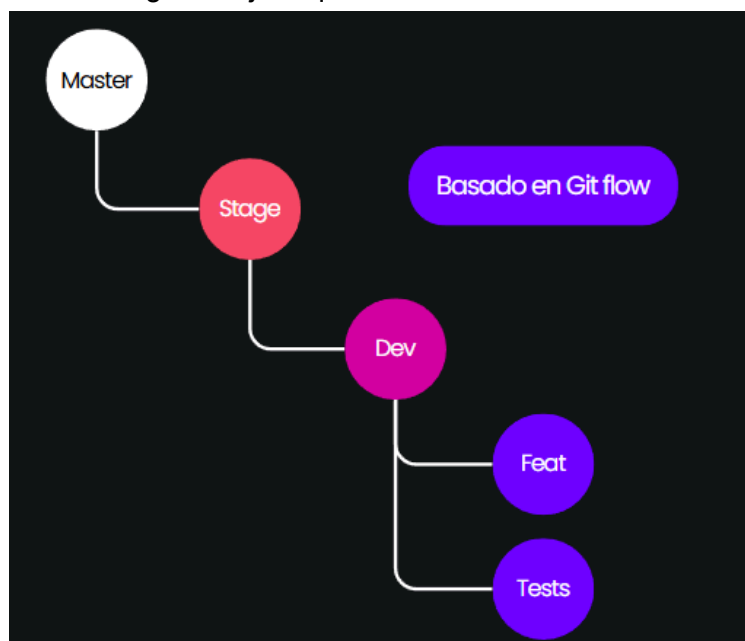
- Andrés Camilo Romero Ruiz
- Brayan Steven Ortega García

### Metodología Ágil y Estrategia de Branching

- Para el proyecto las estrategias de branching fueron las siguientes:
  - Infraestructura: se hizo un repositorio diferente con una única rama main en la cual se encuentran todos los archivos de terraform al igual que las pipelines de despliegue y destrucción



- Código: se hizo uso de una estrategia basada en gitflow con ramas master, stage, dev, features (el nombre que llevan es de libre elección) y tests donde tenemos la siguiente jerarquía



- Para la gestión del proyecto se hizo uso de Scrum y Jira donde podemos ver el backlog, historias, criterios de aceptación, estimación entre otros datos, que como se puede observar en las capturas de los informes ocurrieron 2 sprints.

Proyectos

Projecto final Ingesoft 5

Resumen

Cronograma

Backlog

Tablero

Calendario

Lista

Formularios

Metas

More 7

+

Q Buscar e...

AR +2

Epic

Tipo

Tablero Sprint 2

11 jun – 16 jun (11 actividades)

7 0 28

Completar sprint

<input checked="" type="checkbox"/>	PFI5-12	HU07 – Desarrollo de pruebas unitarias e integración	FINALIZADA	3	BO
<input type="checkbox"/>	PFI5-6	HU05 – Gestión de cambios y versionamiento	FINALIZADA	2	AR
<input type="checkbox"/>	PFI5-2	HU02 – Configuración e integración de CI/CD	FINALIZADA	5	AR
<input checked="" type="checkbox"/>	PFI5-14	HU09 – Automatización y cobertura de pruebas	FINALIZADA	2	AR
<input checked="" type="checkbox"/>	PFI5-15	HU10 – Observabilidad con métricas y logs	FINALIZADA	5	AR
<input checked="" type="checkbox"/>	PFI5-16	HU11 – Trazabilidad y probes	FINALIZADA	3	AR
<input type="checkbox"/>	PFI5-3	HU03 – Gestión de ambientes y despliegue controlado	FINALIZADA	3	AR
<input checked="" type="checkbox"/>	PFI5-13	HU08 – Pruebas E2E, rendimiento y seguridad	FINALIZADA	5	BO
<input checked="" type="checkbox"/>	PFI5-17	HU12 – Exposición de métricas de negocio	TAREAS POR HACER	2	
<input checked="" type="checkbox"/>	PFI5-18	HU13 – Documentación técnica del proyecto	TAREAS POR HACER	3	
<input checked="" type="checkbox"/>	PFI5-19	HU14 – Presentación final del proyecto	TAREAS POR HACER	2	

+ Crear

Quickstart

Proyectos

Projecto final Ingesoft 5

Resumen

Cronograma

Backlog

Tablero

Calendario

Lista

Formularios

Metas

More 7

+

Q Buscar ta...

AR BO

Filtro

Completar sprint

Grupo

POR HACER 3

HU12 – Exposición de métricas de negocio

☒ PFI5-17 2

HU13 – Documentación técnica del proyecto

☒ PFI5-18 3

HU14 – Presentación final del proyecto

☒ PFI5-19 2

+ Crear

EN CURSO

LISTO 8

HU07 – Desarrollo de pruebas unitarias e integración

☒ PFI5-12 3 BO

HU05 – Gestión de cambios y versionamiento

☐ PFI5-6 2 AR

HU02 – Configuración e integración de CI/CD

☐ PFI5-2 5 AR

HU09 – Automatización y cobertura de pruebas

☒ PFI5-14 2 AR

HU10 – Observabilidad con métricas y logs

☒ PFI5-15 5 AR

Quickstart

Sprint

Tablero Sprint 1

▼

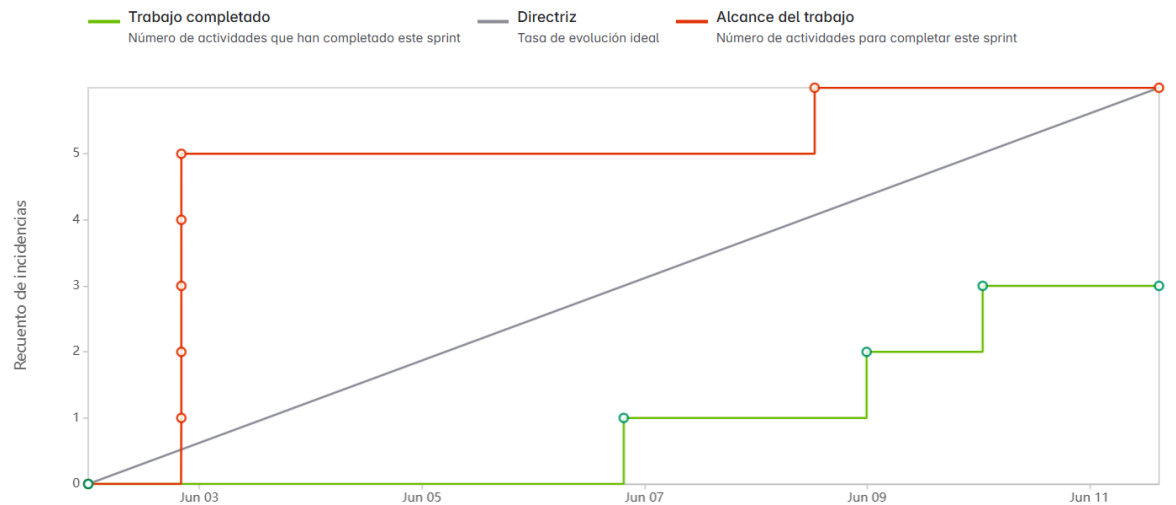
Campo de estimación

Recuento de actividades

▼

...

Fecha - 2 de junio de 2025 - 10 de junio de 2025



Sprint

Tablero Sprint 2

▼

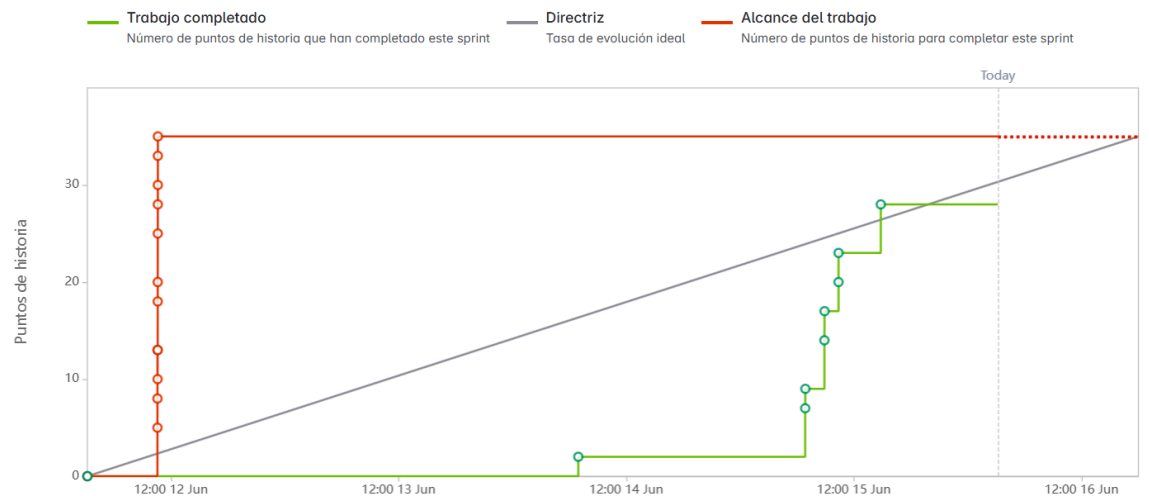
Campo de estimación

Puntos de historia

▼

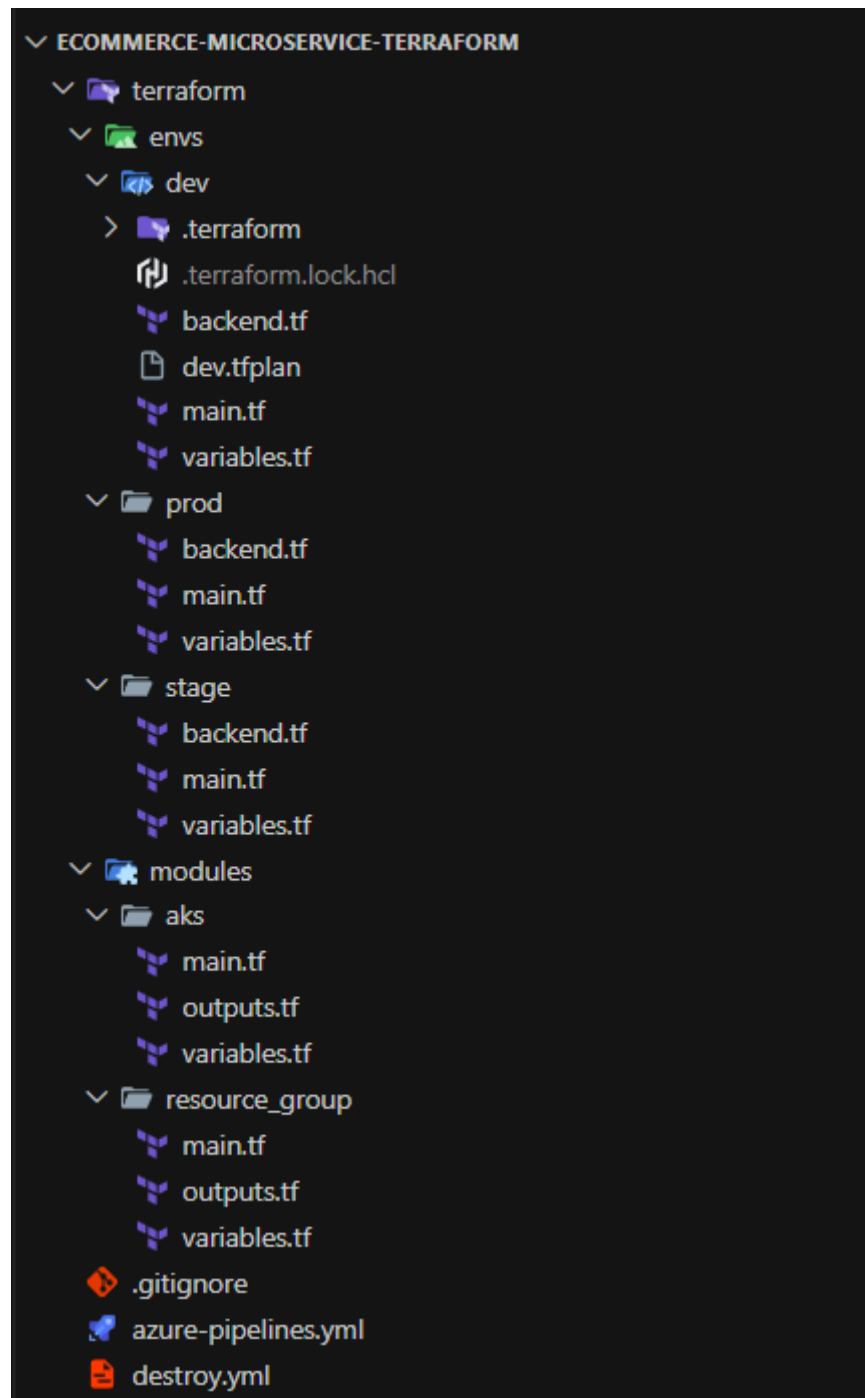
...

Fecha - 11 de junio de 2025 - 16 de junio de 2025



## Infraestructura como Código con Terraform

- Para llevar a cabo la parte de IaC se creó un repositorio aparte cuya única función es almacenar los archivos de tipo terraform con su respectiva información. En este caso la estructura es la siguiente:



- En primer lugar nos vamos a enfocar en la carpeta modules para que visualizar que se hizo uso de una estructura modular donde los dos recursos que se crean, un resource group y un aks cluster, se encuentran en carpetas separadas, cada uno con su main.tf sobre la creación del recurso, variables.tf como el conjunto de valores que esperan recibir y outputs.tf como los valores a retornar tras su creación. Las imágenes se encuentran abajo.

```
main.tf X
terraform > modules > aks > main.tf > resource "azurerm_kubernetes_cluster" "this" > role_based_access_control_enabled
1 resource "azurerm_kubernetes_cluster" "this" {
2   name           = var.cluster_name
3   location       = var.location
4   resource_group_name = var.resource_group_name
5   dns_prefix     = var.dns_prefix
6
7   default_node_pool {
8     name       = "default"
9     node_count = var.node_count
10    vm_size    = var.vm_size
11  }
12
13  identity {
14    type = "SystemAssigned"
15  }
16
17  role_based_access_control_enabled = true
18  tags = var.tags
19 }
```

```
outputs.tf X
terraform > modules > aks > outputs.tf > output "name"
1 output "kube_config_raw" { value = azurerm_kubernetes_cluster.this.kube_config_raw }
2 output "name"           { value = azurerm_kubernetes_cluster.this.name }
```

```
variables.tf X
terraform > modules > aks > variables.tf > variable "tags"
1 variable "cluster_name" { type = string }
2 variable "location"    { type = string }
3 variable "resource_group_name" { type = string }
4 variable "dns_prefix"   { type = string }
5 variable "node_count"   { type = number }
6 variable "vm_size"      { type = string }
7 variable "tags"         { type = map(string) }
```

- En segundo lugar podemos visualizar una carpeta envs, con otras 3 carpetas, dev, stage y prod. Cada una contiene un main.tf que accede a ambos módulos para su despliegue y define el proveedor a usar. También encontramos al backend.tf, indicando dónde se va a guardar de manera remota el .tfstate. Y un archivo de variables.tf que contiene los valores que serán asignados a las variables de cada módulo. Las imágenes se encuentran abajo.

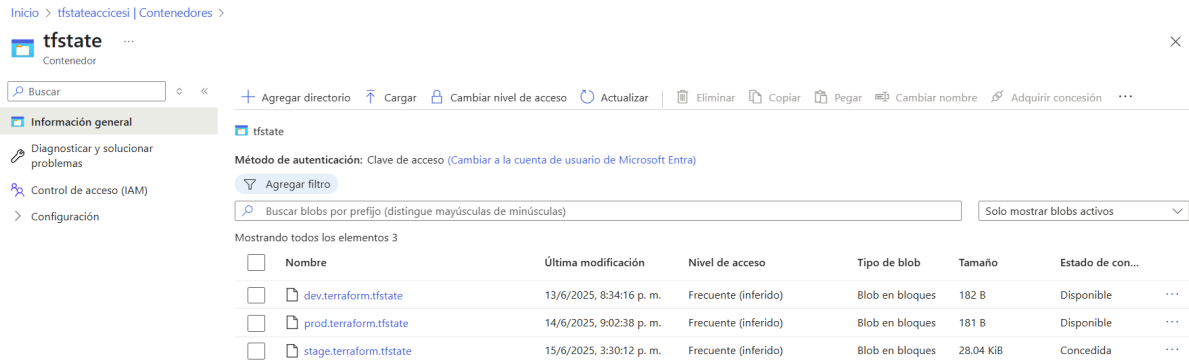
```
main.tf X
terraform > envs > dev > main.tf > module "aks"
1  provider "azurerm" {
2    features {}
3  }
4
5  module "rg" {
6    source = "../modules/resource_group"
7    name   = var.resource_group_name
8    location = var.location
9  }
10
11 module "aks" {
12   source           = "../modules/aks"
13   cluster_name    = var.aks_cluster_name
14   location         = module.rg.location
15   resource_group_name = module.rg.name
16   dns_prefix      = var.dns_prefix
17   node_count      = var.node_count
18   vm_size         = var.vm_size
19   tags            = { environment = var.environment }
20 }
```

```
backend.tf X
terraform > envs > dev > backend.tf > terraform > backend "azurerm" > storage_account_name
1 terraform {
2   backend "azurerm" {
3     resource_group_name = "tfstate-rg"
4     storage_account_name = "tfstateaccicesi"
5     container_name      = "tfstate"
6     key                  = "dev.terraform.tfstate"
7   }
8 }
```

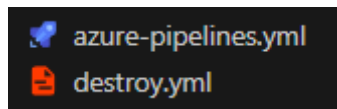
```
variables.tf X
terraform > envs > dev > variables.tf > variable "environment"
1 variable "resource_group_name" { default = "rg-dev" }
2 variable "location"            { default = "eastus" }
3 variable "aks_cluster_name"    { default = "aks-dev" }
4 variable "dns_prefix"          { default = "devaks" }
5 variable "node_count"          { default = 1 }
6 variable "vm_size"             { default = "Standard_DS2_v2" }
7 variable "environment"         { default = "dev" }
```

- Como podemos observar cada ambiente puede tener una configuración personalizada, es decir, se pueden elegir los módulos a usar, el proveedor, la

ubicación del backend remoto, el nombre del contenedor y la llave con que será registrado, al igual que las variables que sean definidas en cada módulo, en este caso se puede elegir el nombre del grupo de recursos, su ubicación, el nombre del aks, su prefijo en el dns, el número de nodos desplegados, el tamaño de la máquina virtual (sus recursos) y el ambiente que será usado como etiqueta al desplegar.



- En cuanto al backend remoto adjunto una imagen de la cuenta de almacenamiento y el contenedor donde se pueden observar los 3 tf.state según el perfil. Este recurso debe estar desplegado ya que usamos la TerraformTask@5 en nuestra pipeline y exige de forma obligatoria que el backend remoto exista antes de ejecutar cualquier comando de terraform



- Se realizaron 2 pipelines con azure devops, una de despliegue y otra de destrucción. Ambas acceden a un service connection que creamos para que las pipelines puedan tener permisos sobre la suscripción y poder crear los recursos. Ambas pipelines se deben ejecutar de forma manual y le preguntan al usuario por el ambiente que se está tratando

Run pipeline

Select parameters below and manually run the pipeline

Branch/tag

main

Select a branch from the list or enter the name of a tag as refs/tags/<tagname>

Commit

Selecciona un entorno

dev

stage

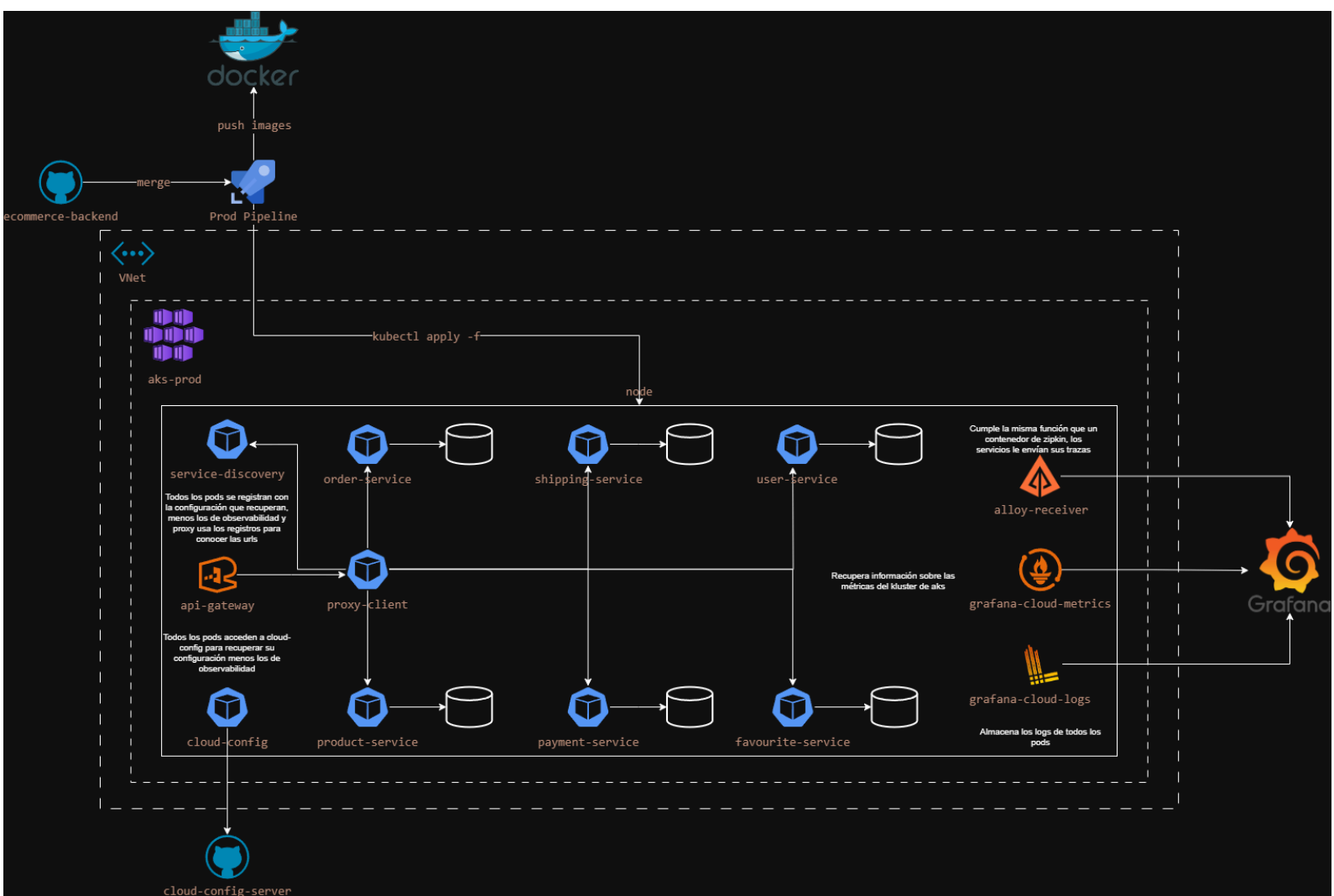
prod

```

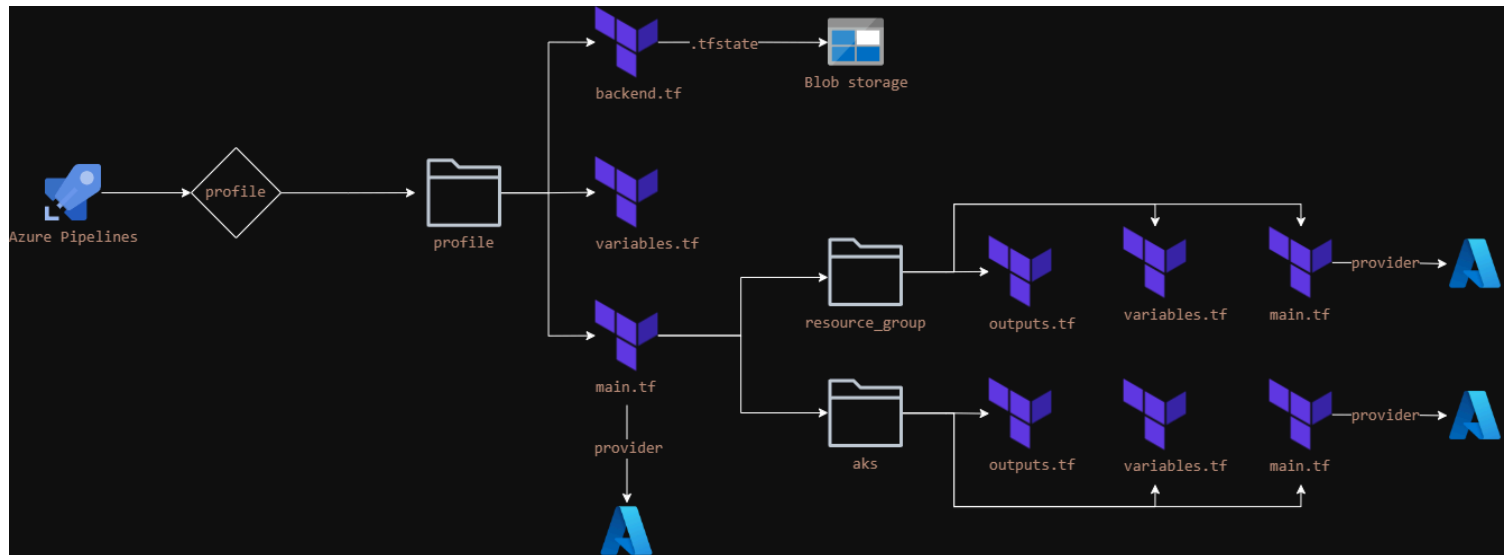
1  trigger: none
2  pr: none
3
4  parameters:
5    - name: environment
6      displayName: 'Selecciona un entorno'
7      type: string
8      default: dev
9      values:
10     - dev
11     - stage
12     - prod
13
14  variables:
15    - name: azureSubscription
16      value: 'MiServiceConnectionAzure'

```

- En cuanto a la infraestructura del proyecto se hicieron los siguientes 2 diagramas al respecto, en caso tal de que alguno de los dos no logren visualizarse adecuadamente sus pdfs también se encuentran en la entrega para una mayor calidad







## Patrones de Diseño

Para los patrones de diseño primero se identificaron los patrones implementados en la infraestructura del proyecto hasta el momento de la entrega del taller 2, y los patrones que se lograron identificar fueron los siguientes:

- Circuit Breaker
- API Gateway
- Service Discovery
- Database per Service
- Command and Query Responsibility Segregation (CQRS)
- External Configuration (Cloud Config)
- Load Balancer
- Health Check
- Saga

Por lo que en base a esto se decidió implementar los siguientes patrones a la infraestructura actual del proyecto:

- **Retry pattern (para resiliencia):**
  - El propósito de este patrón es permitir reintentar automáticamente operaciones fallidas ante errores transitorios, mejorando la resiliencia frente a fallos temporales de red o servicios externos.
  - Sus beneficios son mayor tolerancia a fallos, reducción de errores visibles para el usuario
- **Feature toggle pattern (para configuración):**
  - El propósito de este patrón es permitir habilitar o deshabilitar funcionalidades de la aplicación en tiempo de ejecución sin necesidad de desplegar nuevo código

- Sus beneficios son que facilita pruebas A/B, despliegues progresivos y gestión de funcionalidades experimentales
- Cache aside pattern (para rendimiento/eficiencia):
  - El propósito de este patrón es optimizar el acceso a datos almacenando en caché los resultados de consultas frecuentes y manteniendo la coherencia al invalidar el caché en operaciones de escritura
  - Sus beneficios son la reducción de la latencia, menor carga sobre los servicios backend, mejora del rendimiento general.

### Retry pattern:

- Este patrón se implementó dentro del servicio proxy client para cada uno de los servicios que este tiene mapeado en su carpeta business, los servicios en cuestion son favourite, order, shipping, payment, product y user, en cada uno se añade el importe de retry a través resilience4j y se puso la anotación correspondiente para cada método, como se ve a continuación:

```
import com.selimhorri.app.business.orderItem.model.OrderItemDto;
import com.selimhorri.app.business.orderItem.model.OrderItemId;
import com.selimhorri.app.business.orderItem.model.response.OrderItemOrderItemServiceDtoCollectionResponse;
import io.github.resilience4j.retry.annotation.Retry;

@FeignClient(name = "SHIPPING-SERVICE", contextId = "shippingClientService", path = "/shipping-service/api/shippings")
public interface OrderItemClientService {

    @GetMapping
    @Retry(name = "shippingService")
    ResponseEntity<OrderItemOrderItemServiceDtoCollectionResponse> findAll();

    @GetMapping("/{orderId}/{productId}")
    @Retry(name = "shippingService")
    ResponseEntity<OrderItemDto> findById(
        @PathVariable("orderId") final String orderId,
        @PathVariable("productId") final String productId);

    @GetMapping("/find")
    @Retry(name = "shippingService")
    ResponseEntity<OrderItemDto> findById(
        @RequestBody
```

- También para configurar el funcionamiento del patrón dentro del servicio de proxy client, en su application.yml, se agrego la configuración del retry donde se declara el número de intentos, el tiempo de espera entre cada reintento, las excepciones a tener en cuenta, y las instancias que utilizarían esta configuración que en este caso corresponden a todos los servicios dentro de proxy, la configuración se ve a

continuación:

```
resilience4j:
  retry:
    configs:
      default:
        max-attempts: 3
        wait-duration: 10s
        retry-exceptions:
          - java.io.IOException
          - java.util.concurrent.TimeoutException
          - feign.RetryableException
          - org.springframework.web.client.ResourceAccessException
          - feign.FeignException$ServiceUnavailable
        ignore-exceptions:
          - org.springframework.web.client.HttpClientErrorException.BadRequest
          - org.springframework.web.client.HttpClientErrorException.NotFound
          - org.springframework.web.client.HttpClientErrorException.Conflict
          - org.springframework.web.client.HttpClientErrorException.Unauthorized
          - feign.FeignException.BadRequest
          - feign.FeignException.NotFound
          - feign.FeignException.Conflict
          - feign.FeignException.Unauthorized
          - feign.FeignException.Forbidden
    instances:
      userService:
        base-config: default
      productService:
        base-config: default
      orderService:
        base-config: default
      paymentService:
        base-config: default
      shippingService:
        base-config: default
      favouriteService:
        base-config: default
```

- Y ya como última parte de la configuración del patrón dentro de proxy tenemos en la carpeta config de proxy client, un archivo de configuración para tener logs personalizados en consola al momento de que se esté ejecutando el retry, y estos logs son los siguientes:

```
@Autowired
public RetryLoggingConfiguration(RetryRegistry retryRegistry) {
    this.retryRegistry = retryRegistry;
}

@PostConstruct
public void registerRetryLogging() {
    retryRegistry.getAllRetries().forEach(retry -> {
        retry.getEventPublisher().onRetry(event -> {
            LOGGER.info(
                format:"Retrying operation '{}', attempt {}. Last throwable: {}",
                event.getName(),
                event.getNumberOfRetryAttempts(),
                event.getLastThrowable() != null ? event.getLastThrowable().getClass().getName() + ": " + event.getLastThrowable()
            );
        });

        retry.getEventPublisher().onError(event -> {
            LOGGER.warn(
                format:"Operation '{}' failed after {} attempts. Last throwable: {}",
                event.getName(),
                event.getNumberOfRetryAttempts(),
                event.getLastThrowable() != null ? event.getLastThrowable().getClass().getName() + ": " + event.getLastThrowable()
            );
        });

        retry.getEventPublisher().onSuccess(event -> {
            // Only log if retries actually happened before success
            if (event.getNumberOfRetryAttempts() > 0) {
                LOGGER.info(
                    format:"Operation '{}' succeeded after {} attempts. Last throwable during retries: {}",
                    event.getName(),
                    event.getNumberOfRetryAttempts(),
                    event.getLastThrowable() != null ? event.getLastThrowable().getClass().getName() + ": " + event.getLastThrowable()
                );
            }
        });
    });
}
```

- Ahora observemos un ejemplo de cómo se vería el patrón en funcionamiento, en un ejemplo donde realiza 2 reintentos y en el segundo consigue realizar la operación:

```

r.JwtRequestFilter : **JwtRequestFilter, once per request, validating and extracting token*

atcherServlet      : GET "/app/api/users/2", parameters={}
appingHandlerMapping : Mapped to con.selinhorri.app.business.user.controller.UserController#findById(

binLoadBalancer    : No servers available for service: USER-SERVICE
ngLoadBalancerClient : Load balancer does not contain an instance for the service USER-SERVICE
ggingConfiguration : Retrying operation 'userService', attempt 1. Last throwable: feign.FeignExcept
ring)): [Load balancer does not contain an instance for the service USER-SERVICE]
binLoadBalancer    : No servers available for service: USER-SERVICE
ngLoadBalancerClient : Load balancer does not contain an instance for the service USER-SERVICE
ggingConfiguration : Retrying operation 'userService', attempt 2. Last throwable: feign.FeignExcept
ring)): [Load balancer does not contain an instance for the service USER-SERVICE]
http://service-discovery-container:8761/eureka/apps/PROXY-CLIENT/cf116cccb7c1:PROXY-CLIENT:8900?statu

application/json, application/*+json]
200 OK
http://service-discovery-container:8761/eureka/apps/delta
application/json, application/*+json]
200 OK
o [org.springframework.cloud.netflix.eureka.http.EurekaApplications]
onverterExtractor   : Reading to [com.selinhorri.app.business.user.model.UserDto]
ggingConfiguration : Operation 'userService' succeeded after 2 attempts. Last throwable during retr
entService#findById(String)): [Load balancer does not contain an instance for the service USER-SERVIC

tityMethodProcessor : Using 'application/json', given [*/] and supported [application/json, applica
tityMethodProcessor : Writing [UserDto(userId=2, firstName=anine, lastName=ladjini, imageUrl=https:/

atcherServlet      : Completed 200 OK
r.JwtRequestFilter : **Jwt request filtered!*

http://zipkin-container:9411/api/v2/spans

```

Feature toggle pattern:

- Este patrón se implementó dentro de los servicios cloud config (quien hace de gestor para external configuration en el proyecto) y product service que es al servicio que se le hace el cambio de funcionalidad dinámico, primero en cloud config se ajustó para que hiciera correctamente pull de los cambios que se realizan en el repositorio de cloud-config-server que es quien guarda la configuración de forma externa y también se añadió la configuración para conectarse a un intermediario de mensajes que en este caso es rabbitmq:

```

server:
  port: 9296

spring:
  zipkin:
    base-url: ${SPRING_ZIPKIN_BASE_URL:http://localhost:9411/}
  application:
    name: CLOUD-CONFIG
  cloud:
    config:
      enabled: false
      server:
        git:
          uri: https://github.com/BrayanOrteg/cloud-config-server
          clone-on-start: true
          default-label: master
          force-pull: true
  rabbitmq:
    host: ${SPRING_RABBITMQ_HOST:localhost}
    port: ${SPRING_RABBITMQ_PORT:5672}
    username: ${SPRING_RABBITMQ_USERNAME:guest}
    password: ${SPRING_RABBITMQ_PASSWORD:guest}

```

- Para poder implementar el patrón y que el cambio se hiciera de forma dinámica, fue necesario implementar rabbitmq como intermediario de mensajes y para añadirlo, se agregó en el compose y la pipeline la inicialización del contenedor para este servicio

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rabbitmq
  template:
    metadata:
      labels:
        app: rabbitmq
    spec:
      containers:
        - name: rabbitmq
          image: rabbitmq:3-management-alpine
          ports:
            - containerPort: 5672
            - containerPort: 15672
          env:
            - name: RABBITMQ_DEFAULT_USER
              value: "guest"
            - name: RABBITMQ_DEFAULT_PASS
              value: "guest"
          volumeMounts:
            - name: rabbitmq-data
              mountPath: /var/lib/rabbitmq
      volumes:
        - name: rabbitmq-data
          persistentVolumeClaim:
            claimName: rabbitmq-data
```

```
- task: KubernetesManifest@1
  displayName: 'Deploy RabbitMQ'
  inputs:
    action: 'deploy'
    connectionType: 'azureResourceManager'
    azureSubscriptionConnection: $(azureSubscription)
    azureResourceGroup: $(RESOURCE_GROUP)
    kubernetesCluster: $(CLUSTER_NAME)
    namespace: 'default'
    manifests: '$(K8S_MANIFEST_DIR)/rabbitmq.yaml'
```

- Ahora para hacer efectiva la implementación del patrón, fue necesario dentro de 1 servicio en este caso product service, hacer los siguientes cambios:
  - Primero en la clase de ProductServiceImpl se añadió la anotación de “@RefreshScope” para que dentro de la clase se pudieran hacer cambios dinámicos cuando se publica un mensaje en rabbitmq que avisa a este servicio que se realizó un cambio en la configuración que debe tener en cuenta. Dentro de la misma clase también se hizo una inyección de dependencia de la variable que viene de la configuración externa que señala

el comportamiento del método como se ve a continuación:

```
@Service
@Transactional
@Slf4j
@RequiredArgsConstructor
@RefreshScope
public class ProductServiceImpl implements ProductService {

    private final ProductRepository productRepository;

    @Value("${features.newSearchAlgorithm.enabled:false}")
    private boolean newSearchAlgorithmEnabled;

    @Override
    public List<ProductDto> findAll() {
        log.info(msg:"*** ProductDto List, service; fetch all products *");

        if (newSearchAlgorithmEnabled) {
            log.info(msg:"Using NEW search algorithm to fetch all products!");
        } else {
            log.info(msg:"Using OLD search algorithm to fetch all products.");
        }

        return this.productRepository.findAll()
            .stream()
            .map(ProductMappingHelper::map)
            .distinct()
            .collect(Collectors.toUnmodifiableList());
    }
}
```

- Luego se realizó un cambio dentro de application.yml para que el servicio se pueda conectar y ser listener en rabbitmq:

```
server:
  servlet:
    context-path: /product-service

spring:
  zipkin:
    base-url: ${SPRING_ZIPKIN_BASE_URL:http://localhost:9411/}
  profiles:
    active:
      - dev
  rabbitmq:
    host: ${SPRING_RABBITMQ_HOST:localhost}
    port: ${SPRING_RABBITMQ_PORT:5672}
    username: ${SPRING_RABBITMQ_USERNAME:guest}
    password: ${SPRING_RABBITMQ_PASSWORD:guest}
```

- Por último se creó un archivo bootstrap.yml que se encarga de traer la configuración para el servicio de cloud config y de un archivo de configuración llamado PRODUCT-SERVICE.YML donde se encuentra la configuración del servicio y de la variable newSearchAlgorithm que cambia el comportamiento de un método del servicio.

```

spring:
  application:
    name: PRODUCT-SERVICE
  config:
    import: ${SPRING_CONFIG_IMPORT:optional:configserver:http://localhost:9296}
  cloud:
    config:
      fail-fast: true

```

- Por último en el repositorio de cloud-config-server se añadió el archivo PRODUCT-SERVICE.yml que contiene la configuración necesaria y la variable que modifica el comportamiento del servicio:

```

1  eureka:
2    instance:
3      prefer-ip-address: false
4      hostname: product-service
5    client:
6      serviceUrl:
7        defaultZone: http://service-discovery:8761/eureka/
8
9  features:
10    newSearchAlgorithm:
11      enabled: true

```

- Para visualizar el correcto funcionamiento del patrón se tienen las siguientes capturas donde en el momento en que la variable newSearchAlgorithm que en el momento de estar en false, el servicio al llamar a su método de getAll() suelta el siguiente mensaje:

```

o.s.web.servlet.DispatcherServlet      : GET "/product-service/api/products", parameters={}
s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to com.selimhorri.app.resource.ProductResource#findAll()
c.s.app.resource.ProductResource        : *** ProductDto List, controller; fetch all categories *
c.s.app.service.impl.ProductServiceImpl : *** ProductDto List, service; fetch all products *
c.s.app.service.impl.ProductServiceImpl : Using OLD search algorithm to fetch all products.
org.hibernate.SQL                      :

```

- Y cuando esta con valor true, el cambio se hace de forma dinámica en el cloud config, se manda por medio de un llamado a que cloud config publique un mensaje de cambio en rabbitmq y finalmente product service coge este mensaje y cambia su configuración para ahora soltar este valor en consola al momento de llamar al mismo método

```

DispatcherServlet      : GET "/product-service/api/products", parameters={}
RequestMappingHandlerMapping : Mapped to com.selimhorri.app.resource.ProductResource#findAll()
ProductResource        : *** ProductDto List, controller; fetch all categories *
ProductServiceImpl     : *** ProductDto List, service; fetch all products *
ProductServiceImpl     : Using NEW search algorithm to fetch all products!
QL                     :

```

Cache aside pattern:

- Este patrón se implementó dentro de proxy client en los métodos que más información pueden traer que son los `getAll()` de algunos servicios como user service y product service, el patrón se añadió en la controller de los servicios a diferencia del patrón retry que se encuentra en las implementaciones de los servicios. El caché aside implementado se ve así:

```

import lombok.RequiredArgsConstructor;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.cache.annotation.CacheEvict;

@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserController {

    private final UserClientService userClientService;

    @Cacheable(value = "users")
    @GetMapping
    public ResponseEntity<UserUserServiceCollectionDtoResponse> findAll() {
        return ResponseEntity.ok(this.userClientService.findAll().getBody());
    }

    @Cacheable(value = "userById", key = "#userId")
    @GetMapping("/{userId}")
    public ResponseEntity<UserDto> findById(@PathVariable("userId") final String userId) {
        return ResponseEntity.ok(this.userClientService.findById(userId).getBody());
    }

    @Cacheable(value = "userByUsername", key = "#username")
    @GetMapping("/username/{username}")
    public ResponseEntity<UserDto> findByUsername(@PathVariable("username") final String username) {
        return ResponseEntity.ok(this.userClientService.findByUsername(username).getBody());
    }

    @PostMapping
    @CacheEvict(value = {"users", "userById", "userByUsername"}, allEntries = true)
    public ResponseEntity<UserDto> save(@RequestBody final UserDto userDto) {
        return ResponseEntity.ok(this.userClientService.save(userDto).getBody());
    }

    @PutMapping
    @CacheEvict(value = {"users", "userById", "userByUsername"}, allEntries = true)
    public ResponseEntity<UserDto> update(@RequestBody final UserDto userDto) {
        return ResponseEntity.ok(this.userClientService.update(userDto).getBody());
    }
}

```

- En cuanto a la configuración que se realizó dentro de proxy client para dejar funcionando cache aside, tenemos un archivo de configuración llamado `CacheConfig`, donde se crea un bean `cacheManager` proveniente de spring framework, y dentro de este bean se declaran los `cacheNames` que sirven para



mapear el cache con un nombre en específico para poder encontrarlo como se ve a continuación:

```
package com.selinhorri.app.config;

import org.springframework.cache.CacheManager;
import org.springframework.cache.concurrent.ConcurrentMapCacheManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class CacheConfig {

    @Bean
    public CacheManager cacheManager() {
        return new ConcurrentMapCacheManager(
            ...cacheNames:"users", "userById", "userByUsername",
            "products", "productById",
            "categories", "categoryById"
        );
    }
}
```

- en funcionamiento el patron cuando se llama por primera vez a uno de los métodos y este no se encuentra en cache en consola podemos ver el siguiente mensaje en los logs:

```
Servlet.DispatcherServlet : GET "/app/api/products", parameters={}
a.RequestMappingHandlerMapping : Mapped to com.selinhorri.app.business.product.controller.ProductController#findAll()
interceptor.CacheInterceptor : Computed cache key 'SimpleKey []' for operation Builder[public org.springframework.http.ResponseEntity
ss='' | sync='false'
interceptor.CacheInterceptor : No cache entry for key 'SimpleKey []' in cache(s) [products]
interceptor.CacheInterceptor : Computed cache key 'SimpleKey []' for operation Builder[public org.springframework.http.ResponseEntity
ss='' | sync='false'
tpMessageConverterExtractor : Reading to [com.selinhorri.app.business.product.model.response.ProductProductServiceCollectionDtoRe
n.a.HttpEntityMethodProcessor : Using 'application/json', given [/*] and supported [application/json, application/*+json, applicat
n.a.HttpEntityMethodProcessor : Writing [ProductProductServiceCollectionDtoResponse(collection=[ProductDto(productId=1, productTitl
Servlet.DispatcherServlet : Completed 200 OK
nfig.filter.JwtRequestFilter : **Jwt request filtered!**

HTTP POST http://zipkin-container:9411/api/v2/spans
Accept=[text/plain, application/json, application/*+json, /*]
Writing [B@2950a16b] as "application/json"
Response 202 ACCEPTED
```

- y cuando se vuelve a hacer el llamado y ya la información sobre ese llamado ya se encuentra en cache el mensaje que se muestra es el siguiente:

```

: Reading to [org.springframework.cloud.netflix.eureka.http.EurekaApplications]
app.config.filter.JwtRequestFilter : **JwtRequestFilter, once per request, validating and extracting token*

web.servlet.DispatcherServlet : GET "/api/products", parameters={}
s.m.a.RequestMappingHandlerMapping : Mapped to con.selinhorri.app.business.product.controller.ProductController#findAll()
cache.interceptor.CacheInterceptor : Computed cache key 'SimpleKey []' for operation Builder[public org.springframework.http.ResponseEntity
unless='' | sync='false'
cache.interceptor.CacheInterceptor : Cache entry for key 'SimpleKey []' found in cache 'products'
v.s.m.a.HttpEntityMethodProcessor : Using 'application/json', given [*/] and supported [application/json, application/*+json, application
v.s.m.a.HttpEntityMethodProcessor : Writing [ProductProductServiceCollectionDtoResponse(collection=[ProductDto(productId=1, productTitle=
web.servlet.DispatcherServlet : Completed 200 OK
app.config.filter.JwtRequestFilter : **Jwt request filtered!*

: HTTP POST http://zipkin-container:9411/api/v2/spans
: Accept=[text/plain, application/json, application/*+json, */]

```

- Para tener en cuenta, cuando se hace un llamado que cambie la base de datos como un llamado create, update o delete sobre un servicio que tiene implementado cache, se borra automáticamente el caché para cuando se vuelva a realizar un get, éste no se traiga de caché con información vieja, sino que haga el llamado a base de datos para traer la información más reciente y luego se guarde esto en caché.

## CI/CD Avanzado

- Para la implementación de las pipelines se hizo uso de azure devops con la misma organización creada al inicio del curso para no perder tiempo solicitando permisos para el paralelismo. Las pipelines implementadas son Deploy y Destroy en cuanto al terraform, Dev Pipeline, Stage Pipeline y Prod Pipeline en cuanto al código




















Pipelines

New pipeline

RecentAllRuns

Filter pipelines

Recently run pipelines

Pipeline	Last run
 Deploy	#20250615.5 • fix: max of 4 vcpus for students subscription Manually triggered for  main  Just now
 Destroy	#20250615.4 • refactor: change node_count for a faster deployment Manually triggered for  main  12m ago  6m 53s
 Stage Pipeline	#20250615.6 • feat: merge pull request #4 from BrayanOrtega/dev Individual CI for  stage  53m ago  19m 26s
 Dev Pipeline	#20250615.6 • fix: correction in postman collection Individual CI for  dev  1h ago  14m 38s
 Prod Pipeline	#20250615.7 • fix: turn of for PRs Individual CI for  master  14h ago  15s

- Para la promoción controlada se crearon security rules sobre las ramas master, stage y dev

Ruleset Name \*

master

Enforcement status

⌚ Disabled

Bypass list

+ Add bypass

Exempt roles, teams, and apps from this ruleset by adding them to the bypass list.

Bypass list is empty

Targets

Which branches do you want to make a ruleset for?

Target branches

Branch targeting determines which branches will be protected by this ruleset. Use inclusion patterns to expand the list of branches under this ruleset. Use exclusion patterns to exclude branches.

Branch targeting criteria

Add target

+ master

Applies to 1 target: master

Rules

Which rules should be applied?

Branch rules

☐ Restrict creations

Only allow users with bypass permission to create matching refs.

☐ Restrict updates

Only allow users with bypass permission to update matching refs.

☒ Restrict deletions

Only allow users with bypass permissions to delete matching refs.

☐ Require linear history

Prevent merge commits from being pushed to matching refs.

☐ Require deployments to succeed

Choose which environments must be successfully deployed to before refs can be pushed into a ref that matches this rule.

☐ Require signed commits

Commits pushed to matching refs must have verified signatures.

☒ Require a pull request before merging

Require all commits be made to a non-target branch and submitted via a pull request before they can be merged.

Hide additional settings ^

Required approvals

2

The number of approving reviews that are required before a pull request can be merged.

☒ Dismiss stale pull request approvals when new commits are pushed

New, reviewable commits pushed will dismiss previous pull request review approvals.

☐ Require review from Code Owners

Require an approving review in pull requests that modify files that have a designated code owner.

☐ Require approval of the most recent reviewable push

Whether the most recent reviewable push must be approved by someone other than the person who pushed it.

☐ Require conversation resolution before merging

All conversations on the pull request must be resolved before merging and merged pull requests are merged.

☒ Require status checks to pass

Choose which status checks must pass before the ref is updated. When enabled, commits must first be pushed to another ref where the checks pass.

Show additional settings v

☒ Block force pushes

Prevent users with push access from force pushing to refs.

☐ Require code scanning results

Choose which tools must provide code scanning results before the reference is updated. When configured, code scanning must be enabled and have results for both the commit and the reference being updated.

Save changes

Revert changes

- Se implementó SonarQube usando SonarCloud y las tasks de SonarCloudPrepare@3 que prepara la conexión con SonarCloud con el service connection y Maven@4 que se encarga de ejecutar el escaneo con el Sonar

```
- stage: BuildProject
  displayName: 'Build Project with SonarQube Analysis'
  jobs:
    - job: BuildJob
      steps:
        - task: SonarCloudPrepare@3
          inputs:
            SonarQube: 'ecommerce-sonar-qube'
            organization: 'brayan-organization'
            scannerMode: 'other'
            extraProperties: |
              sonar.projectKey=brayan-organization_First-pipeline
              sonar.projectName=First pipeline

        - task: Maven@4
          inputs:
            mavenPomFile: 'pom.xml'
            goals: 'clean package'
            options: '-DskipTests'
            publishJUnitResults: false
            javaHomeOption: 'JDKVersion'
            mavenVersionOption: 'Default'
            mavenAuthenticateFeed: false
            effectivePomSkip: false
            sonarQubeRunAnalysis: true
```

ecommerce-sonar-qube

Edit

ID: 881f759b-f782-4958-86af-2d77c2de399f


Overview

Usage history


Approvals and checks

Details

Service connection type

 SonarCube Cloud  
using token based authentication

Creator

 ANDRES CAMILO ROMERO RUIZ  
1105361345@uicesi.edu.co

★ brayan-organization / First pipeline

Private

Passed

Last analysis: 14/06/2025, 21:55 • 8k Lines of Code • Java, XML

E 2

Security

A 0

Reliability

A 64

Maintainability

E 0.0%

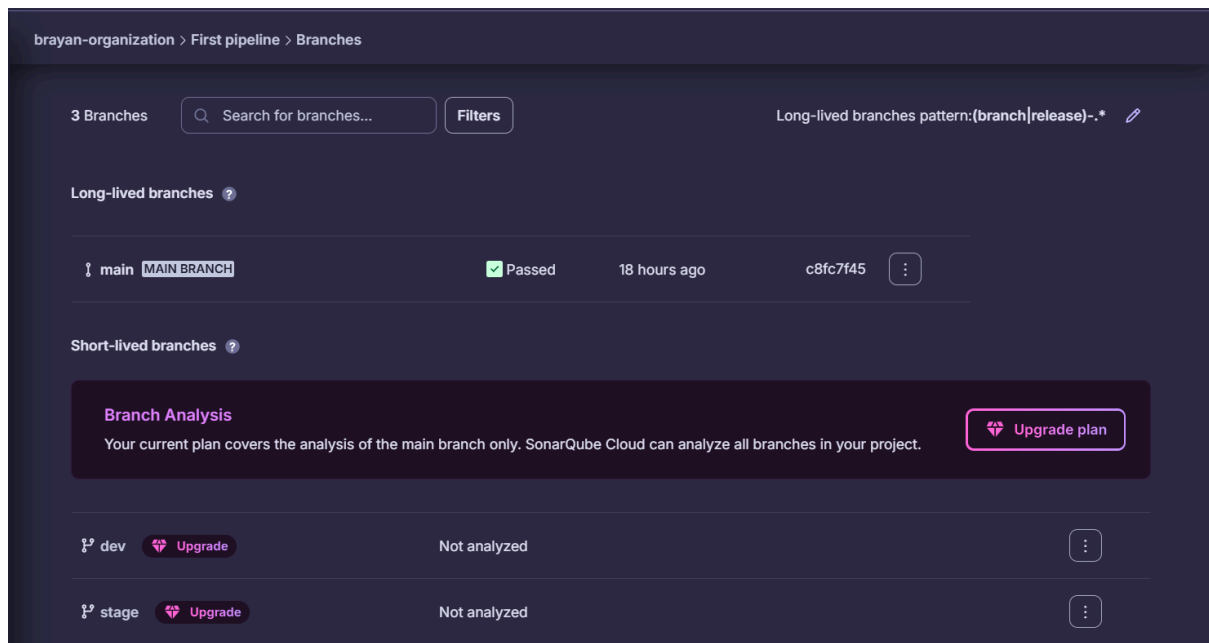
Hotspots Reviewed

0.0%

Coverage

9.4%

Duplications



- Para la ejecución de Trivy y el escaneo de vulnerabilidades se instala dentro del agente y se ejecuta un análisis sobre todas las imágenes después de hacer build pero antes de publicar en docker hub. Se configuró que el Trivy continúe su ejecución sin importar las vulnerabilidades que encuentre ya que las imágenes parten de openjdk:11 que posee 20 vulnerabilidades críticas y 56 de alto riesgo por lo que el Trivy siempre se detiene con el objetivo de evitar que las imágenes sean usadas, pero para el desarrollo del taller este aviso será ignorado.

```
- script: |
  docker compose -f ${COMPOSE_YML} build
  displayName: 'Build Docker Images'

- script: |
  echo "Installing Trivy..."
  sudo apt-get update
  sudo apt-get install -y wget apt-transport-https gnupg lsb-release
  wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
  echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" | sudo tee /etc/apt/sources.list.d/trivy.list
  sudo apt-get update
  sudo apt-get install -y trivy
  trivy --version
  displayName: 'Install Trivy'

- script: |
  echo "Scanning Docker Images with Trivy..."
  declare -A service_images
  service_images["cloud-config-container"]="kenbra/cloud-config-e-commerce-boot"
  service_images["api-gateway-container"]="kenbra/api-gateway-e-commerce-boot"
  service_images["proxy-client-container"]="kenbra/proxy-client-e-commerce-boot"
  service_images["order-service-container"]="kenbra/order-service-e-commerce-boot"
  service_images["payment-service-container"]="kenbra/payment-service-e-commerce-boot"
  service_images["product-service-container"]="kenbra/product-service-e-commerce-boot"
  service_images["shipping-service-container"]="kenbra/shipping-service-e-commerce-boot"
  service_images["user-service-container"]="kenbra/user-service-e-commerce-boot"
  service_images["favourite-service-container"]="kenbra/favourite-service-e-commerce-boot"
  for service_name in "${!service_images[@]}"; do
    image_name="${service_images[$service_name]}${IMAGE_TAG_SUFFIX}"
    echo "--- Scanning image: $image_name ---"
    trivy_image "$image_name" || echo "Trivy scan failed for $image_name but continuing pipeline..."
  done
  displayName: 'Scan Docker Images with Trivy'

- script: |
  docker compose -f ${COMPOSE_YML} push
  displayName: 'Push Docker Images'
```

Target	Type	Vulnerabilities	Secrets
***order-service-ecommerce-boot:stage (debian 11.4)	debian	620	-
home/app/order-service.jar	jar	82	-

Legend:  
- '-': Not scanned  
- '0': Clean (no security findings detected)

\*\*\*order-service-ecommerce-boot:stage (debian 11.4)  
=====

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apt	CVE-2011-3374	LOW	affected	2.2.4		It was found that apt-key in apt, all versions, do not correctly... <a href="https://avd.aquasec.com/nvd/cve-2011-3374">https://avd.aquasec.com/nvd/cve-2011-3374</a>
bash	CVE-2022-3715	HIGH		5.1-2+deb11u1		bash: a heap-buffer-overflow in valid_parameter_transform <a href="https://avd.aquasec.com/nvd/cve-2022-3715">https://avd.aquasec.com/nvd/cve-2022-3715</a>
	TEMP-0841856-B18BAF	LOW				[Privilege escalation possible to other user than root] <a href="https://security-tracker.debian.org/tracker/TEMP-0841856-B1-8BAF">https://security-tracker.debian.org/tracker/TEMP-0841856-B1-8BAF</a>

- Para el versionado semántico se hizo uso de @semantic-release que se instala con npm y se configura un archivo de .releaserc y se añaden las dependencias desarrollo al package.json:

.releaserc:

```
{
  "branches": ["master"],
  "plugins": [
    "@semantic-release/commit-analyzer",
    "@semantic-release/release-notes-generator",
    "@semantic-release/changelog",
    [
      "@semantic-release/git",
      {
        "assets": ["CHANGELOG.md"],
        "message": "chore(release): ${nextRelease.version} [skip ci]\n\n${nextRelease.notes}"
      }
    ],
    [
      "@semantic-release/github",
      {
        "successComment": false,
        "failComment": false,
        "failTitle": false,
        "releasedLabels": false
      }
    ]
  ]
}
```

package.json

```
"devDependencies": {
  "@semantic-release/changelog": "^6.0.3",
  "@semantic-release/git": "^10.0.1",
  "@semantic-release/github": "^11.0.3",
  "semantic-release": "^24.2.5"
}
```

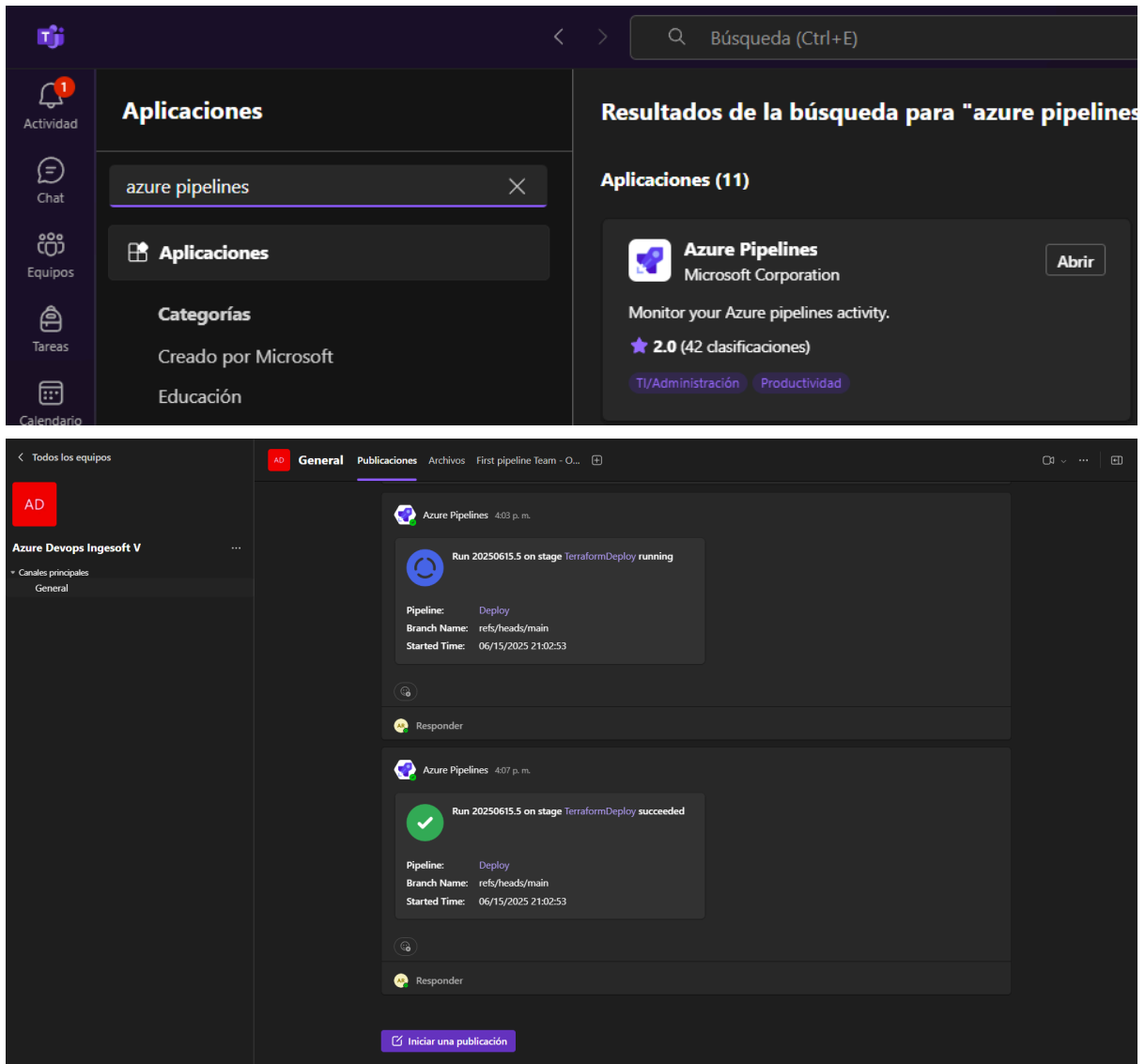
- Es entonces como dentro de la pipeline se añaden los siguientes pasos para instalar las dependencias y ejecutar el versionado semántico, @semantic-release se encarga automáticamente de revisar todos los commits existentes entre dos tags/versiones del código y genera las release notes automáticamente, todo commit con el formato “feat:” será considerado MINOR, “fix:” como PATCH y “perf:” o “BREAKING CHANGE:” se consideran MAJOR

```
- stage: SemanticRelease
  displayName: 'Semantic Release Stage'
  jobs:
    - job: SemanticReleaseJob
      displayName: 'Install and Run Semantic Release'
      steps:
        - task: NodeTool@0
          inputs:
            versionSpec: '20.x'
            displayName: 'Install Node.js'

        - script: |
            npm ci
            displayName: 'Install dependencies'

        - script: |
            npx semantic-release
            displayName: 'Run semantic-release'
```

- Para las notificaciones automáticas como estamos trabajando con azure devops descubrimos que existe una aplicación dentro de teams que permite vincular la organización y proyecto con el teams de tal forma que notifique sobre toda pipeline, su estado si es exitoso o fallido y el paso en que falló, su configuración se hace a través del propio chat de teams donde envía los mensajes:



## Aprobación Manual

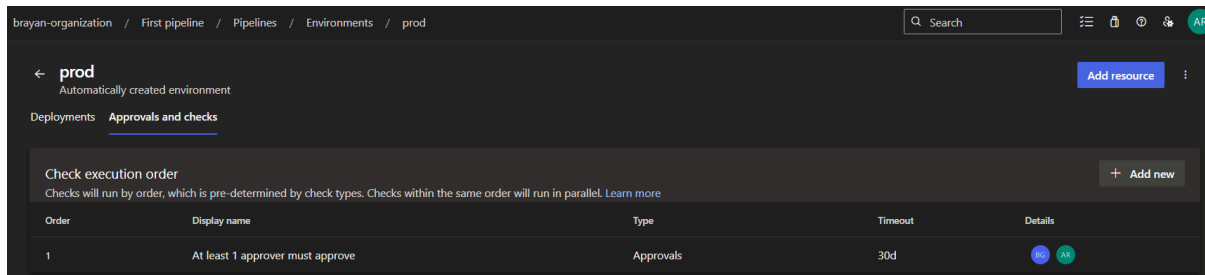
- Para la aprobación del despliegue se creó un deploy job y un environnement que exige aprobaciones por parte de los usuarios elegidos

```

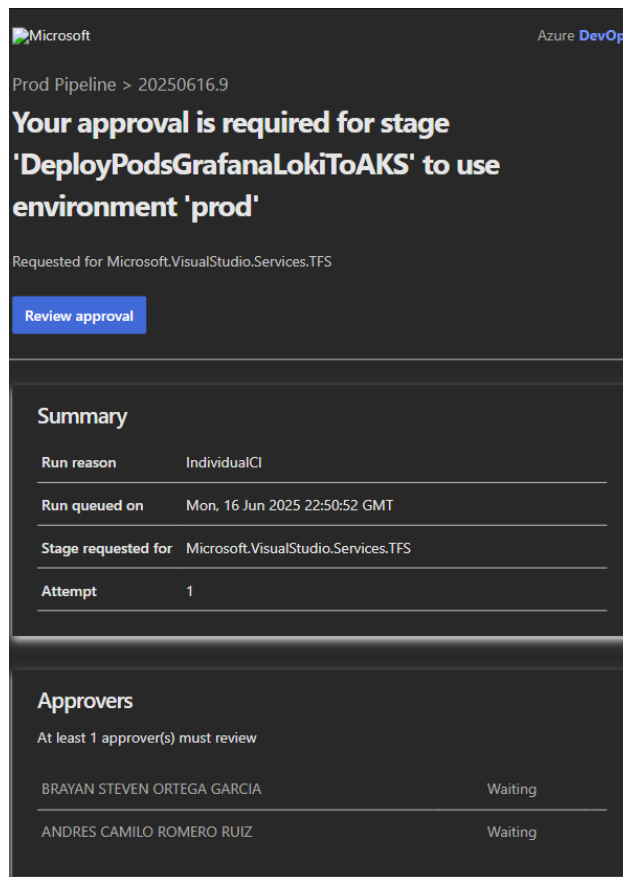
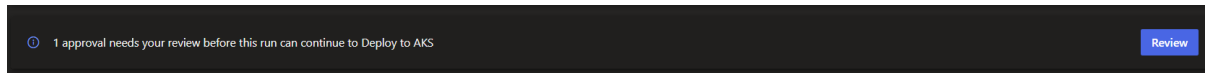
- stage: DeployPodsGrafanaLokiToAKS
- displayName: 'Deploy to AKS'
- dependsOn: PushDockerImages
- jobs:
- deployment: DeployJob
- displayName: "Deployment"
- environment: "prod"
- strategy:
- runOnce:
- deploy:
- steps:

```





- De esta forma al llegar a la etapa de despliegue la pipeline se detiene hasta recibir la aprobación de alguno de los usuarios que pertenecen al grupo. Cada uno es notificado a través del correo.



## Pruebas Completas

- La automatización de las pruebas se hizo de la siguiente forma:
  - **Unitarias y de Integración** con publicación de coverage de Jacoco y reporte de Junit

```
- stage: RunTests
  displayName: 'Run Unit and Integration Tests'
  dependsOn: BuildProject
  jobs:
    - job: TestJob
      steps:
        - task: Maven@4
          inputs:
            mavenPomFile: 'pom.xml'
            goals: 'test'
            publishJUnitResults: true
            testResultsFiles: '**/surefire-reports/TEST-*.xml'
            codeCoverageToolOption: 'JaCoCo'
            javaHomeOption: 'JDKVersion'
            mavenVersionOption: 'Default'
            mavenAuthenticateFeed: false
            effectivePomSkip: false
            sonarQubeRunAnalysis: false

        - task: PublishCodeCoverageResults@2
          inputs:
            summaryFileLocation: '**/site/jacoco/jacoco.xml'
            reportDirectory: '**/site/jacoco'
            codecoverageTool: 'jacoco'
            displayName: 'Publish Code Coverage Results'
            condition: succeededOrFailed()
```

[/ First pipeline](#) / [Pipelines](#) / [Stage Pipeline](#) / 20250615.6

#20250615.6 • feat: merge pull request #4 from BrayanOrteg/dev

Stage Pipeline

This run is being retained as one of 3 recent runs by pipeline.

Summary

Tests

Code Coverage

Summary

1 Run(s) Completed ( 1 Passed, 0 Failed )

83

Total tests

83 Passed

0 Failed

0 Others

100%

Pass percentage

23s 156ms

Run duration 

↓ 1s 494ms

0

Tests not reported

[/ First pipeline](#) / [Pipelines](#) / [Stage Pipeline](#) / 20250615.6

#20250615.6 • feat: merge pull request #4 from BrayanOrteg/dev

Stage Pipeline

This run is being retained as one of 3 recent runs by pipeline.

Summary

Tests

Code Coverage

Summary

Information

Parser:

MultiReport (10x JaCoCo)

Assemblies:

55

Classes:

185

Files:

161

Coverage date:

06/15/2025 - 20:12:46 - 06/15/2025 - 20:14:25

Line coverage

22%

Covered lines:

397

Uncovered lines:

1377

Coverable lines:

1774

Total lines:

5571

Line coverage:

22.3%

Branch coverage

2%

Covered branches:

64

Total branches:

2520

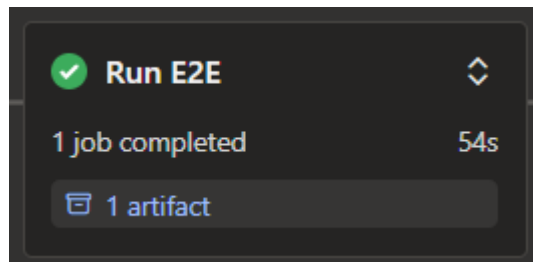
Branch coverage:

2.5%

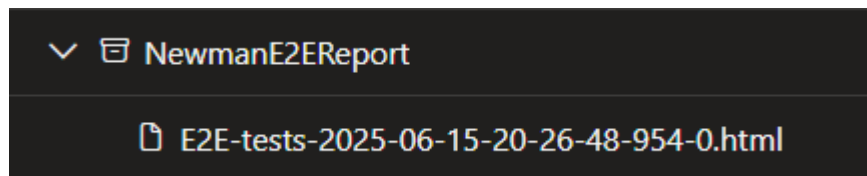
- **E2E** haciendo uso de una colección de Postman ejecutándose con Newman con la task NewmanPostman@4, se genera un artefacto en html extra con el reporte

```
- stage: E2E
  displayName: 'Run E2E'
  dependsOn:
    - OWASP
    - DeployToAKS
  variables:
    E2E_BASE_URL_FOR_NEWMAN: $[ stageDependencies.DeployToAKS.DeployJob.outputs['SetE2EVar.E2E_BASE_URL'] ]
  jobs:
    - job: NewmanJob
      steps:
        - script: |
            echo "E2E_BASE_URL_FOR_NEWMAN in E2E stage is $(E2E_BASE_URL_FOR_NEWMAN)"
            if [ -z "$(E2E_BASE_URL_FOR_NEWMAN)" ] || [ "$(E2E_BASE_URL_FOR_NEWMAN)" == "http://:8080" ]; then
            echo "##vso[task.logissue type=error;]E2E_BASE_URL_FOR_NEWMAN is empty or invalid in E2E stage."
            fi
          displayName: 'Debug E2E Base URL in E2E Stage'
        - task: AzureCLI@2
          inputs:
            azureSubscription: $(azureSubscription)
            scriptType: 'bash'
            scriptLocation: 'inlineScript'
            inlineScript: 'az aks get-credentials --resource-group $(RESOURCE_GROUP) --name $(CLUSTER_NAME) --overwrite-existing'
        - script: |
            API_GATEWAY_IP=$(kubectl get service api-gateway --namespace default -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
          displayName: 'Get API Gateway IP'
        - task: Npm@1
          inputs:
            command: 'custom'
            customCommand: 'install newman newman-reporter-htmlextra -g'
          displayName: 'Install Newman and htmlextra reporter'
        - task: NewmanPostman@4
          inputs:
            collectionSourceType: 'file'
            collectionFileSource: 'postman-collections/E2E-tests.postman_collection.json'
            environmentSourceType: 'none'
            globalVars: 'base_url=$(E2E_BASE_URL_FOR_NEWMAN)'
            reporters: 'htmlextra'
            htmlExtraDarkTheme: true
            htmlExtraLogs: true
            htmlExtraTestPaging: true
          displayName: 'Run Newman E2E Tests'
        - task: PublishBuildArtifacts@1
          inputs:
            PathToPublish: '$(System.DefaultWorkingDirectory)/newman'
            ArtifactName: 'NewmanE2EReport'
          displayName: 'Publish Newman E2E Report'
          condition: succeededOrFailed()
```

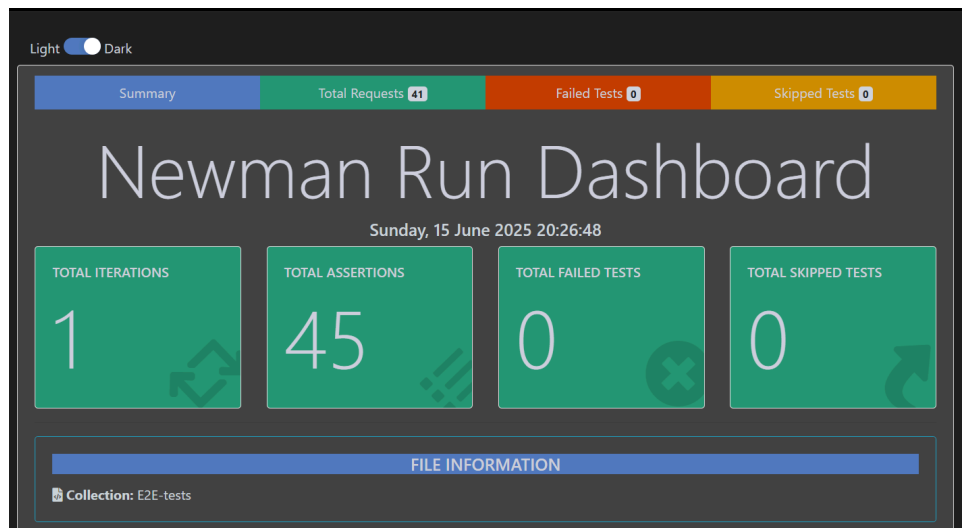
○



○



○



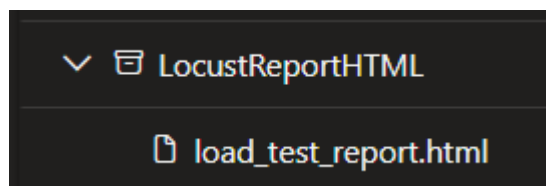
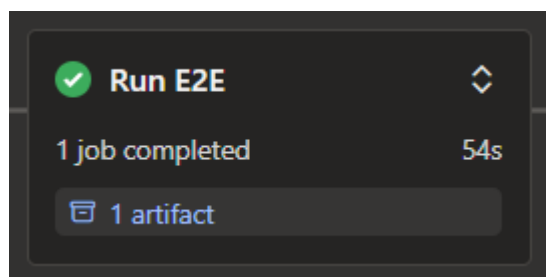
- **Estrés** con Locust, se instala python, se realizan varias validaciones para obtener la url del HOST, validar que exista y sea la correcta, para finalmente ejecutar la prueba y exportar el informe html como artefacto

```
export LOCUST_HOST="$LOCAL_HOST_FOR_LOCUST_SCRIPT"
echo "LOCUST_HOST environment variable will be set to: '$LOCUST_HOST' for locust execution"

echo "Executing Locust..."
python3 -m locust -f locustfile.py \
  --headless \
  -u $(LOCUST_USERS) \
  -r $(LOCUST_SPAWN_RATE) \
  -t $(LOCUST_RUN_TIME) \
  --csv=load_test_report \
  --html=load_test_report.html

echo "Locust execution finished. Checking for report files in $(pwd):"
ls -la
displayName: 'Debug and Run Locust Test'

- task: PublishBuildArtifacts@1
  displayName: 'Publish Locust HTML Report'
  inputs:
    pathToPublish: '$(Build.SourcesDirectory)/locust/load_test_report.html'
    artifactName: 'LocustReportHTML'
    publishLocation: 'Container'
  condition: always()
```



# Locust Test Report

**During:** 15/6/2025, 3:27:31 p. m. - 15/6/2025, 3:28:01 p. m. (30 seconds)

**Target Host:** http://9.169.14.10:8080

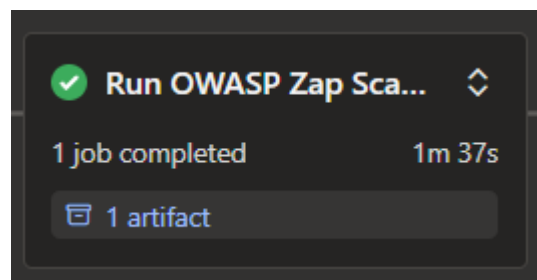
**Script:** locustfile.py

## Request Statistics

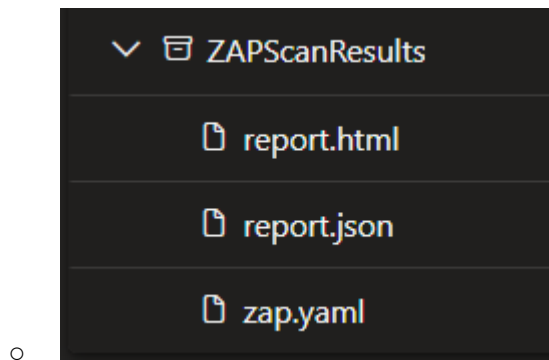
- 
- El reporte de locust se encuentra en el archivo `reporte_locust.md`
- **Seguridad** con ZAP, se trae la imagen de docker hub, se ejecuta el contenedor de ZAP para ejecutar las pruebas sobre la url y finalmente publicar el artefacto

```
- stage: OWASP
  displayName: 'Run OWASP Zap Scanner'
  dependsOn:
    - DeployToAKS
  variables:
    ZAP_TARGET_URL_FROM_DEPLOY: $[ stageDependencies.DeployToAKS.DeployJob.outputs['SetE2EVar.E2E_BASE_URL'] ]
  jobs:
    - job: OWASPJob
      steps:
        - script: |
            echo "Creating ZAP working directory..."
            mkdir -p $(System.DefaultWorkingDirectory)/owaspzap
            echo "Setting permissions for ZAP working directory..."
            sudo chmod -R 777 $(System.DefaultWorkingDirectory)/owaspzap
            echo "Pulling ZAP Docker image..."
            docker pull ghcr.io/zaproxy/zaproxy:stable
            echo "Running ZAP baseline scan against $(ZAP_TARGET_URL_FROM_DEPLOY)..."
            docker run --rm \
              -v $(System.DefaultWorkingDirectory)/owaspzap:/zap/wrk/:rw \
              ghcr.io/zaproxy/zaproxy:stable zap-baseline.py \
              -t $(ZAP_TARGET_URL_FROM_DEPLOY)/user-service/api/users \
              -J report.json -r report.html
            echo "ZAP scan finished. Reports are in $(System.DefaultWorkingDirectory)/owaspzap"
          displayName: 'Run OWASP ZAP Baseline Scan'
        - task: PublishBuildArtifacts@1
          inputs:
            PathToPublish: '$(System.DefaultWorkingDirectory)/owaspzap'
            ArtifactName: 'ZAPScanResults'
          displayName: 'Publish ZAP Scan Results'
          condition: succeededOrFailed()
```

○



○



Site: <http://9.169.14.10:8080>

Generated on Sun, 15 Jun 2025 20:25:38

ZAP Version: 2.16.1

ZAP by [Checkmarx](#)

#### Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	0
Low	3
Informational	2
False Positives	0

## Change Management y Release Notes

- En cuanto a la generación automática de release notes y etiquetado como fue mencionado anteriormente @semantic-release se encarga de eso y lo publica en CHANGELOG.md

## Planes de Rollback

Durante la implementación del proyecto se establecieron mecanismos de rollback tanto a nivel de infraestructura como de código, con el fin de garantizar la estabilidad del sistema frente a errores en despliegues o fallos durante el runtime. A continuación se detallan los planes establecidos:

### 1. Rollback de Infraestructura (Terraform)

Para la gestión de infraestructura se usó Terraform de forma modular, con estados remotos separados por entorno (dev, stage, prod). Esto permitió una mayor trazabilidad y control de versiones sobre los recursos desplegados en Azure.

#### Estrategia de rollback:

- Se habilitó una pipeline manual de destrucción (Destroy pipeline) que puede ser utilizada como mecanismo de emergencia si se requiere revertir completamente los recursos de un entorno específico.

- Los entornos e imágenes están separados, por lo que los cambios en dev o stage pueden evaluarse y revertirse sin afectar a prod.

## **2. Rollback de Despliegues en AKS (CI/CD)**

Para la entrega continua se usaron pipelines en Azure DevOps que incluyen stages con despliegue manual, pruebas automatizadas y escaneo de seguridad. Se definieron entornos (environments) en Azure DevOps con aprobaciones manuales que permiten validar antes de aplicar cambios.

### **Estrategia de rollback en AKS:**

- Se implementaron readiness y liveness probes en los servicios, lo que evita que un despliegue defectuoso entre en producción. Si el despliegue falla, Kubernetes evita exponer el pod a tráfico.
- En caso de falla crítica tras un despliegue en prod, se puede ejecutar una pipeline manual que aplique el chart de la versión anterior directamente o se puede hacer un rollback con Helm desde la CLI.

## **3. Rollback de Código Fuente**

El proyecto adopta un enfoque basado en Gitflow, lo que proporciona una estructura clara para gestionar versiones estables del código.

### **Estrategia de rollback en código:**

- Solo se permite la integración a master a través de Pull Requests desde stage, que a su vez provienen de dev, lo que reduce la probabilidad de introducir cambios no testeados.
- En caso de error en producción tras un merge, se puede revertir directamente el commit problemático con `git revert` o desplegar un tag anterior usando `@semantic-release`, que mantiene el versionado automático.
- Las pipelines están integradas con SonarCloud, Trivy y ZAP, por lo que cualquier versión que no supere los umbrales de calidad y seguridad es automáticamente rechazada antes de ser promovida.
- En cada release, se genera un changelog con `@semantic-release`, facilitando la identificación de versiones estables.

## **4. Rollback de Configuraciones Dinámicas**

El patrón Feature Toggle implementado con Spring Cloud Config y RabbitMQ permite habilitar o deshabilitar funcionalidades de forma dinámica.

### **Estrategia de rollback de configuración:**

- Los cambios en configuración se almacenan en un repositorio Git independiente (cloud-config-server) y se propagan vía RabbitMQ.

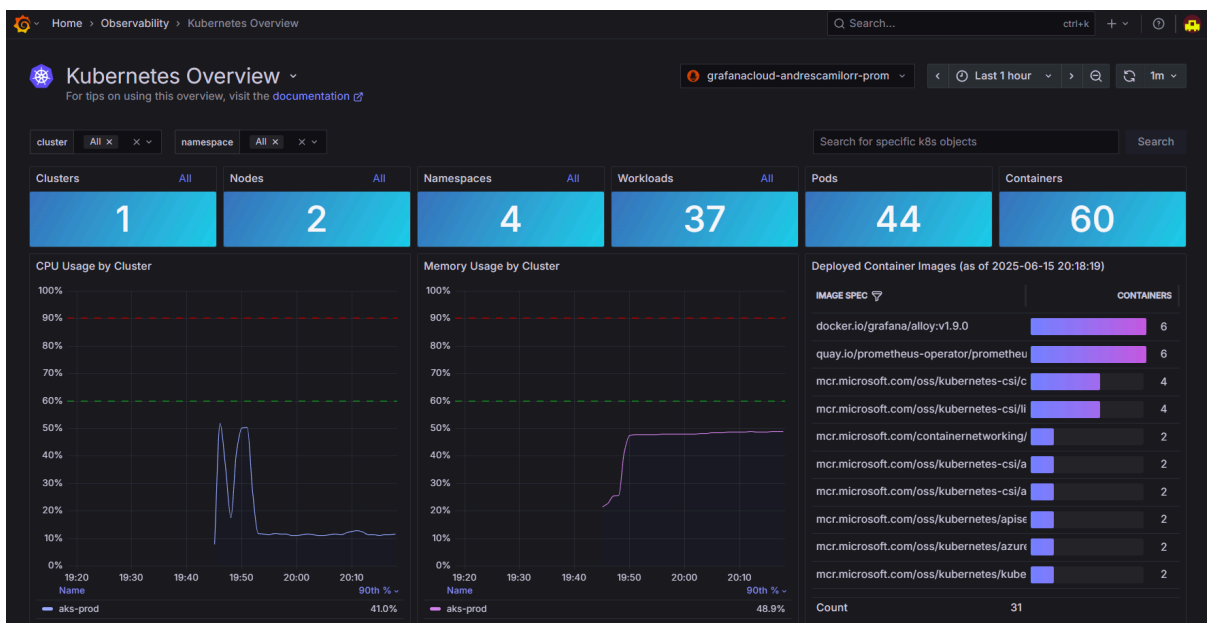
- Para revertir un cambio de configuración, basta con modificar el archivo de configuración remoto y realizar un git push, lo cual se propaga automáticamente a los servicios mediante @RefreshScope.
- Esta arquitectura permite realizar rollback de configuraciones sin necesidad de redeploy.

## Observabilidad y Monitoreo

- Para el monitoreo se hizo uso de Grafana cloud donde tras seguir el flujo de kubernetes, start sending data, instalar el backend, elegir Azure AKS, generar el token y usar helm para el despliegue. Nos va a devolver un comando que debemos ejecutar dentro una máquina con kubectl y acceso al cluster para desplegar los pods que se encargan de recuperar métricas y enviarlas a grafana cloud, en este caso hacemos uso de grafana-cloud-metrics (prometheus), grafana-cloud-logs (loki) y alloy-receive (zipkin).

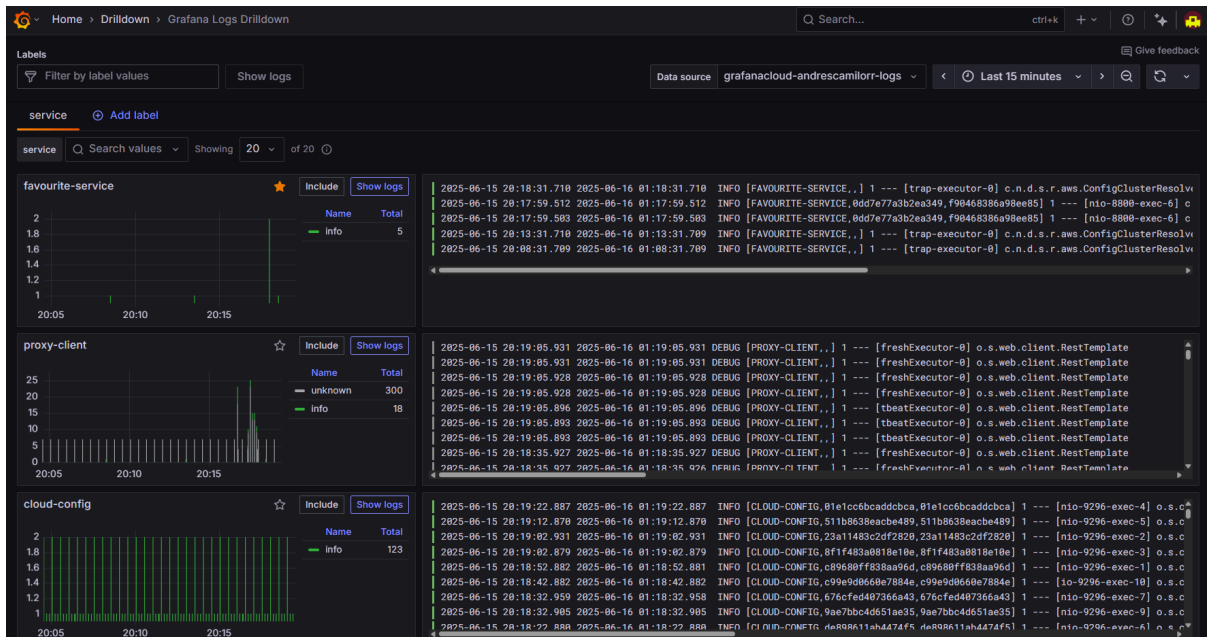
```
helm repo add grafana https://grafana.github.io/helm-charts &&
helm repo update &&
helm upgrade --install --atomic --timeout 300s grafana-k8s-monitoring gra
fana/k8s-monitoring \
  --namespace "default" --create-namespace --values - <<'EOF'
cluster:
  name: my-cluster
destinations:
  - name: grafana-cloud-metrics
    type: prometheus
    url: https://prometheus-prod-56-prod-us-east-2.grafana.net./api/prom/pu
```

- Estadísticas del cluster:



- Gestión de logs con loki:

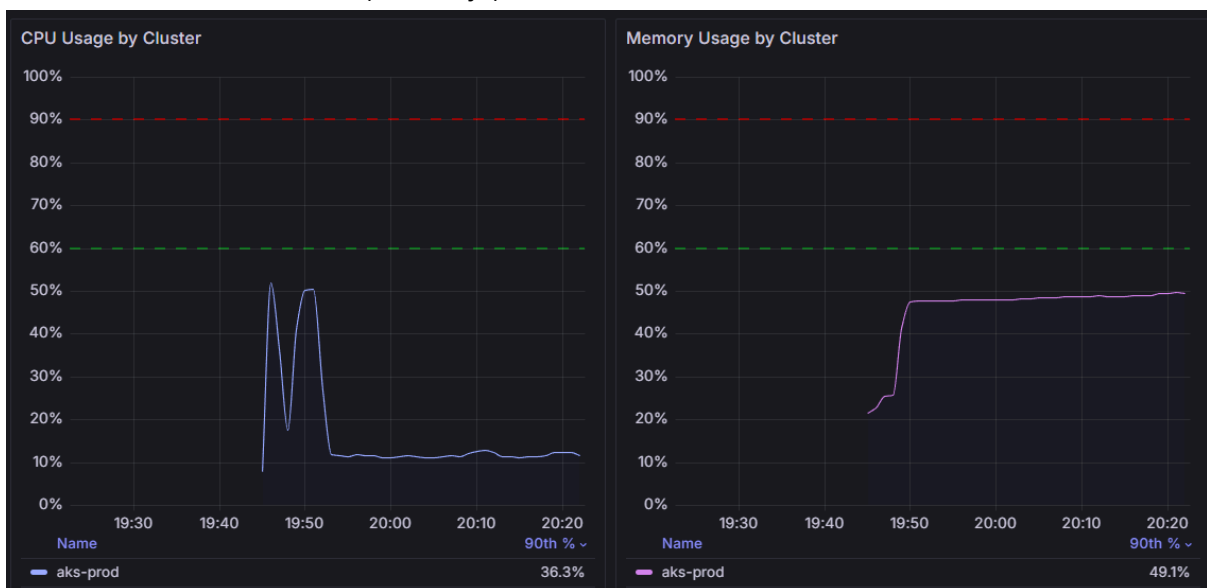




- Gestión de trazas zipkin en grafana:



- En cuanto a las alertas configuradas para situaciones críticas con grafana se configura de forma automática cuando el uso de cpu o memoria aks superan los límites establecidos (línea roja)



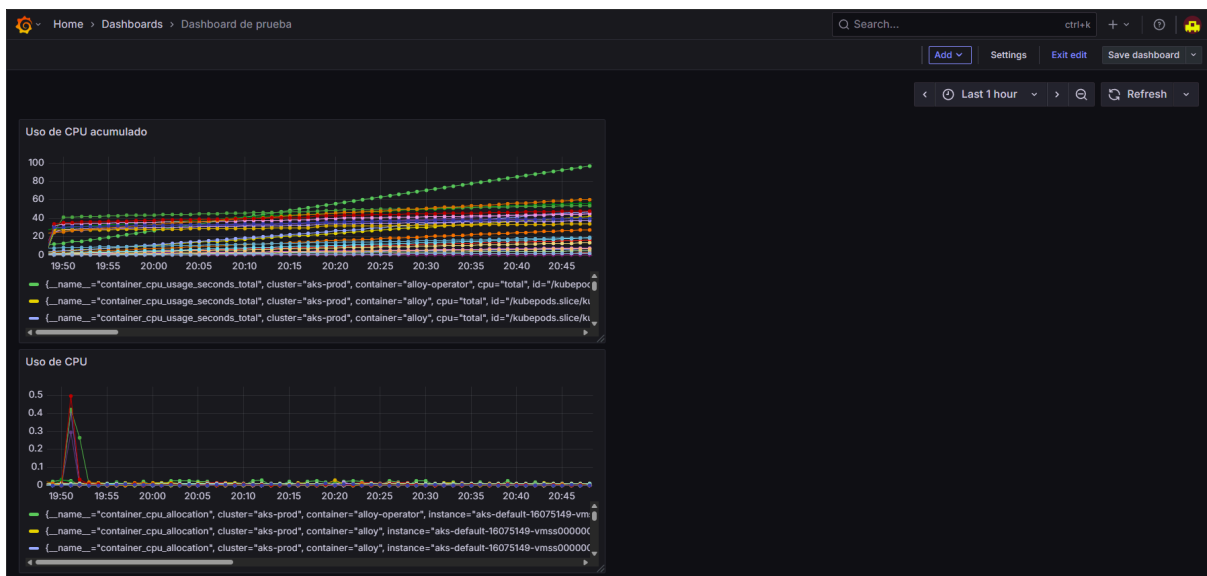
- En cuanto a health checks y readiness/liveness probes todos los pods los traen configurados ya que sino la tarea encargada de desplegar los k8s falla al no ser capaz de establecer si la aplicación se encuentra en funcionamiento

```

readinessProbe:
  httpGet:
    path: /actuator/health
    port: 9296
  initialDelaySeconds: 60
  periodSeconds: 10
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 3
livenessProbe:
  httpGet:
    path: /actuator/health
    port: 9296
  initialDelaySeconds: 90
  periodSeconds: 30
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 3

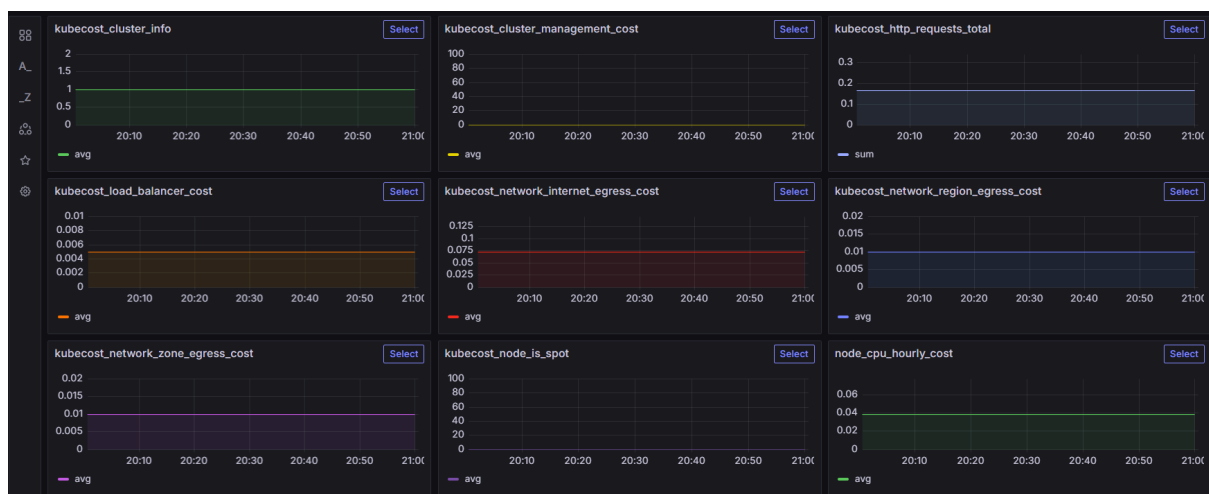
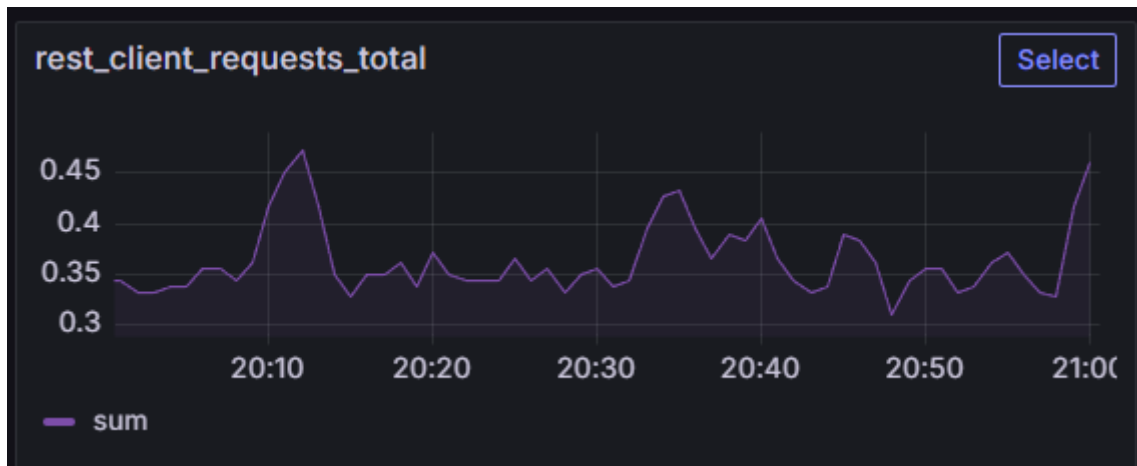
```

- Además se pueden crear dashboards en el propio grafana en base a lo que deseamos vigilar, como puede ser por cluster, namespace, o pod y extraer una métrica de relevancia, por ejemplo aquí se pueden observar dos paneles que hicimos del uso de CPU acumulado y el uso de CPU histórico en el namespace default, es decir sobre todos los pods que pertenecen a default



- No realizamos dashboards sobre un pod en específico ya que al ser un taller constantemente estamos bajando la infraestructura y desplegando de nuevo, por lo que el id del pod cambia y el dashboard dejaría de funcionar cuando se baje la infraestructura, mientras que los que hicimos que se enfocan en el default namespace ellos siempre van a funcionar sin importar cuantas veces se suba o baje la infraestructura. De todas formas en los videos se va a mostrar como crear un dashboard en base a un pod.

- En cuanto a métricas grafana ofrece desde el inicio una gran cantidad tanto técnicas como de negocio:



- Vemos métricas técnicas como las request y de negocio como las tipo kubecost que relacionan el uso de recursos con el gasto que generan. Aportando valor al negocio que las use.

## Costos:

Componente	Costo	Fuente
AKS	\$0,10 por clúster/hora	<a href="#">Precios: Azure Kubernetes Service (AKS)   Microsoft Azure</a>
VMs (Standard_DS2_v2)	\$0.146 por nodo/hora	<a href="#">DS2 v2 pricing and specs - Vantage</a>
Storage (Disco y Backend)	\$0.16 por GB en el SSD (Archivos de VM, etc) \$0.021 por GB tipo Blob (Backend)	<a href="#">Azure Files Pricing   Microsoft Azure</a> <a href="#">Azure Blob Storage pricing   Microsoft Azure</a>
Load Balancer (Asumimos que es por GB usados)	\$0.005 por GB	<a href="#">Pricing—Load Balancer   Microsoft Azure</a>

Ancho de banda (Asumiendo que están desplegados en la misma región)	Gratis	<a href="#">Pricing - Bandwidth   Microsoft Azure</a>
--	--------	---

### Configuración de un Agente (extra)

- Durante el proyecto nos quedamos sin minutos de ejecución para los parallel jobs por lo que tuvimos que configurar nuestro propio agente usando WSL y uno de nuestros equipos

The screenshot shows the 'Private projects' section of the Azure DevOps interface. It displays a 'Microsoft-hosted' agent with a 'Free tier' of '1 parallel job up to 1800 mins/mo'. A status bar indicates 'Currently 1801/1800 minutes are consumed' and provides a link to 'Purchase parallel jobs'. Below this, the 'Default' agent configuration is shown with tabs for 'Jobs', 'Agents', 'Details', 'Security', 'Approvals and checks', and 'Analytics'. The 'Agents' tab is active, showing a table with agent details.

Name	Last run	Current status	Agent version	Enabled
the-chamber ● Online	Just now	Idle	4.255.0	<input checked="" type="checkbox"/> On

### Videos:

### Patrones:

- <https://youtu.be/914fHtqeBrs>

### Despliegue y grafana:

- <https://youtu.be/qNvGGIkFE4>