Universidad Politécnica de Madrid

DESARROLLO DE UNA APLICACIÓN DE MICROSERVICIOS CON TÉCNICAS DEVOPS Y USO DE DATADOG PARA PROMOVER LA OBSERVABILIDAD

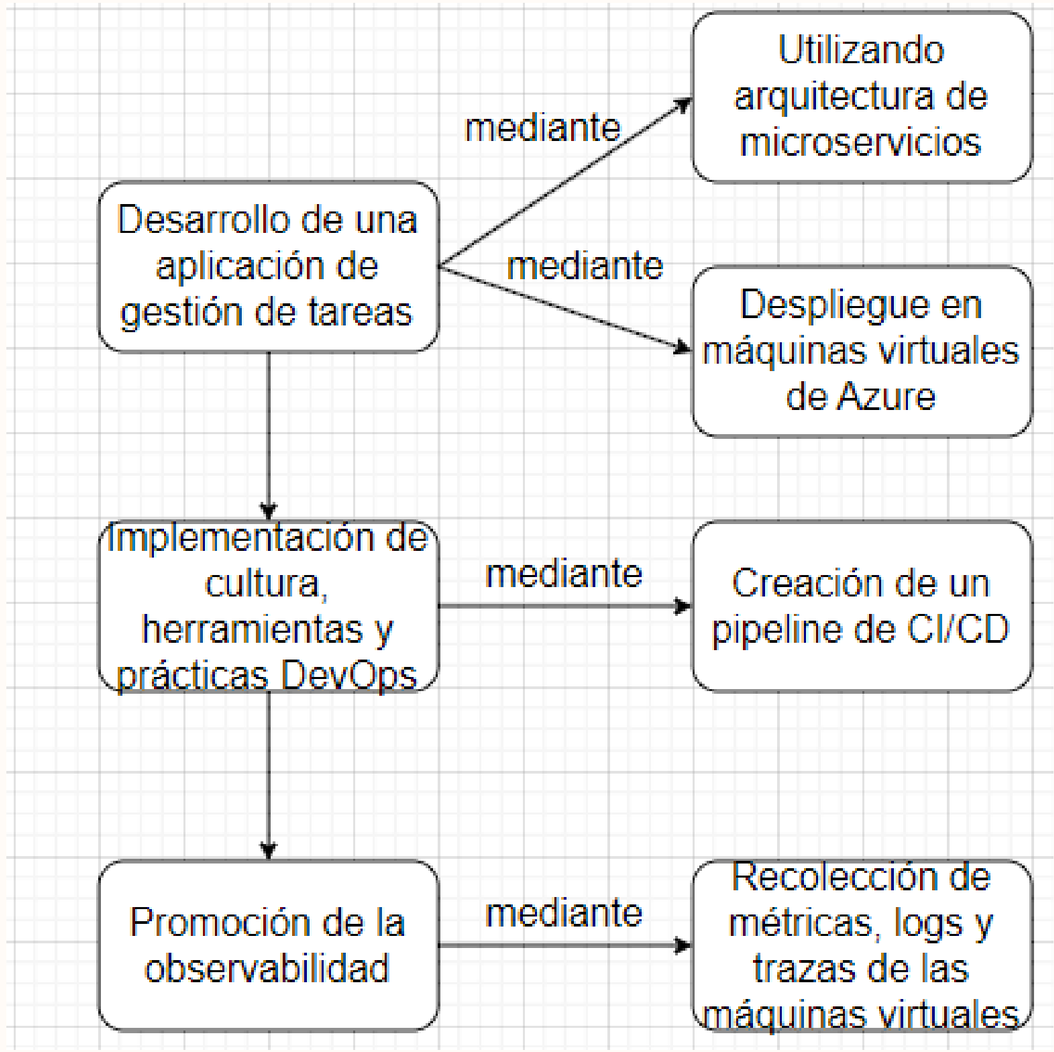
Presentado por Daniil Cebanu Muntean
Tutor: Jessica Diaz Fernandez

UPM

RESUMEN

Este proyecto de fin de grado consiste en:

- Desarrollo de un gestor de tareas empleando herramientas actuales y arquitectura de microservicios.
- Creación de un pipeline para aplicar cultura, prácticas y herramientas DevOps.
- Despliegue de la aplicación utilizando el servicio de Tomcat.
- Promoción de la observabilidad haciendo uso de Datadog.



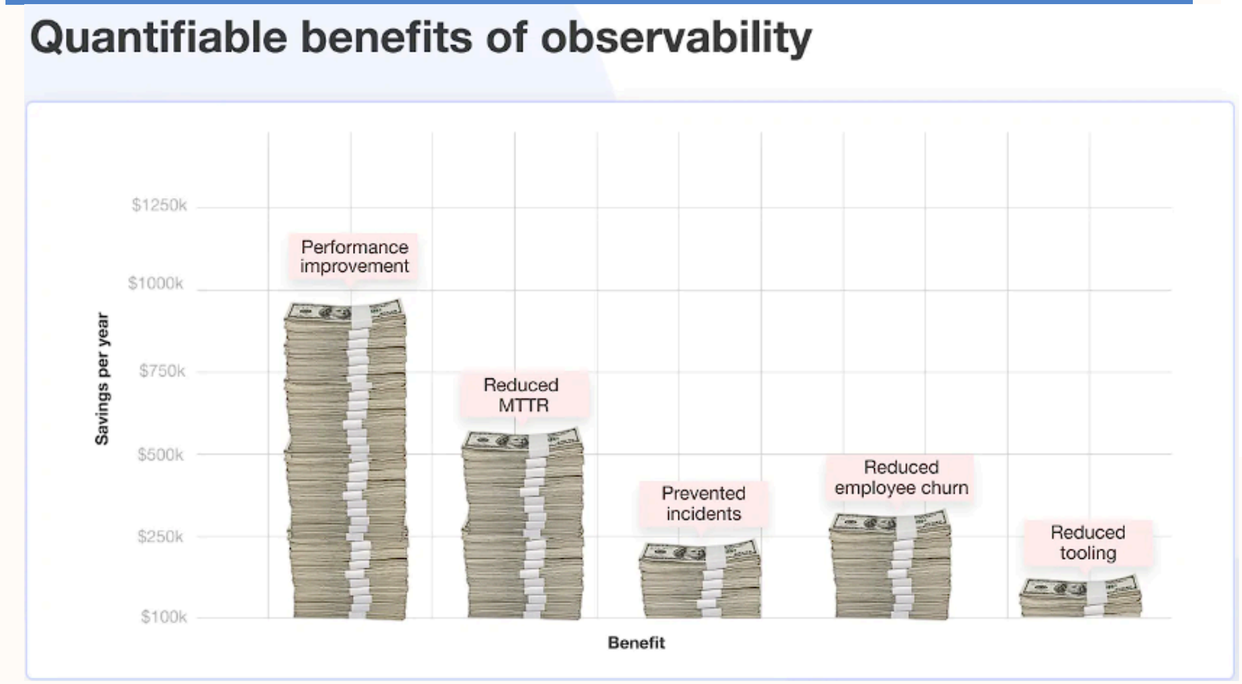
Fuente propia

UPM

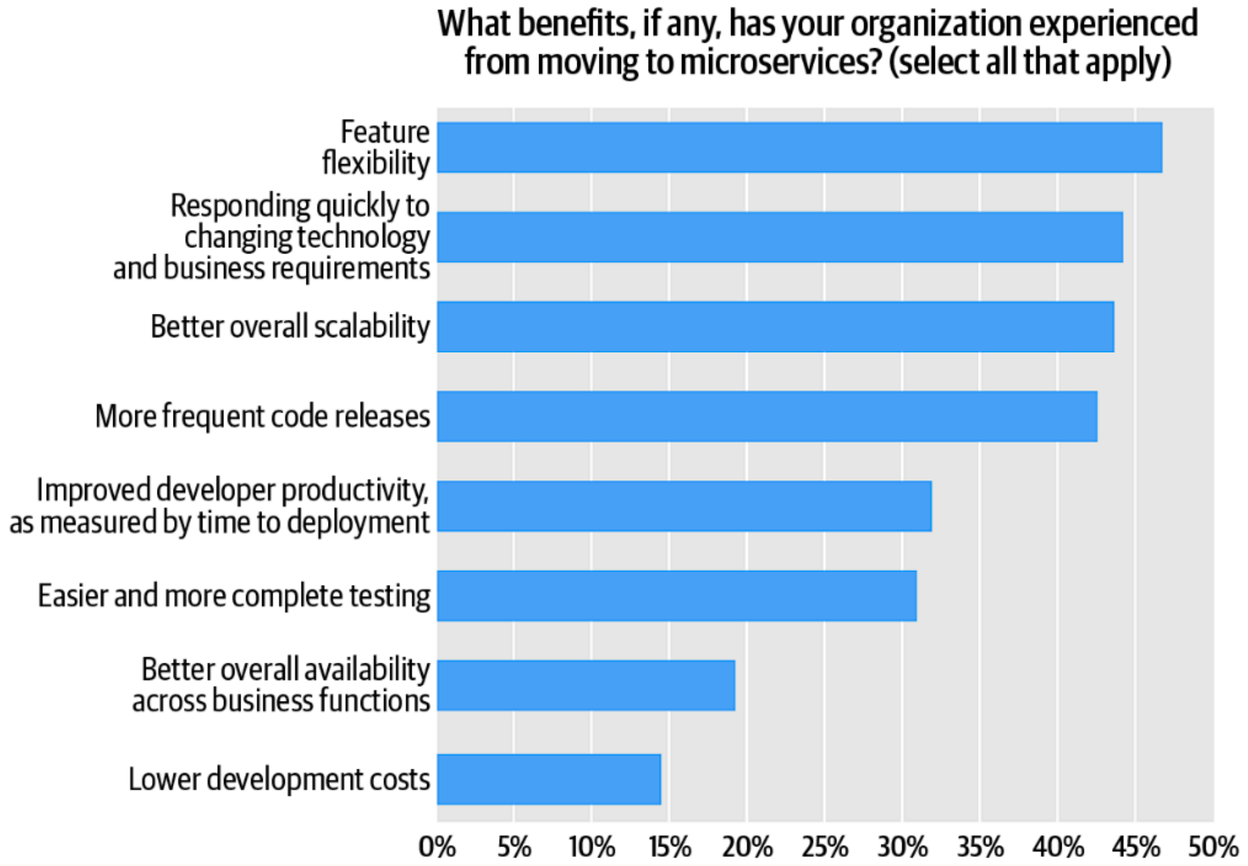
ÍNDICE

1. INTRODUCCIÓN

- La arquitectura de microservicios se adopta y usa cada vez más, indicando que las ventajas que más notan las empresas son la flexibilidad, la rápida respuesta a cambios, mejor escalabilidad y entregas de código más rápidas.
- La implementación de observabilidad en entornos cloud reduce gastos a las empresas.



Fuente: simform



Fuente: oreally

UPM

2. PROBLEMA A RESOLVER

La motivación de este proyecto es crear una aplicación que use la arquitectura de microservicios y aplicarle herramientas y prácticas de la cultura DevOps junto a observabilidad, ya que esta cultura y esta habilidad de recolectar datos son beneficiosos para la satisfacción de los clientes.

1
Integración del desarrollo software con la arquitectura de microservicios

2
Desarrollo de un gestor de tareas que cumple con los estándares de desarrollo sostenible de la ONU

3
Uso de estándares que cumplan con los requisitos de calidad y seguridad de un proyecto software

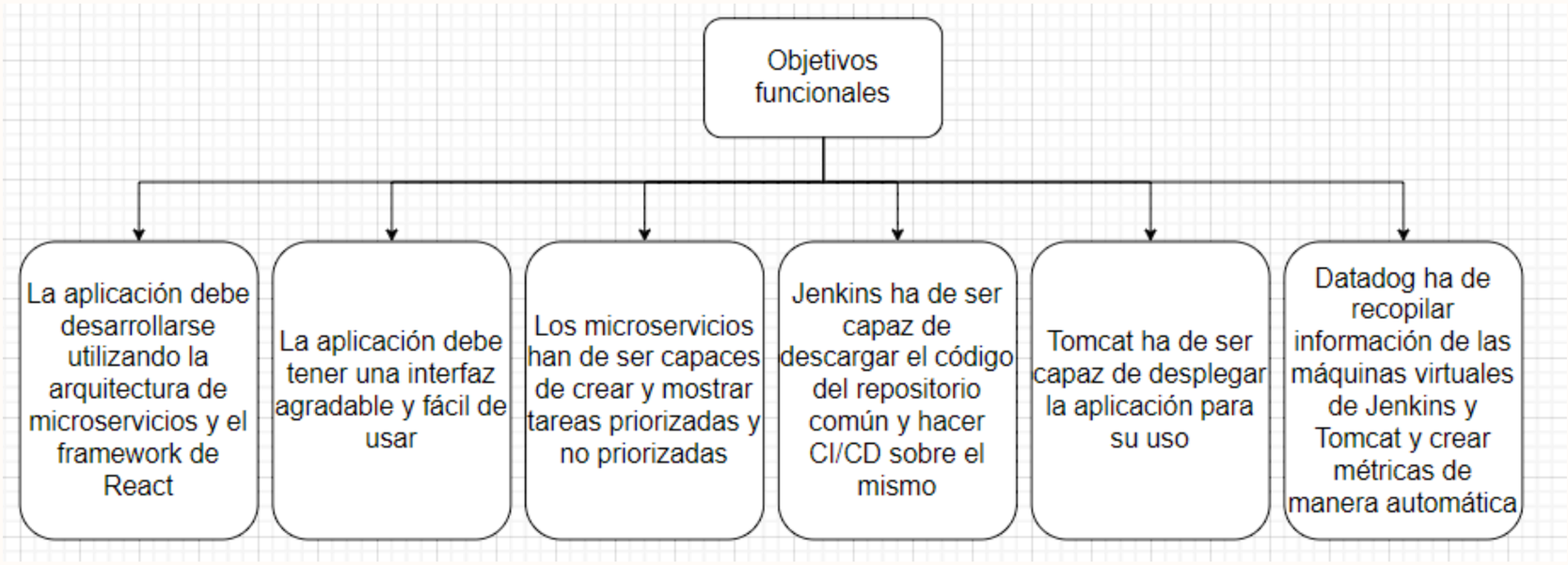
4
Implementación de cultura DevOps para realizar Continuous Integracion (CI) y Continuous Delivery (CD) sobre la aplicación

5
Uso de plataformas cloud para el desarrollo y despliegue de la aplicación mediante el uso de IaC

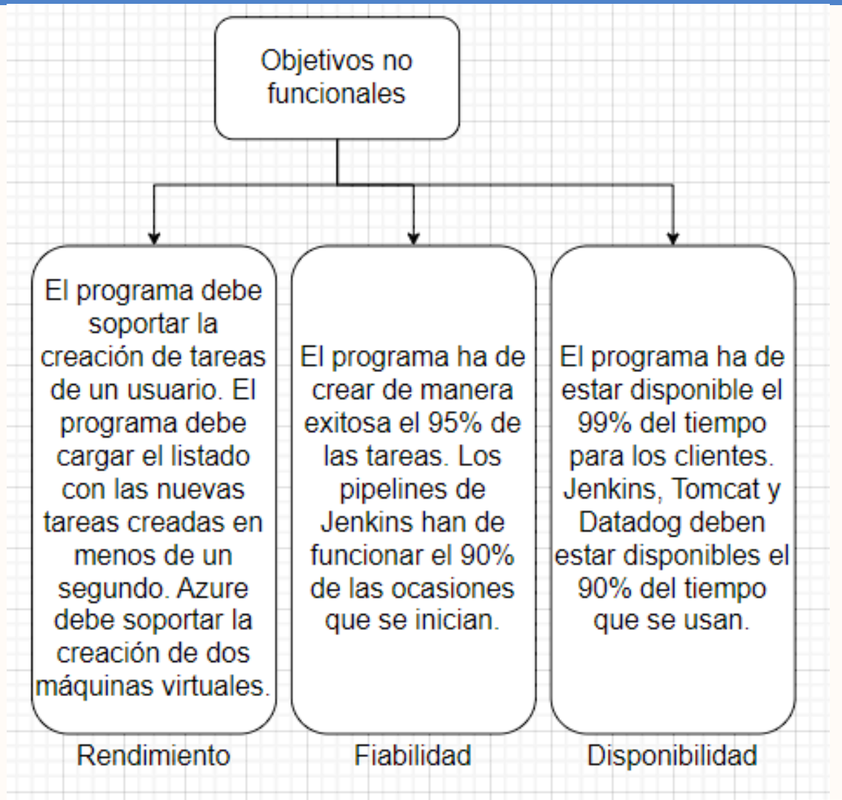
6
Aplicación de la observabilidad usando herramientas actuales

3. OBJETIVOS

El objetivo general del proyecto es fomentar el uso de las herramientas y prácticas de la cultura DevOps y observabilidad mediante el desarrollo de un caso práctico de una aplicación de microservicios para conseguir entregas de código continuas y de calidad.



Fuente propia

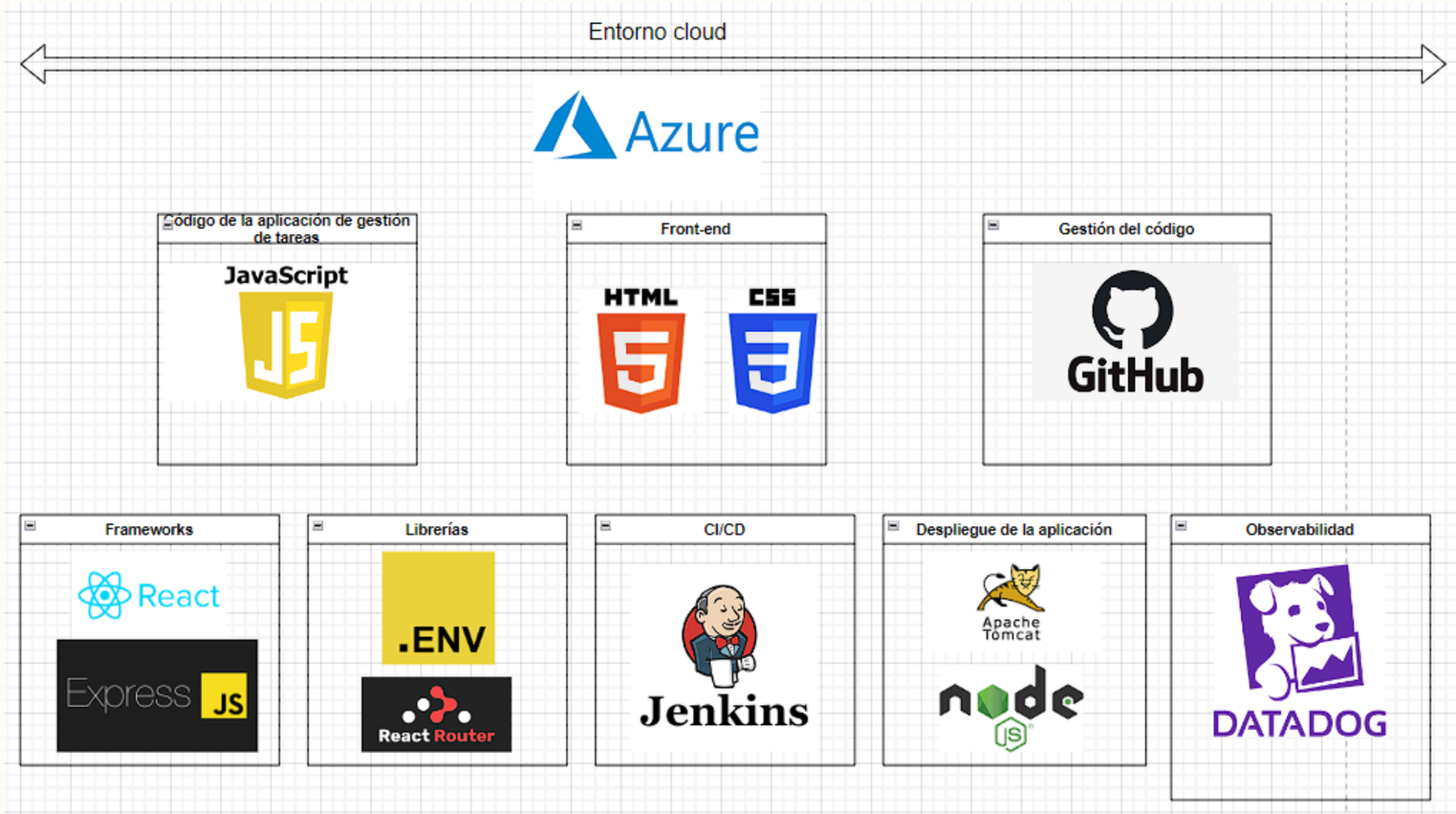


Fuente propia

UPM

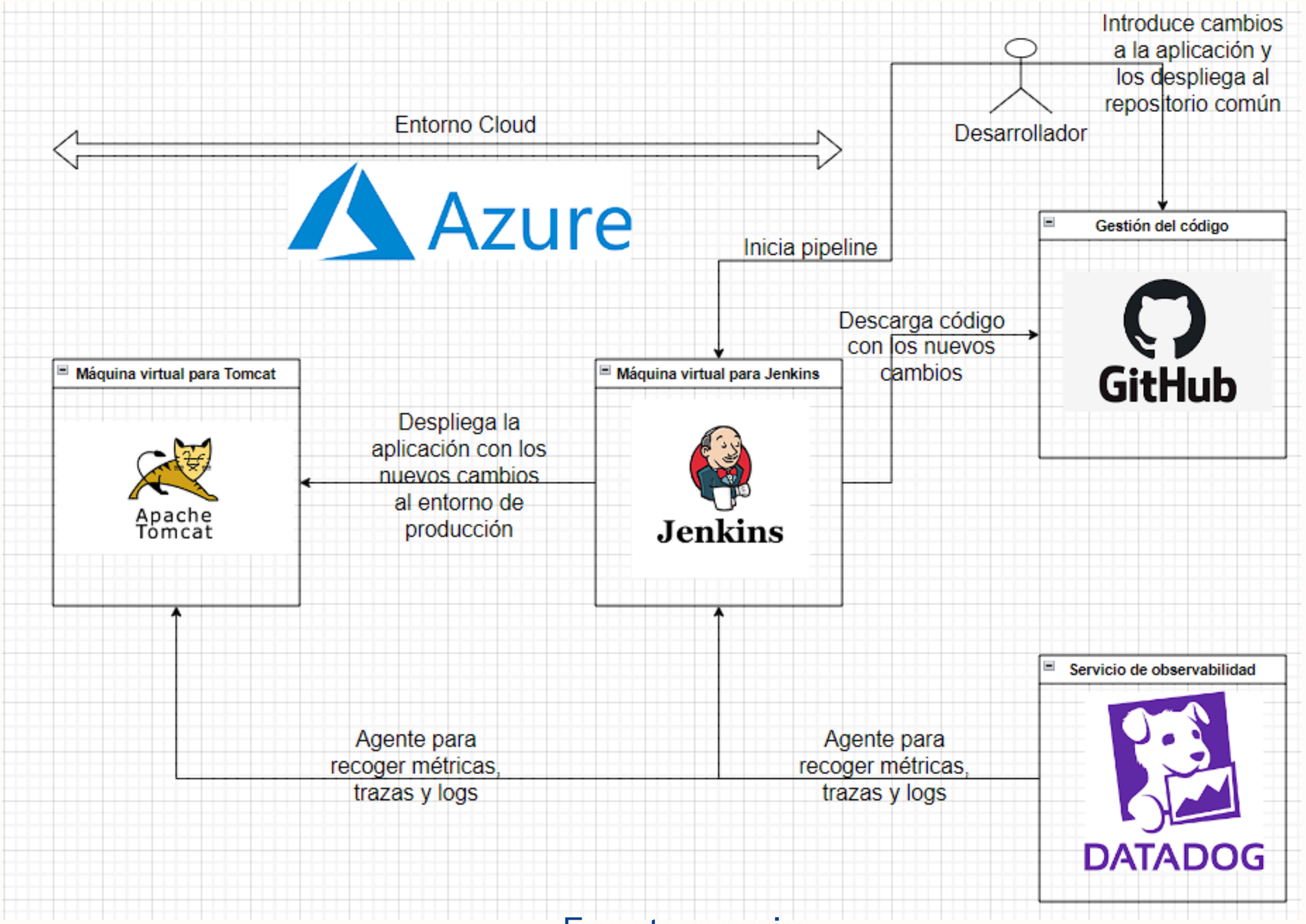
4. HERRAMIENTAS UTILIZADAS

- Visual Studio Code
- JavaScript
- HTML y CSS
- Node.js
- React
- Github
- Azure
- Jenkins
- Tomcat
- Datadog



Fuente propia

4. HERRAMIENTAS UTILIZADAS



Fuente propia

UPM

5. PROPUESTA DE SOLUCIÓN

Para cumplir con los objetivos establecidos, se desarrolla un gestor de tareas con las siguientes características:

- Se utiliza **npm** para instalar las dependencias necesarias especificadas en los ficheros “package.json”.

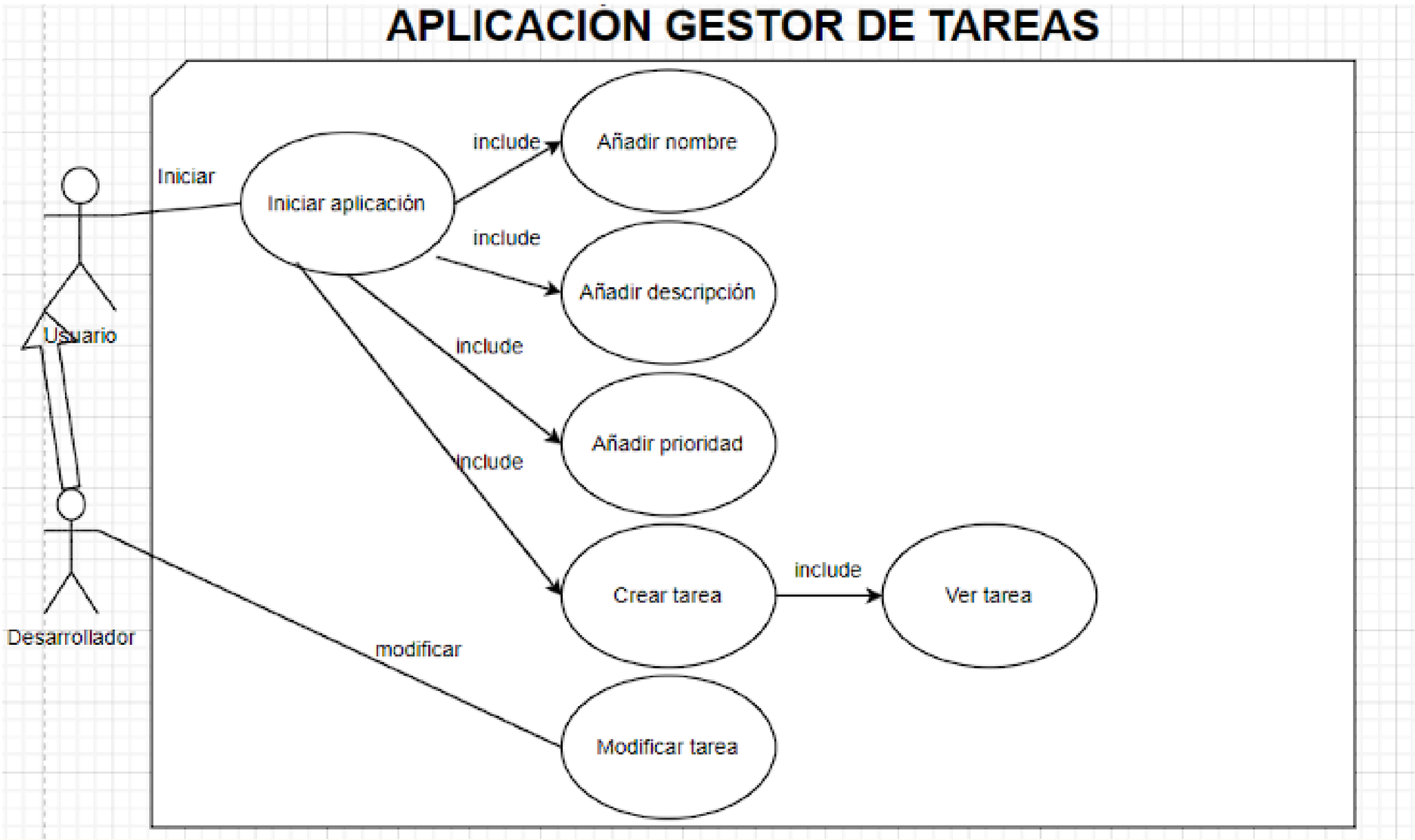
- Creación de máquinas virtuales en Azure, una para alojar al servicio de Jenkins y otra para alojar el servicio de Tomcat. Es necesario gestionar las redes para que se permita el tráfico de paquetes entre ambas máquinas.

- Se desarrollan dos microservicios, uno se encarga de tareas con prioridad y otro para tareas sin prioridad. Esta separación en microservicios se debe a factores como el principio de separación de intereses, la optimización del rendimiento, la escalabilidad

- Todas las tareas permiten mejorar la gestión de metas y del tiempo, por lo que se cumplen con los objetivos 3, 8 y 12 del desarrollo sostenible propuesto por la ONU.

UPM

5. PROPUESTA DE SOLUCIÓN



Fuente propia

6. MICROSERVICIOS

El microservicio que gestiona las tareas priorizadas, **priority-service,** tiene las siguientes funciones:

- **generateId()** permite generar ids únicos para las tareas. Utiliza el módulo crypto de node.js para generarlos.
- **handleError()** permite gestionar los errores que ocurran en los endpoints de la API del microservicio.
- **createTask()** permite crear tareas priorizadas mientras se le pase por parámetro un título, una descripción y una prioridad.
- **getTasks()** permite recoger todas las tareas del usuario.
- **updateTask()** permite actualizar una tarea mediante su id.

```
const PORT = process.env.PRIORITY_SERVICE_PORT || 3002;
app.listen(PORT, () => {
  console.log(`Priority management microservice running on port ${PORT}`);
});
```

Fuente propia

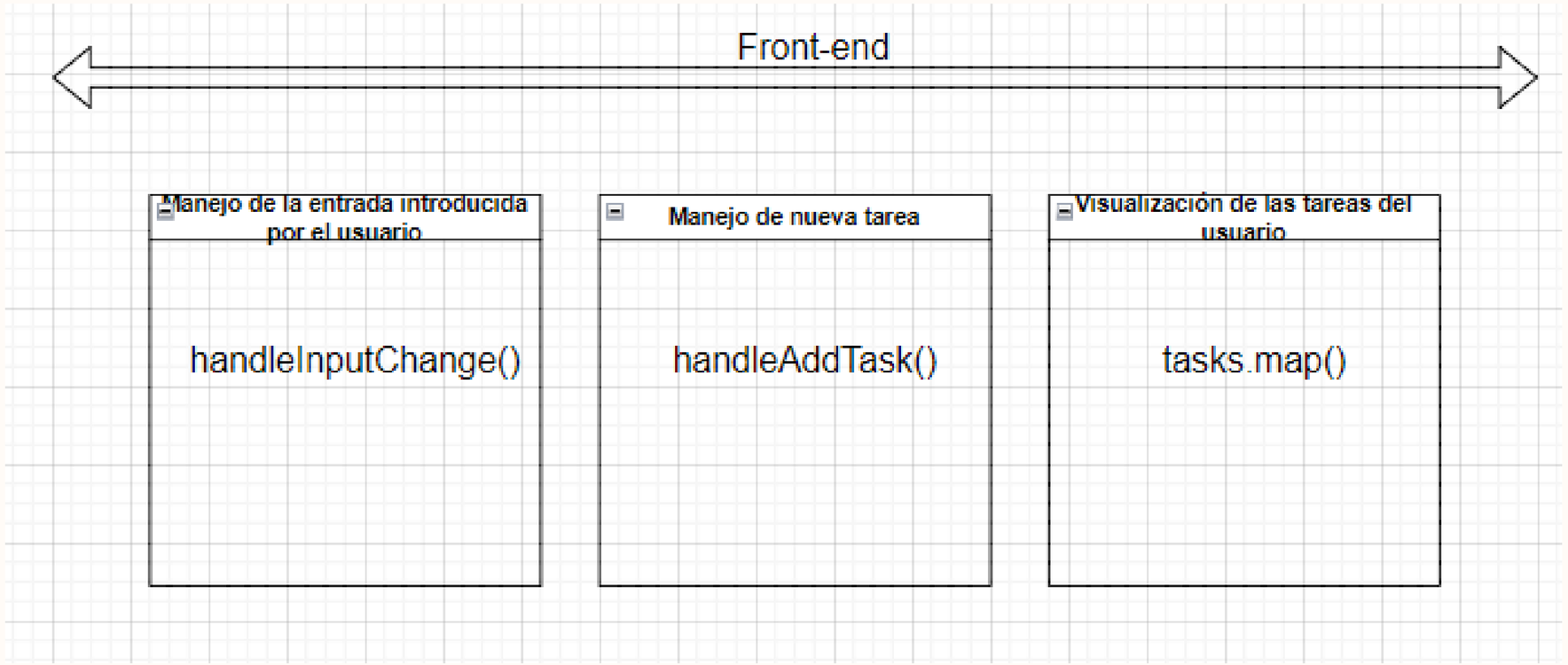
El microservicio que gestiona las tareas no priorizadas, **task-service**, tiene las siguientes funciones distintivas del anterior microservicio:

- **createTask()** permite crear tareas sin necesidad de requerir una prioridad.
- **updateTask()** actualiza una tarea sin necesidad de modificar la prioridad.

```
const PORT = process.env.TASK_SERVICE_PORT || 3001;
app.listen(PORT, () => {
  console.log(`Task management microservice running on port ${PORT}`);
});
```

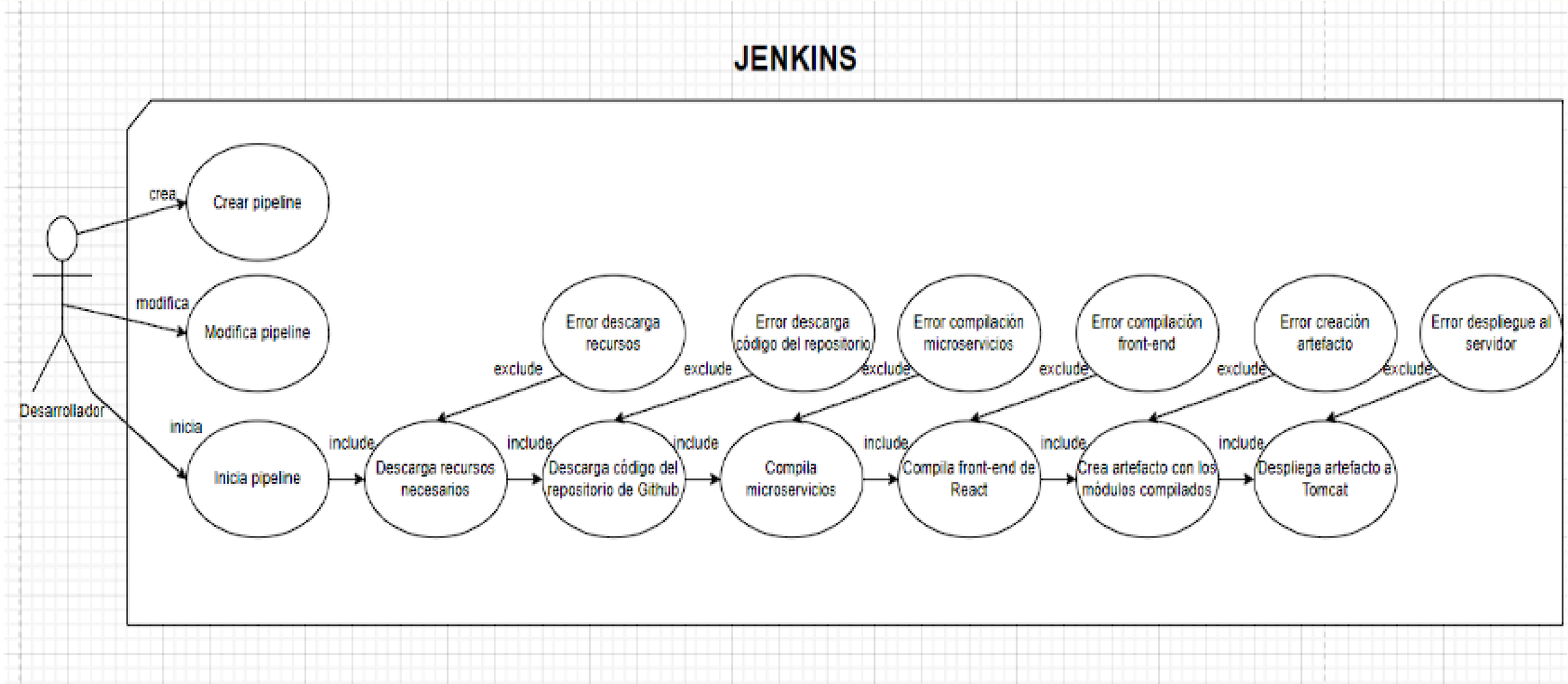
Fuente propia

7. FRONT-END



Fuente propia

8. PIPELINE DE CI/CD



Fuente propia

8. PIPELINE DE CI/CD

```
steps {
    nodejs(nodeJSInstallationName: 'Node2230'){
        dir('webapp') {
            sh 'npm install'
            sh 'npm run build'
            stash name: 'webapp', includes: 'build/**'
        }
    }
}
post {
    failure {
        script {
            env.STAGE_RESULT = 'FAILURE'
        }
    }
}
```

Fuente propia

```
when {
    equals expected: 'SUCCESS', actual: env.STAGE_RESULT
}
```

Fuente propia

```
dir('webapp') {
    unstash 'webapp'
}
```

Fuente propia

```
steps {
    withCredentials([usernamePassword(credentialsId: "${TOMCAT_CREDENTIALS}", usernameVariable: 'TOMCAT_USER', passwordVar
        sh "ssh ${TOMCAT_USER}@${TOMCAT_HOST} mkdir -p ${TOMCAT_DEPLOY_DIR}/priority-service"
        sh "ssh ${TOMCAT_USER}@${TOMCAT_HOST} mkdir -p ${TOMCAT_DEPLOY_DIR}/task-service"
        sh "ssh ${TOMCAT_USER}@${TOMCAT_HOST} mkdir -p ${TOMCAT_DEPLOY_DIR}/webapp"
        sh "scp -r priority-service ${TOMCAT_USER}@${TOMCAT_HOST}:${TOMCAT_DEPLOY_DIR}"
        sh "scp -r task-service ${TOMCAT_USER}@${TOMCAT_HOST}:${TOMCAT_DEPLOY_DIR}"
        sh "scp -r webapp/build ${TOMCAT_USER}@${TOMCAT_HOST}:${TOMCAT_DEPLOY_DIR}/webapp"
    }
}
```

Fuente propia

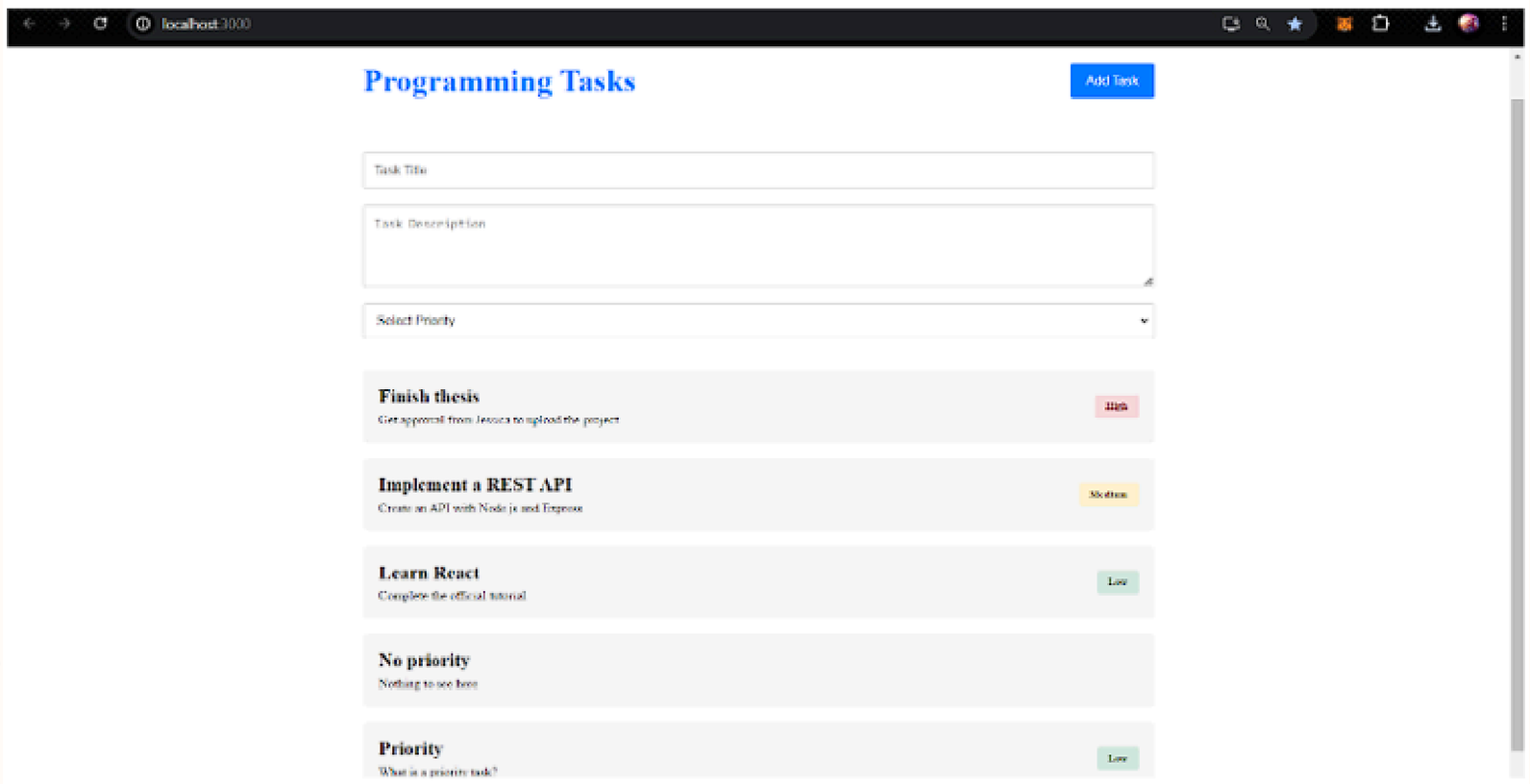
9. RESULTADO

1. Se ha desarrollado una aplicación de arquitectura de microservicios con front-end con el framework de React.

2. Utilizando el framework de React la interfaz de usuario es agradable y fácil de usar.

3. Los microservicios son capaces de gestionar tanto tareas priorizadas como sin priorizar.

4. Mediante la creación de un pipeline de CI/CD con Jenkins se ha conseguido descargar código de Github y desplegarlo de forma automática.



Fuente propia

9. RESULTADO

5

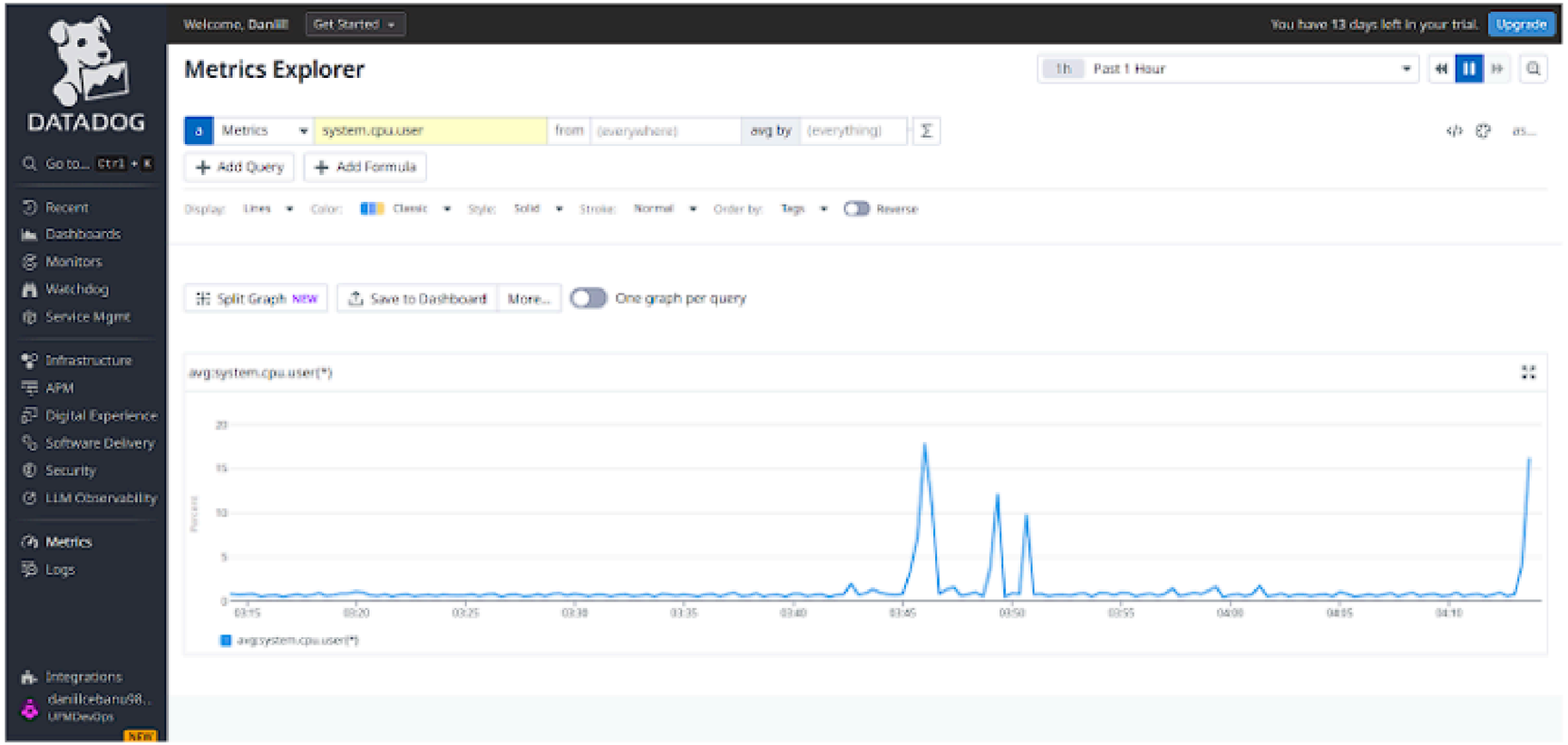
Se utiliza Tomcat para el despliegue de la aplicación en una máquina virtual de Azure.

6

Se usa Datadog para implementar sus agentes en las máquinas virtuales de Azure y aplicar observabilidad.

7

Se cumple con los requisitos no funcionales de rendimiento, fiabilidad y disponibilidad.



Fuente propia

UPM

10. IMPACTO SOCIAL Y MEDIOAMBIENTAL

Impacto social:

- Fomentar el desarrollo sostenible mediante el uso de la automatización para la gestión y el despliegue de una aplicación desarrollada con la arquitectura de microservicios.
- Atraer a usuarios que estén interesados en gestionar sus tareas y tiempo mediante una aplicación fácil de usar y con una UI atractiva.
- Fiabilidad y seguridad a los usuarios por el uso de tecnología cloud.
- Compromiso social con el desarrollo sostenible debido al uso de herramientas y técnicas actuales que fomentan estas ideas.

Impacto medioambiental:

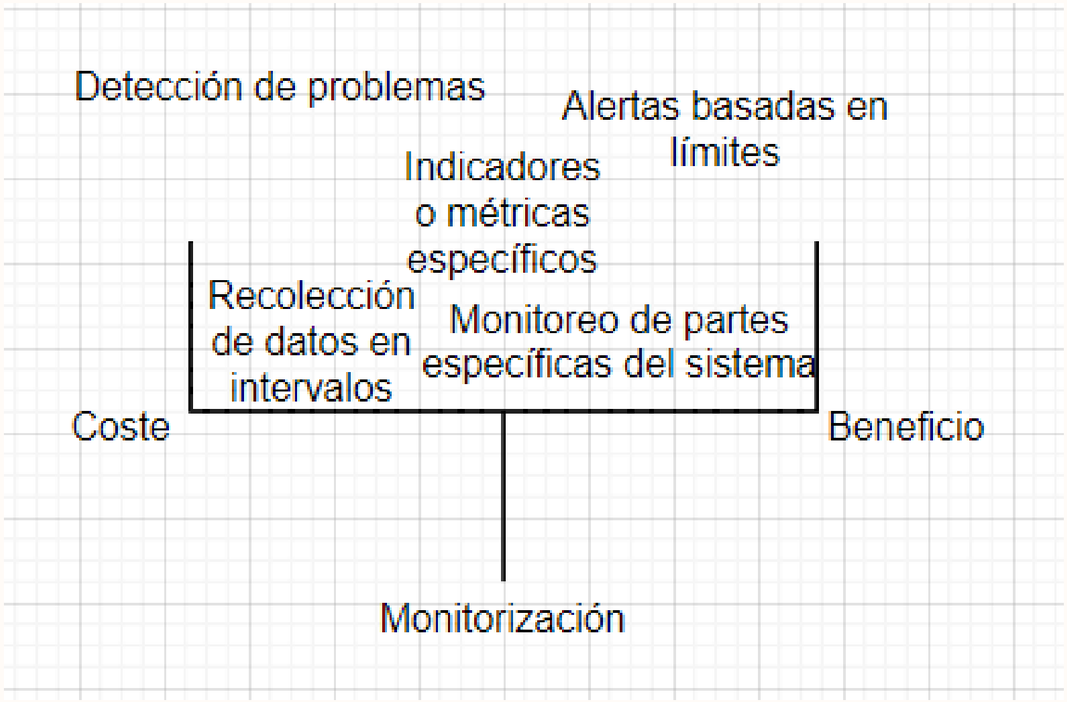
- Eliminar ineficiencias en servidores locales, haciendo uso de tecnología cloud.
- Mayor productividad y ahorro mediante el uso de herramientas y técnicas actuales.
- Cumplimiento con los estándares de la agenda 2030 mediante la promoción del uso de una aplicación de gestión de tareas y mediante el uso de automatización y observabilidad para eliminar ineficiencias tanto en los servidores como en los servicios.

11. CONCLUSIÓN

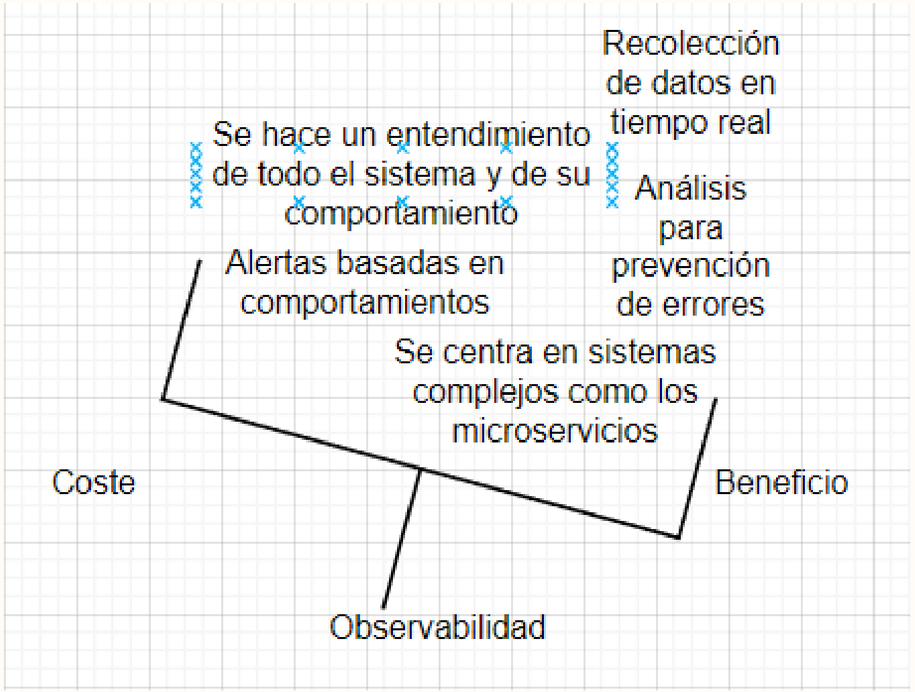
Se deducen las siguientes conclusiones tras las realización del proyecto:

- Trasladar el proyecto a un entorno de producción real requiere de una inversión modesta debido al uso de tecnología cloud y Datadog.
- El éxito de un proyecto como el de estas características require de aplicar los objetivos de observabilidad con el fin de obtener todos los datos necesarios para poder arreglar todos los problemas.
- Se ha demostrado que el uso de Datadog como herramienta para recolectar todo tipo de datos es eficiente y sirve para resolver los problemas encontrados.
- Se crea una aplicación de microservicios y con React capaz de ser gestionada por un pipeline de CI/CD en Jenkins y desplegado con Tomcat, ambos en servidores distintos de Azure. Además, Datadog es capaz de recolectar toda la información necesaria para poder resolver problemas.
- Se genera un ecosistema donde se usan las mejores herramientas y técnicas y se fomenta la adopción de cloud y servicios compatibles con cloud. Observabilidad ayuda a resolver problemas en tecnologías cloud cada vez más complejas.

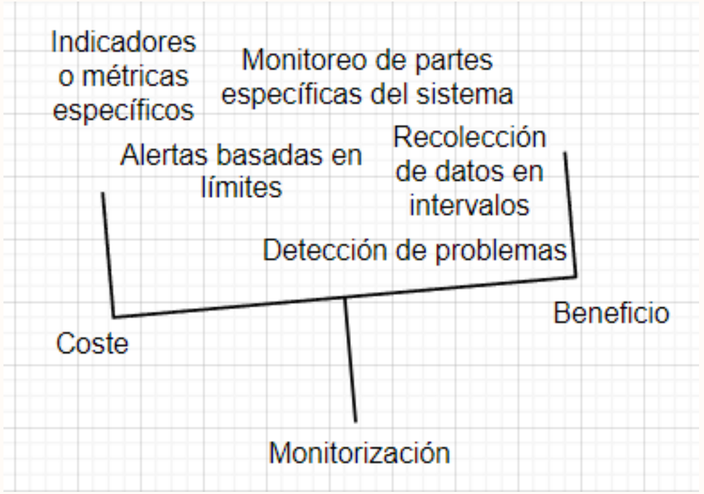
11. CONCLUSIÓN



Fuente propia



Fuente propia



Fuente propia

MUCHAS GRACIAS

Presentado por Daniil Cebanu Muntean
Universidad Politécnica de Madrid