Entornos de desarrollo en la nube para alumnado.

Javier Colmeiro Aznar
Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación
Administración de redes y sistemas operativos

Mario Prieto Vega
David Bañeres Besora
Javier Panadero Martínez
Montse Serra Vizern

Junio 2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Entornos de desarrollo en la nube para alumnado.*
Nombre del autor:	*Javier Colmeiro Aznar*
Nombre del consultor/a:	*Mario Prieto Vega*
Nombre del PRA:	*David Bañeres Besora, Javier Panadero Martínez, Montse Serra Vizern*
Fecha de entrega (mm/aaaa):	05/2023
Titulación:	*Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación*
Área del Trabajo Final:	*Administración de redes y sistemas operativos*
Idioma del trabajo:	*Castellano*
Palabras clave	*Desarrollo remoto, Docker, CDE.*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Durante los estudios universitarios hay asignaturas que requieren un laboratorio de prácticas donde comprobar de forma experimental lo estudiado. Para esto, hacen uso de laboratorios remotos o máquinas virtuales pre-configuradas que, aparentemente, permiten que todos los alumnos dispongan de las mismas herramientas. No obstante, en la práctica, este tipo de laboratorios necesitan tecnologías de virtualización instaladas en la máquina del usuario y de cierto grado de recursos y conocimientos que hacen que su uso sea un proceso tedioso y complicado.

Teniendo todo esto en cuenta, se plantea estudiar las diferentes soluciones de código libre y privado del mercado centradas en desarrollo de código remoto en cloud privada o pública tipo PaaS y elegir cual sería la ideal para implantar en un entorno educacional. Este tipo de tecnología, basada habitualmente en sistemas de contenedores Docker, permiten utilizar unos ficheros de configuración que describen las características y herramientas que necesita el servicio y pueden ser desplegados de forma instantánea para su uso.

De esta manera, se podría facilitar el acceso a este tipo de laboratorios y aislar al alumno de la parte de requisitos técnicos y de configuración para ejecutarlos así como facilitar las tareas de corrección de los profesores mediante la automatización del envío de resultados y su proceso.

Abstract (in English, 250 words or less):

At the degree there are subjects that require a remote lab to practically test what we have learned. For that, it is customary to use server remote labs or virtual machines pre-configured that apparently allows every student to have same set of tools. Nevertheless, these solutions need virtualization technologies installed on personal computers and some specific hardware resources and knowledge that make them use something tedious and complicated.

Having all this in mind, a big research is needed about current opensource, and private solutions centred in cloud development environments in private clouds or public clouds with PaaS and choose one that would be a good match for an educational institution. This kind of technology, generally based in Docker allows to have configuration files that manage all needs and allows its use almost instantaneously.

With this, we could facilitate access to this kind of remote labs and isolate students from technical and configuration requirements and to ease teacher's correction process with automation.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del trabajo

Durante el grado, hay muchas asignaturas de programación que podrían beneficiarse de tener un laboratorio de prácticas donde comprobar de forma experimental lo estudiado en la parte teórica. Para ello, una opción habitual es la de hacer uso de laboratorios remotos basados en máquinas virtuales pre-configuradas que, aparentemente, permiten que todo el alumnado disponga de las mismas herramientas de desarrollo.

No obstante, en la práctica, este tipo de laboratorios sufren de necesitar por parte del alumno de ordenadores con tecnologías de virtualización disponibles y cierto grado de recursos de *hardware* y conocimientos de informática que hacen de su uso un proceso tedioso y mas complicado de lo habitual.

Por otra parte, estos entornos pueden sufrir errores que no son fácilmente reproducibles en otro equipo y solucionar estos problemas lleva tiempo y conocimientos que el alumno normalmente no dispone.

1.2 Objetivos del trabajo

Se pretende realizar un estudio global de las capacidades técnicas de las soluciones actuales desplegadas en nube privada o pública en modalidad *PaaS* de contenedores para desarrollo de aplicaciones de software como *Gitpod, GitHub CodeSpaces, JetBrains Spaces, Eclipse Che, Redhat Dev Spaces*, etc. Con esto se debe detallar las tecnologías empleadas, los requisitos de infraestructura así como los costes asociados al uso de la infraestructura y de licencias de uso o de pago por uso de hora de computación.

Seguidamente, se desarrollará una configuración básica funcional de cada uno de los sistemas evaluados y se desplegará un prototipo funcional que simulará algún tipo de entorno de desarrollo remoto para valorar sus tiempos de carga, de despliegue, latencia, versatilidad de uso, y facilidad de uso.

Finalmente, se desarrollará una comparativa de estas tecnologías y se seleccionará la que mejor se adaptaría a un entorno educacional por motivos de características, facilidad, licencias, costes y usabilidad.

No entraría dentro del alcance del proyecto aunque se necesita de ellos para poder realizarlo :

- Disponer o desarrollar una infraestructura *cloud* privada.
- Disponer o desarrollar un sistema de evaluación de resultados de laboratorios automático.

1.3 Enfoque y método seguido

Para el desarrollo de este TFG se ha optado por realizar una enfoque de investigación y evaluación práctica de las múltiples soluciones que existen en el ámbito de los entornos de desarrollo remoto en el mercado actual. Así pues, el método seleccionado para este trabajo involucra una combinación de investigación teórica y aplicación práctica.

En primer lugar, se centra la investigación teórica en la búsqueda y el estudio de las diversas soluciones tanto de código abierto como privativas así como de despliegue en nube pública como privada. De esta manera se pretende obtener una visión completa del ecosistema actual de las soluciones identificando las características, beneficios y desventajas de cada una de ellas, así como las que pueden adaptarse a las necesidades del desarrollo de código en una organización educacional.

Paralelamente a la primera, tras encontrar una solución se procede a una valoración práctica inicial de la solución usando, normalmente, las propias versiones de evaluación gratuita para conseguir una idea general de las capacidades de la herramienta, facilidad de uso y rendimiento. Este proceso de pruebas debería ayudar a identificar la solución que quiere desplegarse en la infraestructura propia elegida.

Finalmente, tras realizar una comparativa exhaustiva de todas las soluciones estudiadas y evaluadas se selecciona una que, aparentemente, supondrá una buena elección para las necesidades de una entidad educacional. Con este enfoque se espera adquirir un conocimiento amplio sobre el mercado actual de una tecnología nueva y en proceso de aceptación inicial y con un abanico de soluciones y posibilidades enorme.

1.4 Planificación del trabajo

A continuación se adjunta un diagrama de Gantt con las fases de desarrollo del proyecto.

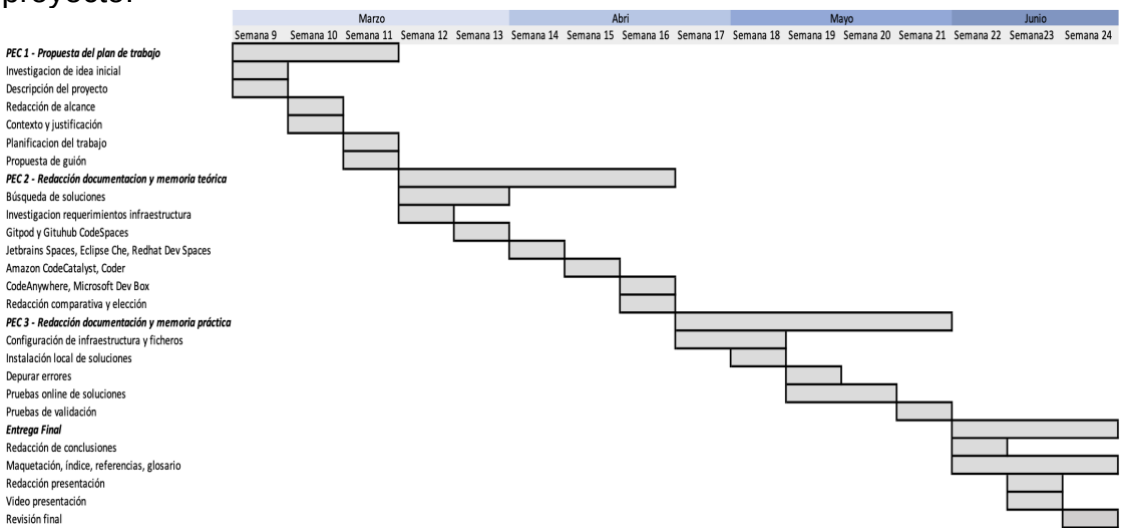


Ilustración 1. Diagrama de Gantt de la planificación del trabajo.

La planificación por semanas quedaría de la siguiente manera.

Partimos de la Semana 9 del año 2023, empezando el 1 de marzo y nos extendemos hasta la semana 24 del año 2023 siendo la fecha tope 18 de Junio.

- **Semana 9. Investigación de idea inicial y descripción del proyecto.**
  Investigación inicial de la situación actual de los entornos de desarrollo haciendo hincapié en los que usan tecnologías de despliegue remoto. Redación inicial de la descripción del proyecto, contexto y justificación.

- **Semana 10. Redacción de alcance, contexto y justificación.**
  Redación inicial del alcance, descripción del proyecto, contexto y justificación.

- **Semana 11. Planificación del trabajo y propuesta de guion.**
  Redacción de un guion inicial a seguir y planificación del trabajo en semanas en base a este.

- **Semana 12. Búsqueda de soluciones e investigación de requerimientos de infraestructura.**
  Búsqueda especifica de todas las soluciones que ofrecen entornos de desarrollo remotos para redactar la lista de soluciones a probar e investigar. Documentar el tipo de infraestructura que requieren para identificar cuales se adaptan mejor a un servidor privado o cloud privada.

- **Semana 13. Búsqueda de soluciones e investigación y pruebas con Gitpod y GitHub CodeSpaces.**
  Documentación sobre estas soluciones y pruebas de uso, funcionamiento y rendimiento. Investigación sobre requerimientos de infraestructura así como de licencias y precio.

- **Semana 14. Investigación y pruebas con Jetbrains Spaces, Eclipse CHE y Redhat Dev Spaces.**
  Documentación sobre estas soluciones y pruebas de uso, funcionamiento y rendimiento. Investigación sobre requerimientos de infraestructura así como de licencias y precio.

- **Semana 15. Investigación y pruebas con Amazon CodeCataltst y Coder.**
  Documentación sobre estas soluciones y pruebas de uso, funcionamiento y rendimiento. Investigación sobre requerimientos de infraestructura así como de licencias y precio.

- **Semana 16. Investigación y pruebas con CodeAnywhere y Microsoft Dev Box. Redacción de comparativa y elección.**
  Documentación sobre estas soluciones y pruebas de uso, funcionamiento y rendimiento. Investigación sobre requerimientos de infraestructura así como de licencias y precio.

- **Semana 17. Configuración de la infraestructura y ficheros.**
  Configurar una infraestructura base para poder desplegar la solución evaluadas en nube privada y basada en un servidor baremetal con una distribución debían 11. Instalación de Proxmox y creación de linux container para el despliegue de la solución.

- **Semana 18. Configuración de la infraestructura e instalación local de soluciones**
  Continuación de la instalación y configuración de la infraestructura así como de la instalación de la plataforma de Coder para su testeo.

- **Semana 19. Solución de errores en la infraestructura y pruebas online de soluciones PaaS.**
  Investigación y solución de errores en la infraestructura, fallos, incompatibilidad de paquetería, versiones de software.

- **Semana 20. Pruebas online de soluciones PaaS.**
  Investigación y documentación de creación de ficheros descriptores de entornos de desarrollo remotos como devfile.yaml, compose-dev.yaml, devcontainer.json, o Dockerfile lenguaje YAML, odo o json.

- **Semana 21. Pruebas de validación y buffer.**
  Pruebas que validan el rendimiento, facilidad de uso y las capacidades de la soluciones evaluadas y tiempo extra para realizar nuevas iteraciones o correcciones.

- **Semana 22. Redacción de conclusiones y maquetación del entregable.**
  Redacción de las conclusiones de la investigación general así como de la específica elegida y desplegada. Maquetación final del documento entregable confeccionando índices, referencias, glosarios, etc.

- **Semana 23. Maquetación del entregable, redacción de la presentación y grabación de video de la presentación.**
  Diseño y redacción de la presentación final a realizar así como del guion a seguir para la grabación del video. Continuación de la maquetación final del documento entregable.

- **Semana 24. Revisión general final.**
  Revisión general del documento entregable así como de las presentación y del documento de autoevaluación.

1.5 Breve sumario de productos obtenidos

En este proyecto no se ha obtenido un producto como tal ya que se ha realizado un estudio, evaluación y testeo sobre 10 plataformas distintas. En cada una de ellas se ha desplegado un prototipo funcional equivalente al que podría ser un producto final. Además se ofrece una comparativa de las soluciones, sus

características, licencias y precios que dan una visión completa sobre el mercado y ecosistema actual de soluciones, teniendo en cuenta que es una tecnología incipiente y en desarrollo y que cada mes aparecen nuevos actores.

1.6 Breve descripción de los otros capítulos de la memoria

Primeramente, en el capítulo 2, se detalla de forma teórica las características de la tecnología su evolución en el tiempo así como las ventajas e inconvenientes de cada una y hacia dónde se dirige el mercado actual.

A continuación, en el capítulo 3, se hace una preparación precisa sobre la infraestructura a utilizar, herramientas auxiliares y conocimientos necesarios para poder realizar el siguiente apartado.

Seguidamente, en el capítulo 4, se analizan y prueban las soluciones actuales mas utilizadas y se detalla sus características principales, precios y puntos fuertes y débiles.

En el capítulo 5 se realiza una comparativa para finalmente elegir la tecnología que más se adapta a los requisitos funcionales de una entidad educacional.

Por último, en el capítulo 6, se analizan las conclusiones obtenidas.

2. Entornos de desarrollo estandarizados

2.1 ¿Qué son?

Los entornos de desarrollo estandarizados son plataformas que permiten a los desarrolladores de software o estudiantes disponer del mismo juego de herramientas completo donde escribir, ejecutar y depurar código. Estos entornos están diseñados para garantizar que todos los usuarios trabajen en condiciones similares, evitando así las discrepancias que pueden surgir por las diferentes versiones de hardware, software y de herramientas.

Tradicionalmente, estas herramientas se descargan e instalan de forma individual por los usuarios y esto provoca que pueda existir diferencias en la experiencia de usuario respecto a la velocidad, facilidad de uso o errores inesperados. Además, al ser entornos independientes no se pueden actualizar de forma generalizada por parte del mantenedor del software ni añadir funcionalidades sin un proceso manual por parte del usuario.

2.2 Características.

Uniformidad. Proporcionan un entorno de trabajo uniforme y replicable para todos los desarrolladores. De esta manera todos disponen de las mismas herramientas en las mismas versiones y con las mismas configuraciones.

Reproducibilidad. Se puede replicar el entorno en diferentes máquinas con distintos componentes de hardware o software de sistema. Si se necesita cambiar de máquina de trabajo supone un proceso fácil y sencillo y reproducible en un tiempo reducido.

Automatización. Hacen uso de sistemas de aprovisionamiento o de automatización respecto a la instalación y configuración de las herramientas necesarias para el desarrollo de código. Esto limita la posibilidad de errores por parte del usuario.

Fiabilidad. Permiten replicar el mismo sistema en varios lugares por lo que ofrecen una confianza de que todo el mundo va a tener una experiencia similar. Por otra parte, esto también asegura que no aparezcan errores determinados por la configuración específica del desarrollador.

2.3 Evolución.

Ilustración 2. Evolución del despliegue de los entornos de desarrollo

Instalación manual. Inicialmente, cada desarrollador instala las herramientas necesarias en su hardware local. Además, tiene que configurar este según los requerimientos del código a crear o ejecutar y hay que tener en cuenta las versiones, compatibilidades entre ellas y solucionar los posibles errores que aparezcan.

Ventajas:

- Control total. El usuario tiene control total sobre las herramientas, las versiones que utiliza así como el hardware donde lo ejecuta y los recursos que le dedica.

- Offline. No requiere de conexión a internet

Inconvenientes:

- Tiempo. La instalación y configuración requieren de tiempo y dedicación por parte del usuario. También tienen que destinar recursos a su mantenimiento y puesta al día regular.

- Inconsistencias. Al no estar estandarizado, cada usuario puede tener distintas versiones instaladas y causar problemas de compatibilidad y de consistencia respecto al código desarrollado. Además la resolución de problemas se complica al no poder replicar con fiabilidad el sistema.

Scripts de aprovisionamiento. Estos llevan la automatización un paso adelante, definiendo en un código fuente, las herramientas y aplicaciones que se van a necesitar instalar, sus compatibilidades, versiones y adaptabilidad según el sistema operativo. Al ejecutarlo, el usuario instala de forma automática todas las necesidades para desarrollar el código y sus configuraciones específicas de forma desatendida.

Ventajas:

- Automatización. Los scripts de aprovisionamiento automatizan la instalación así como la configuración de las herramientas necesarias, ahorrando tiempo y abstrayendo al usuario del conocimiento específico para instalarlas.

- Reproducibilidad. Los scripts permiten replicar los entornos en varias máquinas de forma desatendida y rápida.

Inconvenientes:

- Conocimiento. Requieren de un conocimiento de paradigmas de automatización para su desarrollo, así como ser capaces de adaptarse los distintos hardware o sistemas operativos donde se vayan a ejecutar.

- Errores en el script. Al ser un proceso desatendido y secuencial, puede ocasionar que un problema temporal en el acceso a uno de los servicios o en que la configuración de este afecte al resultado final obteniendo un entorno erróneo o no funcional.

Máquinas virtuales pre-empaquetadas. Este fue el primer paso para homogenizar los entornos de desarrollo como un todo. En estos, se configura el sistema operativo, las herramientas de software y sus configuraciones y se empaquetan en forma imagen de máquina virtual. Esta se puede distribuir y ejecutar en distintos equipos. Proporciona un mayor grado de uniformidad pero requieren de una cantidad de recursos de hardware importante y su rendimiento suele ser malo, dando una experiencia de usuario sencilla pero lenta.

Ventajas:

- Uniformidad. Todos los usuarios disponen del mismo conjunto de herramientas y versiones.

- Reproducibilidad. El entorno puede ser replicado fácilmente tantas veces como se desee.

Inconvenientes:

- Recursos. Las máquinas virtuales requieren de una cantidad significativa de recursos de hardware así como que estos permitan ejecutar las tecnologías de virtualización.

- Actualizaciones. Al ser un sistema empaquetado cerrado, para actualizar cualquiera de las herramientas o realizar un cambio hay que recrear y redistribuir el sistema completo.

Entornos de desarrollo remotos (CDEs). La etapa más reciente de la evolución de los entornos de desarrollo remoto nos lleva a que las herramientas de desarrollo se encuentran instaladas en la nube. De esta manera, el usuario puede acceder a ellas de forma remota disponiendo de una conexión a internet y un navegador web. Todos los usuarios pueden tener la misma experiencia independientemente de su hardware.
Ventajas:

- Accesibilidad. Todos los usuarios disponen del mismo conjunto de herramientas y versiones.

- Escalabilidad. El entorno puede ser replicado fácilmente tantas veces como se desee.

Inconvenientes:

- Dependencia de la red. Las máquinas virtuales requieren de una cantidad significativa de recursos de hardware así como que estos permitan ejecutar

las tecnologías de virtualización.

- Coste. Al ser un sistema empaquetado cerrado, para actualizar cualquiera de las herramientas o realizar un cambio hay que recrear y redistribuir el sistema completo.

2.4 Ventajas específicas de los entornos de desarrollo remotos

Estos entornos tienen varias ventajas significativas frente al resto de tipos de entornos de desarrollo. A continuación se describen las mas importantes así como su impacto en los usuarios.

- **Accesibilidad.** Los entornos de desarrollo remotos permiten su acceso desde cualquier lugar mediante una conexión a internet. Con esto se evita la necesidad de tener un hardware específico para ello o estar ligado a una estación fija de trabajo. Además, también se evita la parte de instalación y configuración del software por lo que hace que el desarrollo sea mas flexible.

- **Estandarización.** Todos los usuarios del entorno de desarrollo trabajan con las mismas herramientas en un entorno idéntico y replicable en el tiempo, evitando así los problemas que pueden ocasionar los requisitos de hardware, versiones y configuración realizadas individualmente. De esta forma se pueden replicar los entornos y solucionar los problemas que puedan ocurrir de forma genérica. También permite que las actualizaciones sean automáticas y generalizadas.

- **Escalabilidad.** El despliegue físico de estos entornos se realiza en nubes o infraestructuras de servidores que permiten aumentar sus recursos y potencia de forma dinámica según las necesidades inmediatas ajustadas a la carga de trabajo. De esta forma, además de disponer de una potencia de cálculo mayor y centralizada, permite optimizar más el uso de ella y poder ofrecer otras ventajas del trabajo en la nube como almacenamiento o tecnologías específicas como GPUs o redes neuronales de aprendizaje automático.

- **Integración.** Este sistema permite que sea fácilmente integrable con herramientas de almacenamiento y versionado de repositorios de código como GitHub o Gitlab. Asimismo, también permite integrarlos con sistemas de integración continua CI/CD, para la ejecución de tests unitarios o de integración.

- **Productividad.** Este tipo de tecnologías ahorra tiempo de configuración descarga y solución de problemas o incompatibilidades con el equipo del usuario final, por lo que ahorra tiempo y permite dedicarlo a la tarea principal de desarrollo de código. Además, al estar integrado permiten recrearse desde 0 cuando se desee o incluso desplegar el código desarrollado en su destino final.

- **Seguridad.** Al hacer uso de una nube, permite que se pueda aplicar distintas capas de seguridad. Estas pueden incluir el aislamiento físico de los entornos de desarrollo, el cifrado de los datos, de las comunicaciones, y la autentificación de usuario multi-factor. De esta manera se consigue evitar problemas por robo del equipo informático o se puede revocar o restringir el acceso a las distintas partes del entorno de forma rápida y sencilla.

2.5 Inconvenientes específicos de los entornos de desarrollo remotos

Estos entornos no están exentos de sufrir varios inconvenientes o desventajas frente al resto de tipos de entornos de desarrollo. A continuación se describen las más importantes así como su impacto en los usuario.

- **Dependencia de la red.** Estos requieren de una conexión a internet continua y estable. Aunque inicialmente el ancho de banda necesario no es muy importante, la estabilidad de la conexión así como disponer de ella hacen que el usuario siempre tenga que trabajar conectado, estando a merced de la conexión.

- **Coste.** La mayoría de las infraestructuras de servicios de entornos de desarrollo remoto son de pago y requieren de uso de nubes propietarias y de una suscripción por usuario. Esto puede aumentar los costes de operación.

- **Dificultad de configuración.** Aunque para realizar desarrollos simples la configuración suele ser mínima, si se requiere de entornos de desarrollo complejos con muchas herramientas específicas así como distintas dependencias, puede suponer un problema para gestionarlos y hacer que sean operativos entre ellos.

- **Elección de plataforma.** Se trata de una tecnología totalmente emergente y reciente y está en continua evolución. Debido a esto, aunque existe alguna plataforma dominante, hay una gran cantidad de soluciones distintas y elegir la más adecuada para cada caso puede suponer un reto. Además, debido a la posible complejidad de la configuración del entorno, cambiar de proveedor o de plataforma puede suponer un problema en el futuro.

- **Privacidad y seguridad.** Si se hace uso de nubes propietarias, no se tiene el control total sobre los datos o el acceso a ellos y pueden sufrir ataques generalizados a la plataforma o de interrupción del servicio. Además, si la nube es privada, toda interacción del usuario queda registrada en el sistema y puede plantear problemas éticos y legales sobre el control de los usuarios y sus datos.

- **Falta de control.** No se tiene el control de las herramientas instaladas por lo que el usuario no puede elegir y puede sufrir restricciones frente a cómo desarrolla el código, personalizaciones y elección de tecnología.

2.6 Tendencia actual y futuro de la industria

Los entornos de desarrollo estandarizados remotos son una tecnología relativamente nueva y emergente y como tal dispone de una gran cantidad de soluciones distintas, desde empresas de nueva creación hasta gigantes de la industria y del cloud privado como Microsoft, Google, Redhat o Amazon. Además, cada mes aparecen nuevas soluciones y actores que entran a participar y a hacer evolucionar este ecosistema.

Uno de los principales actores es Gitpod. Esta empresa ha ganado cierta posición dominante en el mercado debido a que fue una de las primeras en ofrecer sus servicios clouds. Dan la posibilidad de disponer de un entorno de desarrollo listo para usar en segundos, basándose en un modelo de contenedores y posible integración con plataformas de versión de código como GitHub y Gitlab.

Por otro lado, GitHub Codespaces, impulsado por Microsoft, se está extendiendo de forma relativamente rápida ya que viene integrado de serie directamente en GitHub y permite iniciar el entorno de desarrollo con el código que se está consultando y ofrece una herramienta sencilla para colaborar con varios desarrolladores.

Además, existe una multitud de soluciones disponibles que suelen separarse entre *opensource* o privativas y de despliegue en nube pública o privada. Esto significa que hay una amplia gama de opciones a elegir según necesidades del sector, preferencias y recursos disponibles.

Respecto a la tecnología que hace posible estos entornos de desarrollo, la mayoría de los actores del mercado hacen uso de algún tipo de contenedor, principalmente Docker, para, mediante un lenguaje de descripción de configuraciones como YAML, definir el entorno y desplegarlo en la infraestructura.

Otro elemento que esta irrumpiendo con fuerza en el mercado y que va a suponer uno de los factores diferenciales en la elección de tecnología es el uso de inteligencia artificial, IA, para el desarrollo de código. Estas permiten que mediante una descripción del propósito del programa, la IA haga una propuesta de código funcional. También, permite revisar el código desarrollado, depurar, proponer cambios o ayudar al usuario en su redacción en directo.
En este aspecto Microsoft dispone de varias soluciones, siendo la más famosa GitHub copilot, por su parte, Amazon dispone de CodeWhisperer y Google presentó en mayo su producto Codey.

Con todo esto podríamos resumir el mercado actual y las opciones disponibles en el siguiente juego de características:

- **Tipo de licencia de software.**

  La elección entre Opensource o Privativo tiene un impacto significativo en

cómo se va a poder usar, modificar y distribuir el software.

- **Opensource.** Los productos con licencia de código abierto permiten usar, modificar y distribuir el código de forma libre y adaptarlo a las necesidades y preferencias. Además este tipo de licencia fomenta la transparencia, la colaboración y la evolución del código de una forma distribuida.

- **Privativa.** Con la licencia de software privativo, el código fuente no suele estar disponible y no se tiene la libertad de modificarlo por lo que estás a merced del fabricante y lo que se contrata y distribuye en ese momento. Las personalizaciones son más complicadas y costosas en caso de ser posibles.

- **Tipo de licencia de uso.**

  La elección entre pago por usuario, por organización o gratuito, definirá parte de los costes iniciales de aplicar este tipo de tecnología. Además los servicios gratuitos no suelen llevar asociado soporte técnico o actualizaciones. Las licencias de pago pueden ser mensuales, anuales o de pago por versión del software.

  - **Pago por usuario.** En ese modelo, cada usuario deberá disponer de una licencia activa para poder utilizar el entorno de desarrollo.

  - **Por organización.** En este modelo, la organización, entidad o empresa, realizará un pago único para obtener una licencia de uso del producto que le permitirá tener tantos entornos como desee.

  - **Gratuito.** En este modelo, no se tendrá que realizar ninguna compra de licencia para poder utilizar el software.

- **Tipo de infraestructura.**

  El tipo de infraestructura definirá la cantidad de control que se tiene sobre el entorno de desarrollo y su despliegue, así como el esfuerzo para administrarlo y mantener la infraestructura. Además, el coste también será un factor a tener en cuenta en esta área.

  - **Nube pública.** En este modelo, los entornos de desarrollo se alojarán en un proveedor de servicios Cloud como puede ser AWS o Azure. De esta forma la gestión y mantenimiento de esta recaerá en estas empresas y descargará a la organización de estas tareas.

  - **Nube privada.** En este modelo, la organización tendrá que proveer de una infraestructura adecuada para alojar los entornos de desarrollo remoto. Tendrá control total sobre ella y la responsabilidad de mantenerla, monitorizarla y actualizarla.

- **Tipo de despliegue.**

  El método de despliegue influirá en la eficiencia, portabilidad, escalabilidad y reproducibilidad de los entornos de desarrollo.

  - **Contenedores Docker.** En este tipo de despliegue se empaquetarán las aplicaciones y sus dependencias en un contendor estandarizado. Los usuarios podrán ejecutar estos contenedores individualmente o compartirlos.

  - **Cluster Kubernetes.** Este despliegue es una evolución del anterior ya que se encarga de la orquestación de contenedores que permite gestionar, escalar y desplegar estos en clústeres de servidores.

  - **Máquinas virtuales.** Este despliegue ofrecerá un entorno de desarrollo completo que simula sistema operativo, recursos de hardware específicos y aplicaciones con sus dependencias. Ofrece una gran asilamiento pero son pesados de manejar.

  - **Scripts de aprovisionamiento.** Este tipo de despliegue permite automatizar la instalación y configuración de las herramientas en una cuenta de usuario en un servidor.

  - **Local.** Este despliegue ofrece realizar una configuración local de todas las herramientas necesarias.

- **Inteligencia Artificial.**

  Incorporar herramientas que dispongan de Inteligencia Artificial aplicada a la programación podrá suponer un incremento en la productividad así como aumentar la confianza del código desarrollado.

  - **Incorporada.** Se dispondrá de IAs capaces de ayudar a la redacción de código, así como a su depuración, mejora o la realización de test de integración y unitarios.

  - **No incorporada.** No incorpora ningún tipo de ayuda proporcionada por IAs.

3. Preparación previa al estudio de campo

3.1 Creación de repositorio de código

Como punto de partida, se crea un repositorio de código en GtiHub ya que muchas de las soluciones requieren de una fuente de código en la que basarse para poder iniciar el entorno de desarrollo.

Asignamos un nuevo repositorio privado de código el nombre de uoc-tfg, y lo licenciamos con AGPLv3 para que quede amparado bajo la licencia más común de código abierto y que permite el trabajo derivado siempre bajo la publicación del código.

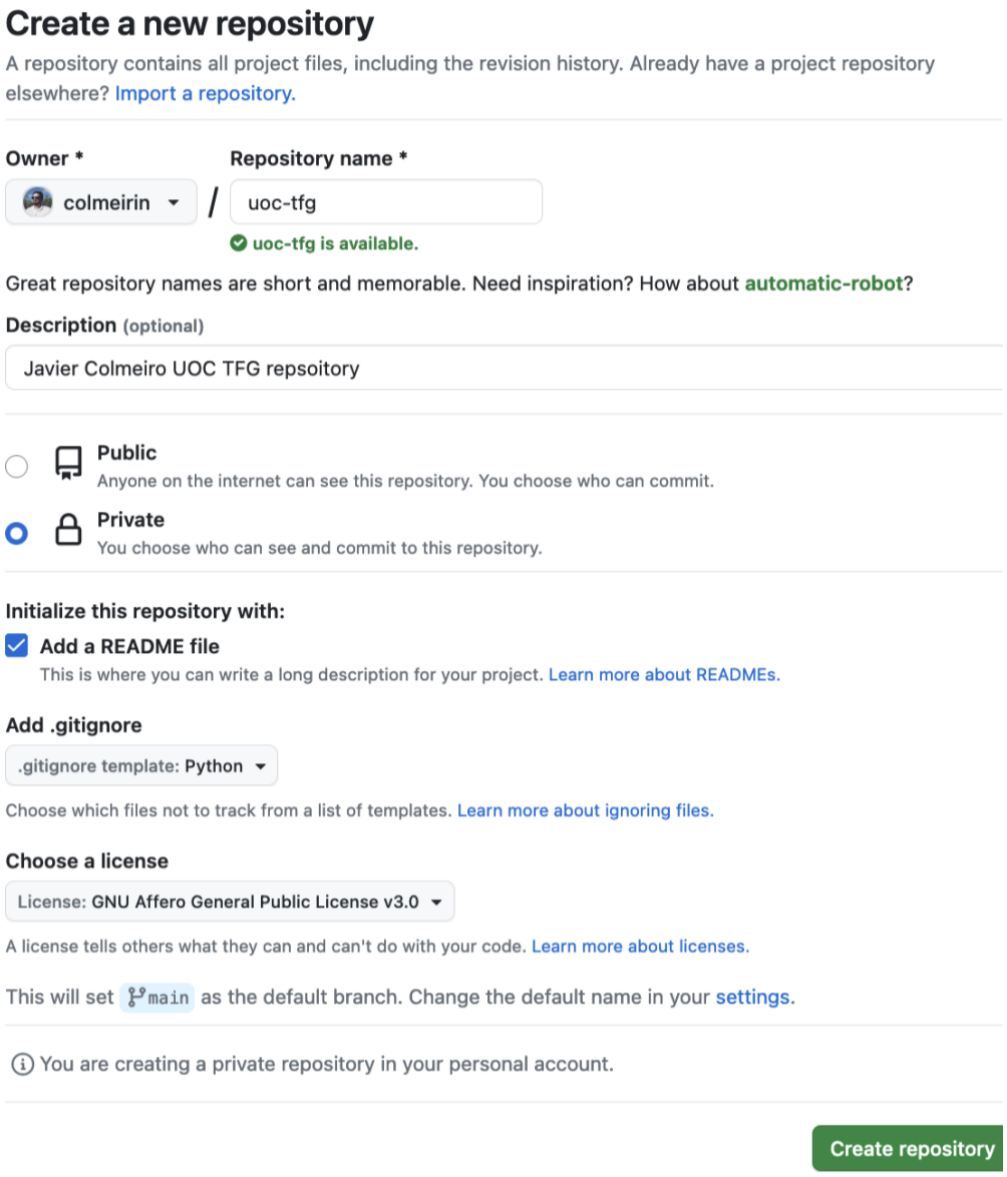


Ilustración 3. Creación repositorio de código en GitHub

En la máquina local clonamos el repositorio y añadiremos un pequeño fichero de código Python que nos permita ejecutar los entornos de desarrollo a probar.

```
[colmeirin@MapplePro sem14-TFG % git clone https://github.com/colmeirin/uoc-tfg
 Clonando en 'uoc-tfg'...
 remote: Enumerating objects: 5, done.
 remote: Counting objects: 100% (5/5), done.
 remote: Compressing objects: 100% (4/4), done.
 remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
 Recibiendo objetos: 100% (5/5), 13.63 KiB | 6.82 MiB/s, listo.
colmeirin@MapplePro uoc-tfg % nano README.md
colmeirin@MapplePro uoc-tfg % git add README.md jcolmeiro_uoc_prime_numbers.py
colmeirin@MapplePro uoc-tfg % git commit
[main beea5aa] + Initial commit adding sample .py and README
 2 files changed, 28 insertions(+)
 create mode 100644 jcolmeiro_uoc_prime_numbers.py
colmeirin@MapplePro uoc-tfg % git push
Enumerando objetos: 6, listo.
Contando objetos: 100% (6/6), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (4/4), listo.
Escribiendo objetos: 100% (4/4), 803 bytes | 401.00 KiB/s, listo.
Total 4 (delta 0), reusado 0 (delta 0), pack-reusado 0
To https://github.com/colmeirin/uoc-tfg
   d4652af..beea5aa  main -> main
```

Ilustración 4. Clonado local del repositorio e inicialización

Por último, el objetivo de las pruebas de campo de las soluciones será ejecutar un entorno de desarrollo remoto en la nube capaz de crear, compilar y depurar código Python y auxiliarmente, ejecutar el motor de bases de datos PostgreSQL. Muchas de las soluciones requerirán de este repositorio para poder utilizarse y también se guardará el contenido de los ficheros de configuración utilizados en cada unas de las plataformas.

3.2 Preparación de la infraestructura nube privada (Proxmox, lvm, Docker)

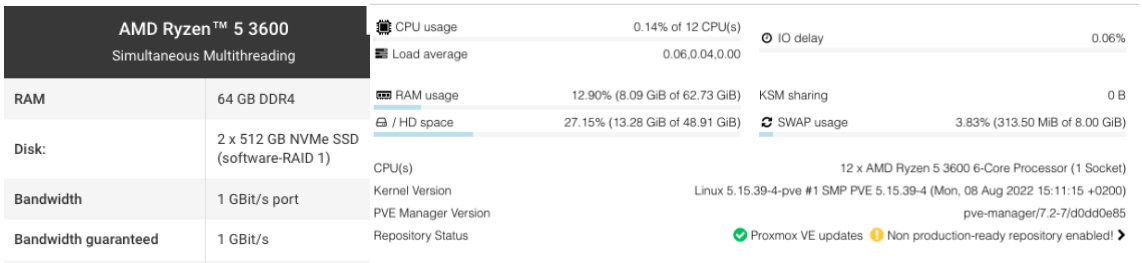


Ilustración 5. Características de hardware del servidor

Disponemos de un servidor root con 12 x AMD Ryzen 5 3600 con 6 cores a 4,2 Ghz, 64GB de memoria RAM y un sistema de almacenamiento RAID 1 de 512 GB NVMe SSD. En él tenemos instalado el software de gestión de contenedores y virtualización Proxmox en su versión v7 y el servicio de proxy inverso Nginx.

Con estos datos iniciales en cuenta, dimensionamos la creación del contenedor para alojar el servicio con el siguiente proceso.
Iniciamos el proceso de preparación de la infraestructura creando un nuevo contenedor.

Ilustración 6. Configuración Proxmox - crear contenedor 1

Nombramos el contenedor donde vamos a instalar el servicio como “uoc”. Asignamos una contraseña al usuario root que debe ser suficientemente robusta sin incluir palabras de diccionario.

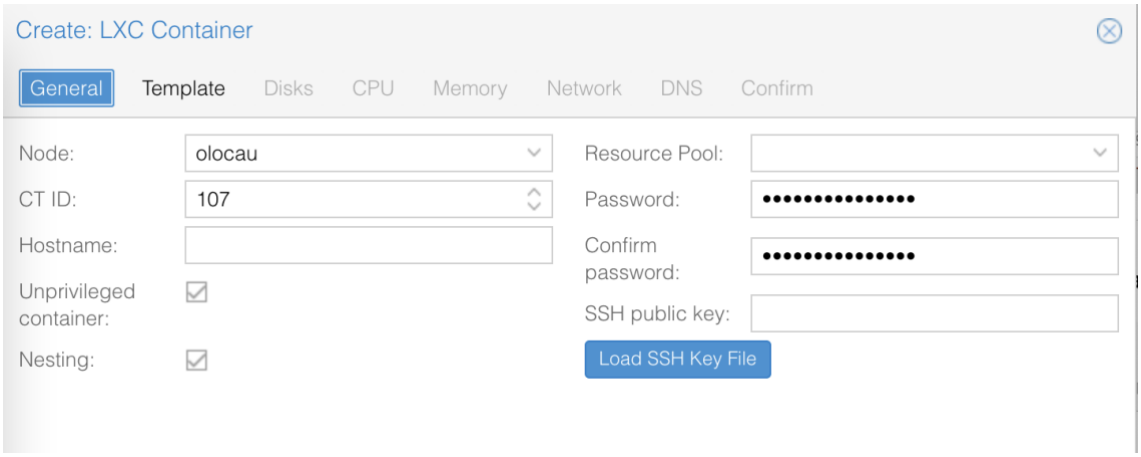


Ilustración 7. Configuración Proxmox - crear contenedor 2

Seleccionamos como sistema operativo base, una imagen de Linux Debian 11. Se utiliza esta distribución Linux y versión por ser una elección muy habitual en el mundo de la administración de servidores. Además, la solución elegida dispone de instrucciones de instalación para esta versión, así como paquetería del tipo deb la cual es compatible con esta distribución.

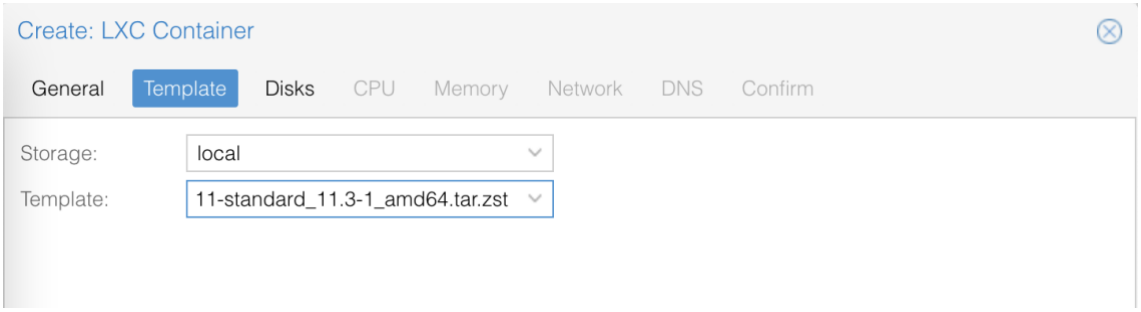


Ilustración 8. Configuración Proxmox - crear contenedor 3

Asignamos los valores de almacenamiento físico, potencia de proceso y memoria. La memoria RAM disponible para el contenedor se especifica a 16Gb. Esto supone un cuarto de la total disponible en el servidor y es un valor correcto para empezar a desplegar la solución. En caso de requerir mas se puede ampliar en cualquier momento. Por otra parte, la memoria de intercambio swap se asigna a un cuarto de la memoria RAM, es decir, 4Gb, el cual suele ser el valor recomendado.

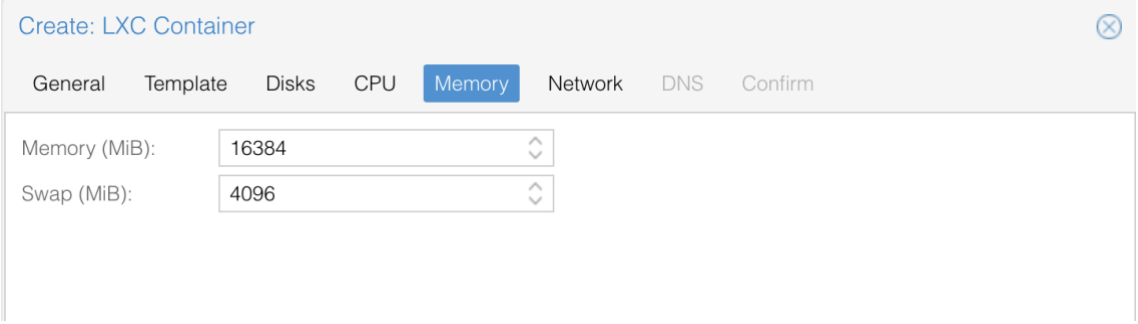


Ilustración 9. Configuración Proxmox - crear contenedor 4

Para la potencia de proceso, se asignan inicialmente 4 procesadores de los 12 disponibles. De esta manera nos aseguramos la estabilidad del sistema y capacidad de crecimiento en caso de necesitarla.

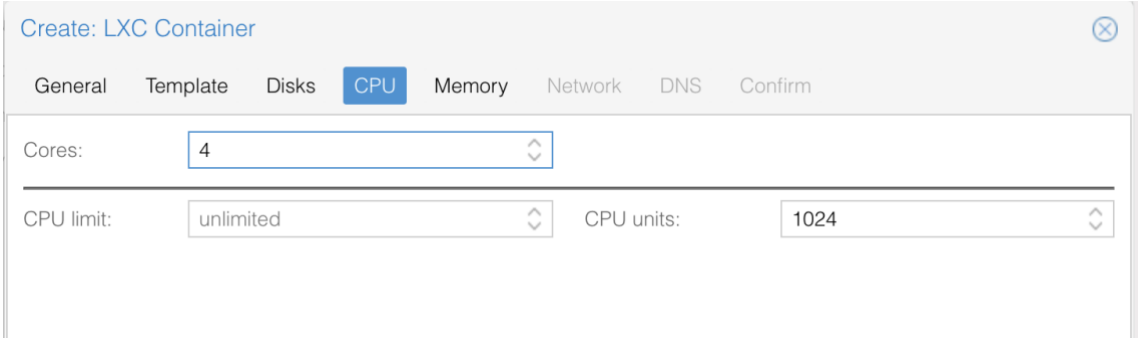


Ilustración 10. Configuración Proxmox - crear contenedor 5

Respecto al almacenamiento físico, asignamos 8Gb de los 512GB disponibles en el raid. La aplicación de Coder en sí ocupa unos 500mb y los entornos de desarrollo, al ser contenedores Docker mínimos deberían ser también de un tamaño reducido. No obstante, como el resto de los parámetros, se puede ampliar en cualquier momento.

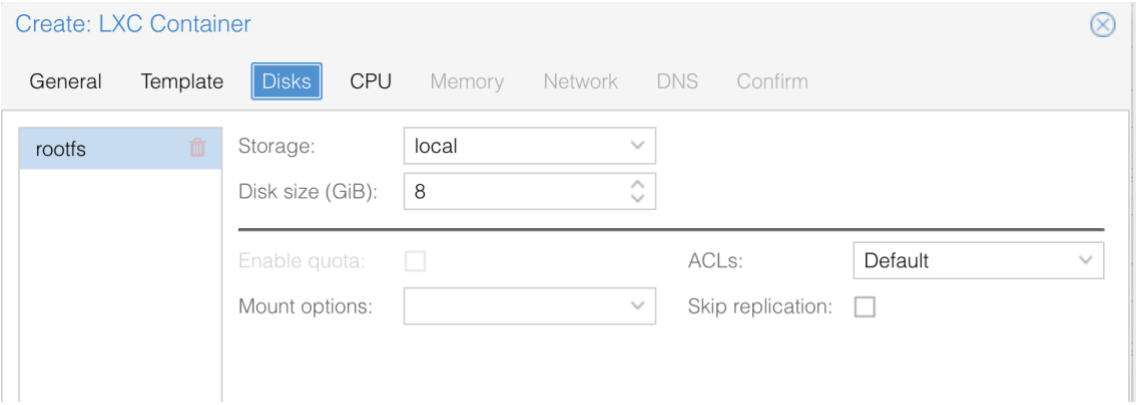


Ilustración 11. Configuración Proxmox - crear contenedor 6

Configuramos ahora la tarjeta de red virtual utilizando el bridge del servidor de cabecera y le asignamos una ip manual que coincide con el número de contenedor del servidor para ser fácilmente identificable. Asignamos como Gateway, el del servidor host para poder tener conexión a internet.

Ilustración 12. Configuración Proxmox - crear contenedor 7

Finalmente, confirmamos el proceso y se crea el contenedor con las características que hemos especificado.

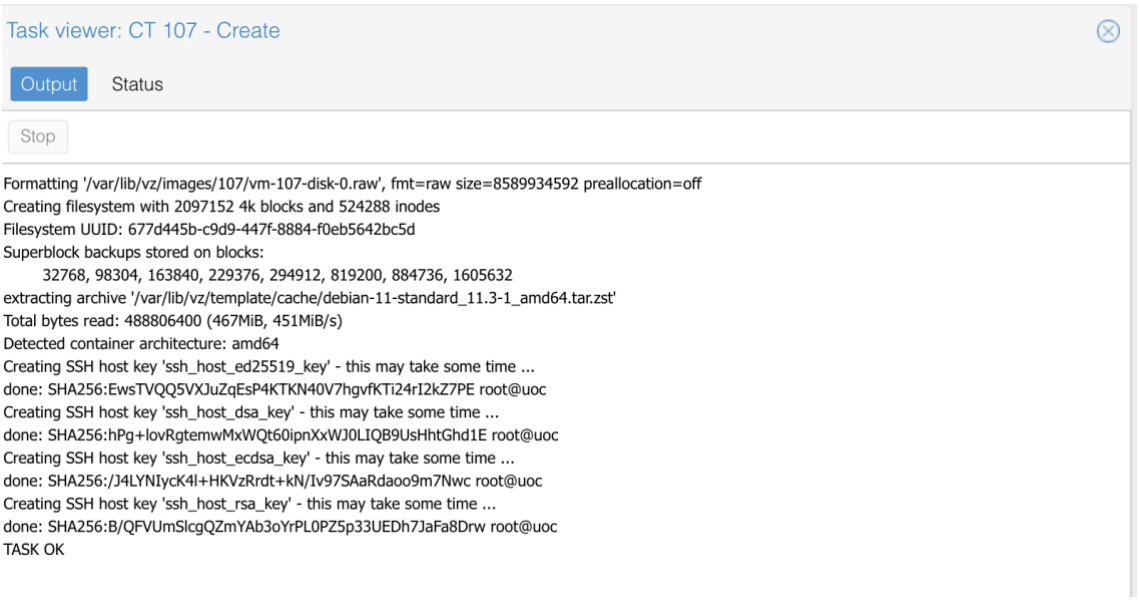


Ilustración 13. Configuración Proxmox - crear contenedor 8

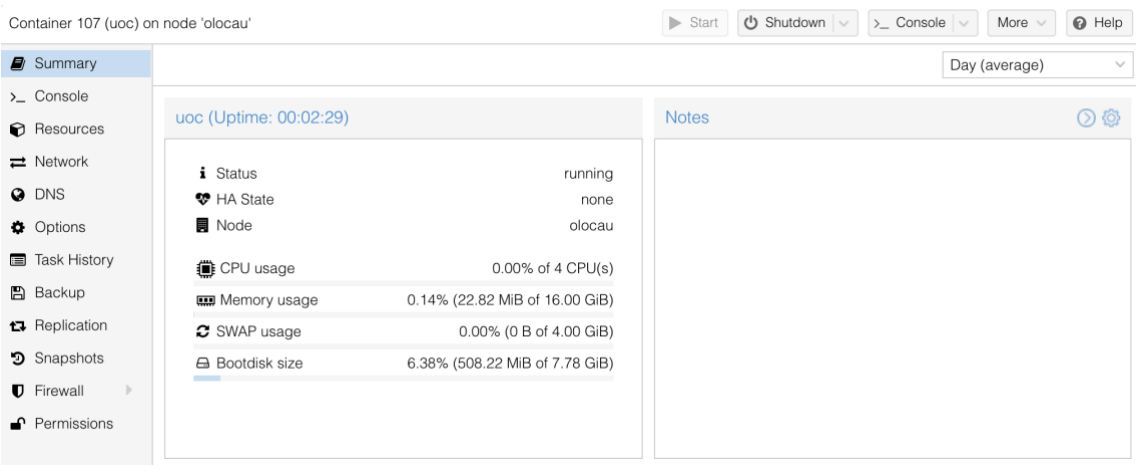


Ilustración 14. Configuración Proxmox - crear contenedor 9

A continuación, nos conectaremos al contenedor para iniciar las tareas iniciales de actualización de paquetería, configuraciones básicas e instalación de la solución elegida.

Empezamos actualizando el sistema e instalamos un juego de herramientas básicas de administración de sistemas que nos permitan trabajar de forma cómoda. También incluimos algunas aplicaciones que proporcionan una capa básica de seguridad como iptables, nftables, ufw y fail2ban.

```
lxc-attach 107 apt-get update
apt-get update
apt-get upgrade
apt-get update && apt-get install -y sudo curl wget net-tools apt-
transport-https ca-certificates software-properties-common build-
essential gnupg2 iptables nftables ufw fail2ban rsync ssh htop iotop
nmon nano vim emacs git screen tmux zip unzip tar
```

Seguidamente, instalamos Docker y docker-compose. Para ello, actualizamos la fuentes de paquetería, instalamos los requisitos previos recomendados y añadimos el repositorio de fuentes específico de Docker.

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
sudo apt-get update
```

Podemos verificar que se ha añadido correctamente las fuentes de repositorio de Docker observando la lista de repositorios que se actualizan.

```
root@uoc:~#  sudo apt-get update
Hit:1 http://ftp.debian.org/debian bullseye InRelease
Hit:2 http://ftp.debian.org/debian bullseye-updates InRelease
Hit:3 http://security.debian.org bullseye-security InRelease
Get:4 https://download.docker.com/linux/debian bullseye InRelease [43.3 kB]
Get:5 https://download.docker.com/linux/debian bullseye/stable amd64 Packages [24.4 kB]
Fetched 67.8 kB in 0s (307 kB/s)
Reading package lists... Done
```

Ilustración 15. Fuentes de software de Docker añadidas

Podemos proceder a instalar ya Docker y docker-compose ejecutando los siguientes comandos.

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-
buildx-plugin docker-compose-plugin
sudo apt-get install docker-compose-plugin
```

Finalmente, en el servidor host que controla la red y el acceso, configuramos el firewall abriendo el tráfico del puerto 3000 para que permita el paso del tráfico de red al servicio y configuramos el proxy inverso Nginx para redirigir el tráfico de un dominio y le asignamos un certificado TLS auto firmado con certbot para hacer mas segura mediante el cifrado de la conexión y el tráfico https.

```
cd /etc/nginx/sites-available
nano uoc.URLTEST.com
cd /etc/nginx/sites-enabled
ln -s ../sites-available/uoc.URLTEST.com
certbot
```

Una vez realizado todo esto tendremos la infraestructura básica configurada para instalar aplicaciones que requieran de nube privada.

3.3 Preparación de cluster Kubernetes (Minikube, kubectl chectl)

Algunas de las soluciones a probar pueden requerir de un despliegue en cluster de Kubernetes. Esta infraestructura es costosa y difícil de configurar pero existe una alternativa llamada Minikube.

Minikube es un proyecto Opensource que permite instalar un cluster de Kubernetes en un contenedor Docker y simular que se dispone de una infraestructura compleja. Este tipo de despliegue está pensado para hacer pequeñas pruebas de concepto y nunca como un sustituto a una infraestructura de cluster de Kubernetes completa. No obstante, para el objeto de estudio de este trabajo es suficiente.

Para instar minikube necesitamos 2 CPUs 2GB de memoria RAM, y 20GB de espacio de disco. En el anterior paso creamos un contenedor base para nuestra infraestructura de 8GB de almacenamiento, para reaprovechar el trabajo de configuración hecho previamente, redimensionaremos el contenedor añadiéndole 40GB de espacio para poder instalar y desplegar minikube.

Para esto, nos dirigimos al panel de Proxmox y en la configuración de recursos del contendor accedemos a Volume Action > resize y especificamos el tamaño extra que queremos añadir.

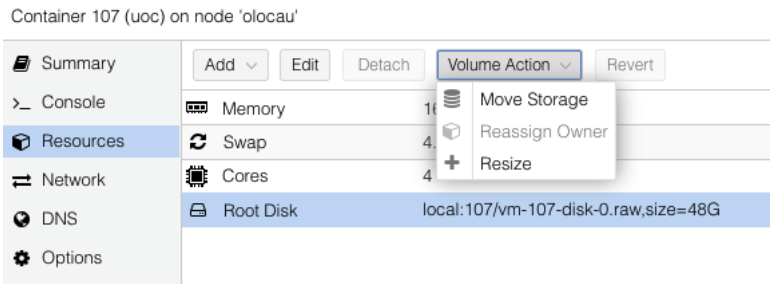


Ilustración 16. Configuración Proxmox - redimensionar disco

A continuación accedemos por consola al contenedor y ejecutamos los siguientes comandos para instalar los requisitos previos recomendados.

```
curl                                                                    -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
minikube config set driver docker
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo apt-get install -y apt-transport-https
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
```

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl
bash <(curl -sL  https://www.eclipse.org/che/chectl/)
```

Una vez ejecutados estos comandos podemos arrancar el Servicio de minikube con:

```
minikube start --addons=ingress,dashboard --vm=true --memory=10240 --
cpus=4 --disk-size=50GB --kubernetes-version=v1.23.9
```

Y tras un par de minutos tenemos disponible el Servicio de minikube en caso de necesitarlo.

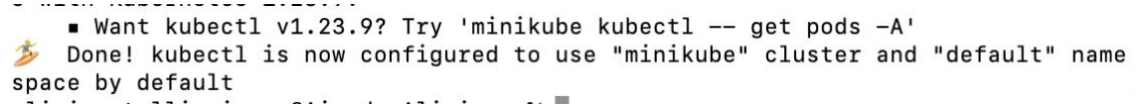


Ilustración 17. Configuración Proxmox - minikube instalado

4. Estudio de mercado sobre las soluciones actuales

4.1 Gitpod

Gitpod fue fundada en 2020 por un equipo de ingenieros cansado de tener que crear, gestionar y mantener entornos de desarrollo locales. Desde entonces y con una filosofía de código abierto, han desarrollado su tecnología de entornos de desarrollo remoto en la nube basándose en tecnología de Docker y Kubernetes.

Esta orientado para ser integrado directamente con un repositorio de código de los principales proveedores como GitHub, GitLab o Bitbucket. Para la configuración de las características del entorno de desarrollo utilizan la sintaxis y lenguaje de YAML utilizada también para la creación de contenedores Docker.

Ofrece una versión PaaS en su propia cloud que se basa en créditos de uso y también permite el despliegue en cloud pública de Amazon o Google. Anteriormente ofrecía una versión self-hosting pero ha sido descatalogada de forma pública y han abierto una lista de espera para la nueva versión *on premise*.

Por defecto, una nueva cuenta de usuario traerá asociado 500 créditos que equivalen a 50 horas de uso en su máquina estándar de 4 cores, 8GB de RAM y 30GB de almacenamiento. A partir de ahí se puede expandir a 1000 créditos al mes por 9€ y seguir usándolo a 0,036 el crédito extra. Por último, ofrece una versión Enterprise que es adaptable y configurable para organizaciones más grandes. Esta versión permite además de mayores capacidades de proceso, poder utilizar espacios aislados, conexiones privadas, ejecutarse en nube privada, etc. No obstante, esta versión no se ha podido evaluar ya que requiere pasar por un embudo de ventas para organizaciones.

En la siguiente tabla se pueden comprobar los distintos tipos de cuentas y sus características y condiciones:

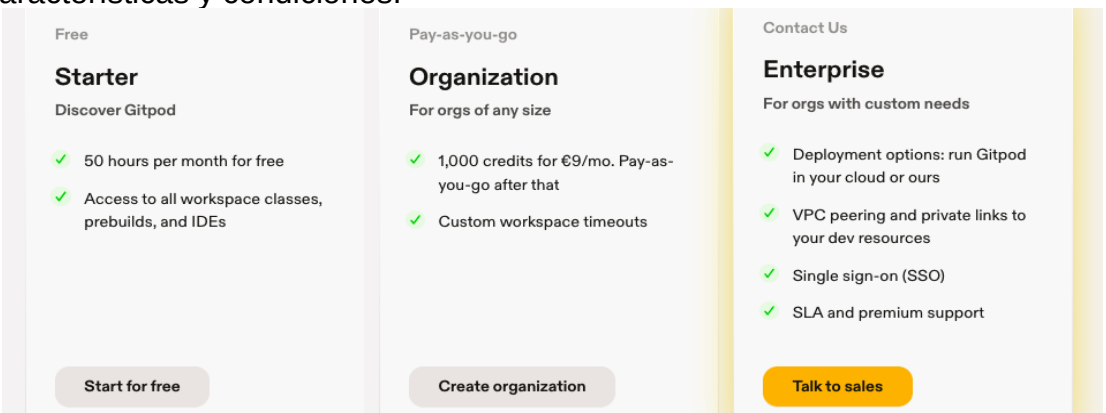


Ilustración 18. Gitpod – precios

En el caso de querer utilizar la opción de máquinas más potentes de 8 núcleos, 16GB de RAM y 50GB de almacenamiento, se consumirá el doble de créditos por hora, de 10 a 20, por lo que la cuenta básica se reducirá a 25 horas al mes y el coste extra por hora de uso será de 0,72€/hora.

Para iniciar la configuración y pruebas de esta plataforma, nos registramos en la aplicación web con una cuenta de GitHub confirmamos el acceso a nuestros datos y configuramos el editor de código por defecto y el estilo visual que queremos utilizar.

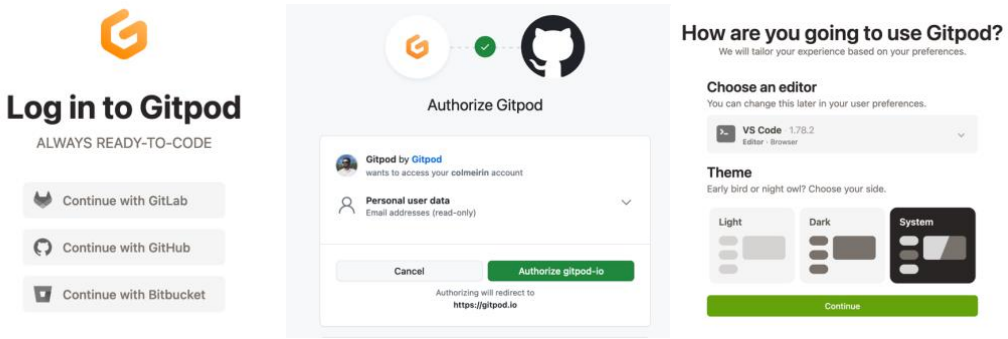


Ilustración 19. Gitpod - autenticación con GitHub

A continuación, crearemos los ficheros de configuración que definirán el entorno de desarrollo en la nube que vamos a utilizar así como sus herramientas de software pre-instaladas.

Para ello crearemos primero un fichero llamado .gitpod.yml en nuestro repositorio de código y en el le especificamos la imagen de Docker a utilizar y las tareas a realizar durante el compilado del entorno.

```
image:  gitpod/workspace-postgres
```

Gitpod ofrece una serie de imágenes de Docker pre-compiladas y listas para usar así que utilizaremos la de PostgreSQL aunque podríamos indicarle que vamos a utilizar un fichero .Dockerfile local para realizar todas las especificaciones que queramos.

Seguidamente y con el fichero ya subido en el repositorio, en la plataforma de Gitpod creamos un nuevo espacio de trabajo.

Ilustración 20. Gitpod - creación nuevo entorno de desarrollo

El sistema comienza a compilar la máquina y al ser una imagen pre-compilada, se lanza el editor web y tenemos el sistema completo y usable en unos 10 segundos y con el motor de bases de datos funcionando.

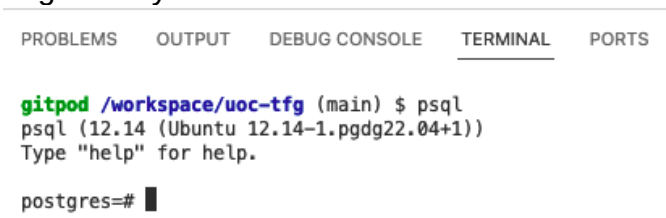


Ilustración 21. Gitpod - PostgreSQL ejecutándose en el contenedor

Podemos también lanzar el entorno en nuestro editor de código local, pero para ello tendremos que instalar la extensión de Gitpod para VS Code, autorizar y autenticarnos con la aplicación de Gitpod y por último, abrir el entorno de desarrollo remoto.

Ilustración 22. Gitpod - proceso de configuración de VS Code local

A continuación, se adjunta un fichero de configuración Dockerfile donde se utiliza la imagen anterior y se realizan unas acciones posteriores para adecuar las versiones de Python. El propósito de esta prueba es comprobar el tiempo de compilado y creación del entorno de desarrollo.

Modificamos el archivo .gitpod.yaml con el siguiente contenido:

```
image:
  file: .gitpod.Dockerfile
```

Y creamos el fichero .gitpod.Dockerfile con el siguiente contenido para actualizar la paquetería, desinstalar la versión de pyenv pre-instalada e instalar una específica.

```
FROM gitpod/workspace-postgres:latest

RUN sudo apt-get update \
    && sudo apt-get update \
    && sudo apt-get clean \
    && sudo rm -rf /var/cache/apt/* /var/lib/apt/lists/* /tmp/*

WORKDIR /home/gitpod/
RUN rm .pyenv -Rf
RUN rm .gp_pyenv.d -Rf
RUN curl https://pyenv.run | bash
RUN pyenv update && pyenv install 3.10.7 && pyenv global 3.10.7
RUN pip install pipenv
```

Una vez guardados los cambios en los ficheros del repositorio, volvemos a la web de Gitpod, borramos el entorno creado anteriormente y creamos un nuevo entorno apuntando al mismo repositorio. Esta vez, al tener que compilar un entorno de desarrollo no estándar tarda unos 7 minutos en ofrecernos el entorno de desarrollo remoto listo para su uso.

Como resumen final, Gitpod ofrece una solución muy completa y fácil de configurar para la gestión de entornos de desarrollo remotos. Con sus imágenes pre-compiladas, ofrece entornos de desarrollo estandarizados para la mayoría de los lenguajes y motores de base de datos y permite parametrizarlos con Dockerfiles o docker-compose si se requiere. Es fácil de usar y la latencia es prácticamente nula.
Como puntos negativos, en su versión gratuita solo permite el uso de su nube propietaria y en las versiones que lo permite, actualmente tiene una lista de espera por lo que no se puede evaluar. Este punto descarta su uso si se quiere utilizar una infraestructura privada, la cual aporta una capa de seguridad y control mayor.

Ideas clave de la solución evaluada:

¿Licencia de software? Opensource
¿Tipo de infraestructura? Nube pública
¿Licencia de uso? Gratuita / de pago
¿Pago por uso? Sí
¿Disponible en IDE local? Sí, mediante extensión
¿Tecnología contenedores? Docker / Kubernetes
¿Fichero de configuración? gitpod.yaml
¿Docker-compose? Sí
¿Dificultad de configuración? Fácil
¿Dificultad de uso? Baja
¿Latencia de uso? Inapreciable
¿Tiempo de compilación del entorno de desarrollo? 10 segundos / 7 minutos
¿Tiempo de inicio una vez compilado? 5 segundos

4.2 GitHub CodeSpaces

GitHub es el repositorio de código mas famoso y utilizado del mundo. Es propiedad de Microsoft y ofrece CodeSpaces como su propuesta al ecosistema de entornos de desarrollo remotos.

Este sistema utiliza el concepto de Codespaces para crear espacios de trabajo en la nube en base a un repositorio de código concreto y utilizando una tecnología propia llamada *devcontainers* o contenedores de desarrollo. Estos contenedores, mediante la utilización de ficheros de configuración JSON, permiten describir las características del entorno de desarrollo a utilizar así como posibles configuraciones y peculiaridades.

Dispone de un Marketplace con un buscador donde podemos encontrar las soluciones compatibles. En ella encontramos la gran mayoría de motores de programación listos para utilizar así como otras características más avanzadas como la utilización de Docker in Docker o Terraform para manejar infraestructuras.

Al ser un sistema PaaS tiene un modelo de negocio de pago por almacenamiento y potencia consumida por hora similar a la que ofrecen los proveedores de nubes públicas como Amazon AWS o Microsoft Azure.

Dispone de dos modelos de uso por defecto, asociados a las tipologías de cuentas de usuario de GitHub, el gratuito personal, *GitHub Free for personal accounts* y el de pago, *GitHub Pro*. El modelo gratuito va asociado de forma predeterminada a cualquier cuenta de usuario de GitHub y permite crear Codespaces para tus repositorios de código con las siguientes limitaciones; 15GB de almacenamiento y 120 horas de tiempo de proceso 1 núcleo cpu al mes. En la modalidad GitHub Pro se dispone de 20GB de almacenamiento y 180 horas de tiempo de proceso de 1 núcleo cpu al mes por $4.

A partir de la superación de los límites asociados a las cuentas de usuario se tarifica por tiempo de uso en base a la siguiente tabla:

Component	Machine type	Unit of measure	Included usage multiplier	Price
Codespaces compute	2 core	1 hour	2	$0.18
Codespaces compute	4 core	1 hour	4	$0.36
Codespaces compute	8 core	1 hour	8	$0.72
Codespaces compute	16 core	1 hour	16	$1.44
Codespaces compute	32 core	1 hour	32	$2.88
Codespaces storage	Storage	1 GB-month	Not applicable	$0.07

Ilustración 23. Codespaces – costes de computación

La fórmula utilizada para el cálculo del tiempo de uso del almacenamiento es: horas de uso * GB de almacenamiento / (24 * 30). La fórmula para el calculo del tiempo de uso de la potencia de proceso es: horas de uso * número de núcleos de proceso. Teniendo estas limitaciones presentes, cualquier usuario de GitHub puede utilizar de forma gratuita un entorno de desarrollo con 2 núcleos de proceso y 2GB de espacio de almacenamiento durante 2 horas a la semana, por ejemplo.

Para iniciar la configuración y pruebas de esta plataforma, navegaremos hasta el repositorio de código creado inicialmente y clicaremos el botón verde de Code y en el menú de puntos suspensivos, seleccionamos “Configure dev container”.

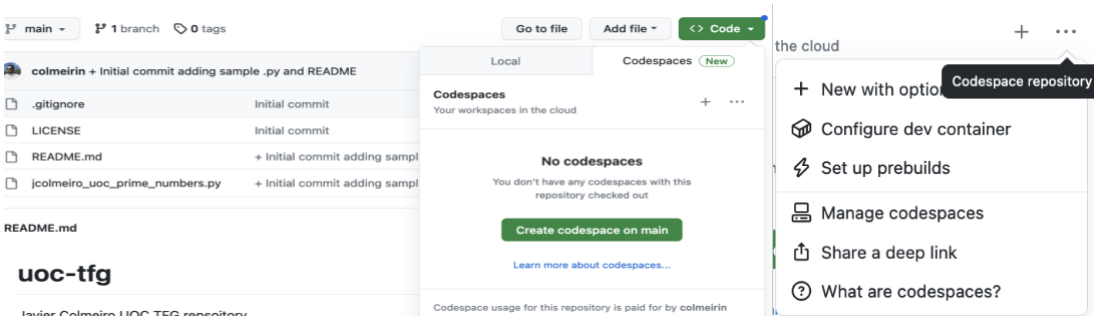


Ilustración 24. Codespaces – proceso creación entorno

Creamos el siguiente fichero de configuración JSON que especifica un entorno de desarrollo de Python en la versión 3.11 y el motor de bases de datos PostgreSQL.

```
{
  "image": "mcr.microsoft.com/devcontainers/universal:2",
  "features": {
    "ghcr.io/devcontainers/features/python:1": {
      "version": "3.11",
      "installTools": true
    },
  "ghcr.io/devcontainers-contrib/features/postgres-asdf:1": {
      "version": "latest"
    }
  }
}
```

Una vez hechos los cambios se guardan en un commit al repositorio y el sistema creará una carpeta .devcontainer con el fichero devcontainer.json con nuestra configuración. A partir de este momento el repositorio estará listo para poder ejecutar la creación de un Codespaces Workspace.

Finalmente, creamos el entorno de desarrollo con el botón de “Create codespace on main”. Permitirá elegir si se quiere una máquina de 2 core o de 4 core y la región geográfica del servidor a utilizar. Una vez confirmado, comenzará el proceso de construcción del contenedor. Este proceso de compilación para la configuración anterior ha tardado 25 minutos. Una vez compilado el contenedor, este empezará a consumir horas de recursos según los límites planteados

anteriormente, por lo que es conveniente apagar el contenedor cuando no se usa.
Para realizar la prueba de uso, elegimos utilizar primero el sistema integrado en navegador web. Este se lanza en pocos segundos y tiene el estilo de VS Code Web. Permite ejecutar el código, depurar el código y tener acceso a la máquina para ver los ficheros creados o lanzar comandos.

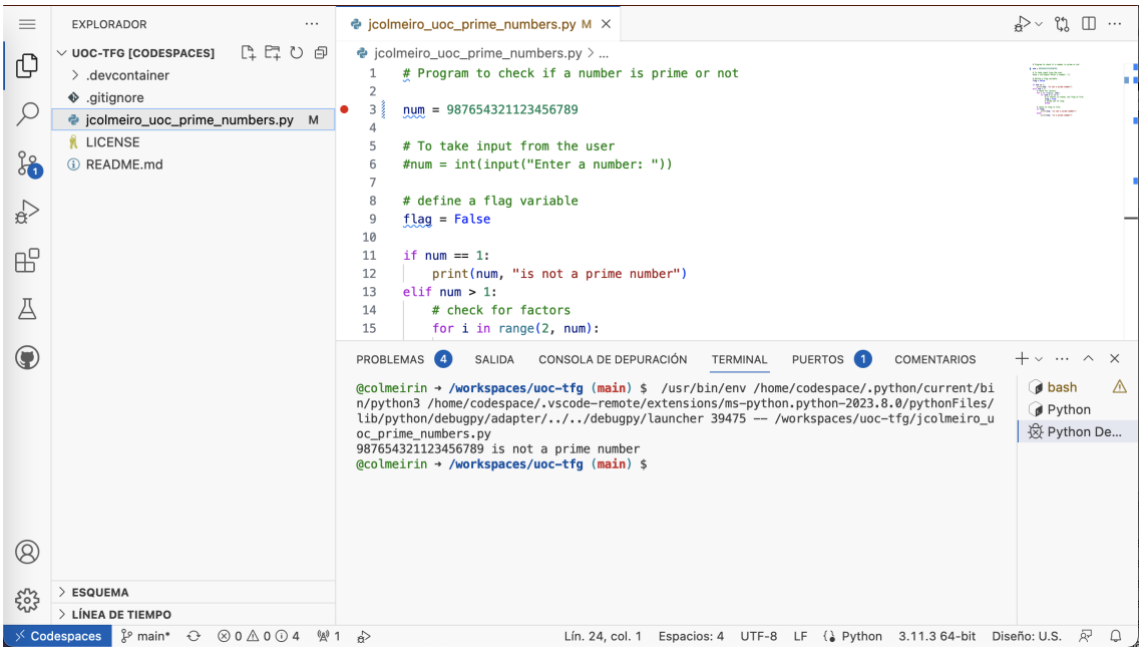


Ilustración 25. Codespaces – IDE ejecutándose

Para lanzar el entorno de desarrollo en local, buscamos e instalamos la extensión de VS Code para GitHub Codespaces, una vez instalada, podemos lanzar el IDE local desde el repositorio. Para ello, la primera vez pedirá verificar que queremos abrir la URI de la extensión y autorizar el acceso a GitHub para VS Code,

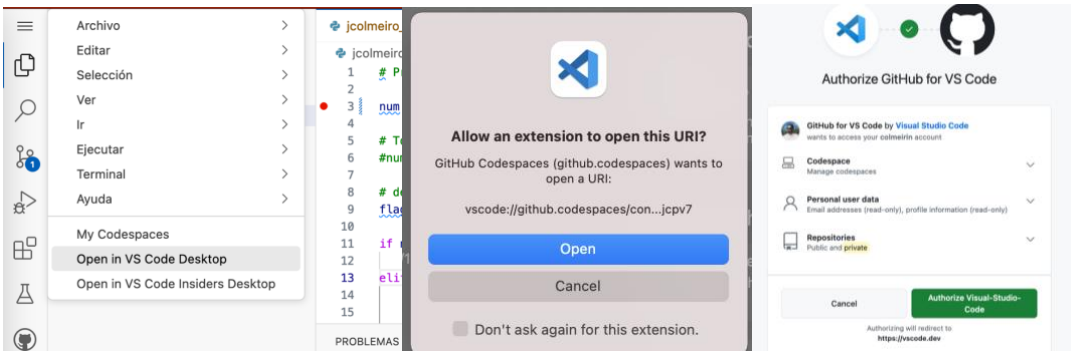


Ilustración 26. Codespaces – Proceso autorización para IDE local

Una vez terminada la configuración abrirá el IDE local. Esta opción puede ser interesante para algunos usuarios porque, aunque es contradictoria a la filosofía del desarrollo remoto en la nube, permite utilizar las configuraciones y personalizaciones que el usuario ya acostumbra a usar. No obstante, aunque sea un IDE instalado localmente en la máquina del usuario, en caso de perder la conexión, no podrá seguir trabajando en el código ni podrá ejecutarlo o lanzarlo ya que la extensión intentará reconectarse permanentemente.

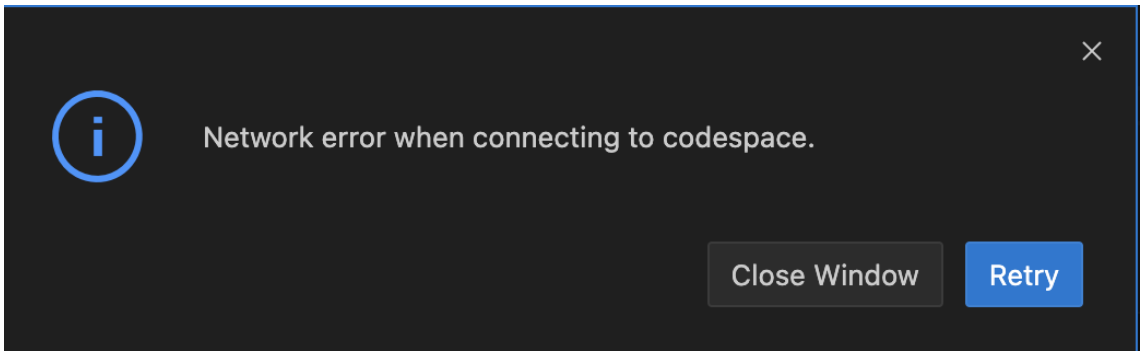


Ilustración 27. Codespaces – Error al lanzar IDE local

Como resumen final, Codespaces ofrece un sistema para la gestión de entornos de desarrollo en la nube completa y potente. Permite una configuración rápida y completa y adaptable al motor de código que se va a utilizar. Está bien integrado con los principales IDE del mercado y en su versión web no tiene latencia.
Como puntos negativos, solo permite el uso de la nube propietaria de GitHub y no se puede utilizar tecnologías específicas o asociadas a los contenedores Docker como docker-compose. Además, los tiempos de uso de potencia y almacenamiento que vienen incluidos en las cuentas gratuitas es mínimo para poder realizar pequeñas pruebas de concepto y supondrán un coste a tener en cuenta.

Ilustración 28. Codespaces – resumen de entornos activos

Ideas clave de la solución evaluada:

¿Licencia de software? Privada
¿Tipo de infraestructura? Nube pública GitHub
¿Licencia de uso? Gratuita
¿Pago por uso? Si
¿Disponible en IDE local? Si, Mediante extensión
¿Tecnología contenedores? Docker / Kubernetes
¿Fichero de configuración? Devcontainer.json
¿Docker-compose? No
¿Dificultad de configuración? Fácil
¿Dificultad de uso? Baja
¿Latencia de uso? Poca
¿Tiempo de compilación del entorno de desarrollo? 25 minutos
¿Tiempo de inicio una vez compilado? 10 segundos

4.3 JetBrains Space

JetBrains, anteriormente llamada IntelliJ Software es la mayor empresa europea en software para el desarrollo de código. Está especializada en los entornos de desarrollo integrado (IDE) y actualmente tiene una gran multitud de variantes de su producto adaptadas a las peculiaridades de cada lenguaje de programación. Además es el creador e impulsor del lenguaje de programación Kotlin

Con toda esta experiencia y pensado en las necesidades de las empresas que desarrollan código, ofrecen una solución completa llamada Space. Esta permite alojar repositorios Git, hacer revisiones de código, gestionar pruebas de integración, gestionar documentación y ejercer el desarrollo remoto.

Para esto último se basan en una tecnología propia basada en contenedores Docker y desplegada en cluster Kubernetes en su nube pública. Además ofrecen la opción de despliegue *on premise* controlado totalmente por el usuario y con posibilidad de elegir servidor con docker-compose o cluster Kubernetes.

Su modelo de negocio se basa por una parte en las licencias de uso y por otra en el pago por uso de computación de su nube pública en base a un sistema de créditos. No obstante, ofrece la posibilidad de usar nubes privadas u *on premise* sin costes adicionales al de la licencia.

Para el uso de su nube pública, dispone de 4 modalidades:

Free ofrece de forma global, 2000 créditos al mes, 10 GB de almacenamiento, 50GB de tráfico de datos de red y 1 entorno de desarrollo activo por miembro.

Team ofrece lo mismo que la free pero los limites de almacenamiento son por miembro en vez de globales, además dobla el numero de créditos mensuales y los espacios disponibles con un coste de 10€ al mes por usuario.

Organization ofrece 10000 créditos mensuales, y aumenta los limites por usuario a 25GB de almacenamiento, 125GB de tráfico de datos y entornos de desarrollo ilimitados con un coste de 25€ al mes por usuario.

Enterprise ofrece 50000 créditos mensuales y aumenta los limites por usuario a 100GB de almacenamiento y 500GB de tráfico de datos con un coste de 125€ al mes por usuario.

A estos costes, hay que sumarles los de las horas de computación que dependen del tipo de máquina y del tamaño del entorno de desarrollo. Ofrecen para esto 3 tipos de máquinas distintas de 4, 8 o 16 cores, 8, 16 o 32GB de RAM y tienen un coste de 50 ($0,40), 100 ($0,80), 200 ($1,60) créditos a la hora. Además hay que

sumarle el coste por almacenamiento el cual va por bloques de 40GB y cuestan 1,2 ($0,01) créditos a la hora.

Dev environment instance type	**Regular** (4 CPU Cores, 8 GB RAM, 40 GB disk drive)	1 hour	50 Computation Credits ($0.40)
	Large (8 CPU Cores, 16 GB RAM, 40 GB disk drive)	1 hour	100 Computation Credits ($0.80)
	Extra Large (16 CPU Cores, 32 GB RAM, 40 GB disk drive)	1 hour	200 Computation Credits ($1.60)
Dev environment storage (active or hibernated)	**40 GB**	1 hour	1.2 Computation Credits ($0.01)

Ilustración 29. JetBrains Space - costes computación

Para el uso de una nube privada u *on premise* dispone de 3 modalidades anuales:

Free permite utilizar hasta 10 aplicaciones en un despliegue privado para un máximo de 10 miembros.

Organization permite utilizar todas las aplicaciones disponibles por un importe de 25€ al mes por usuario facturado de forma anual. (A partir de 3993 €)

Enterprise permite utilizar todas las aplicaciones y todas las herramientas de software disponibles de esta empresa a partir de 50 usuarios con un coste de 125 € al mes por usuario facturado de forma anual. (A partir de 90750 €)

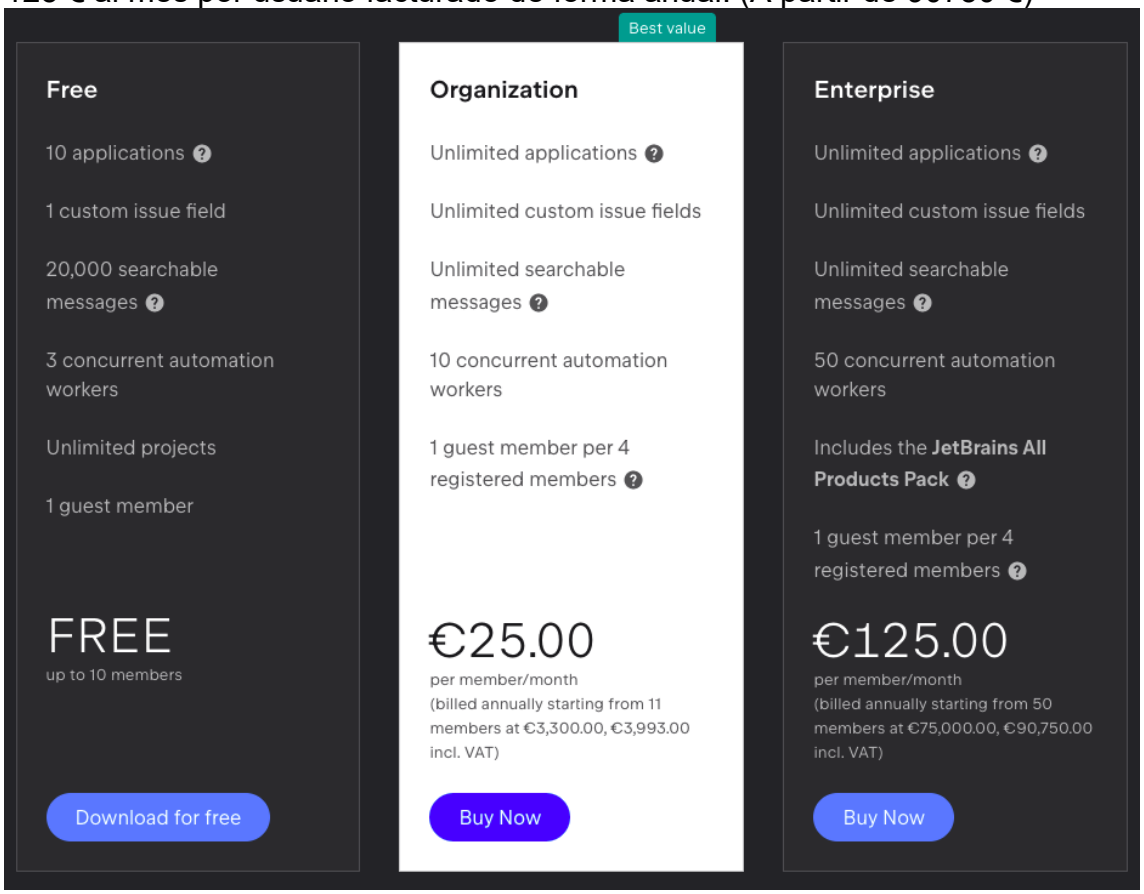


Ilustración 30. JetBrains Space - coste licencias de uso

Para iniciar las pruebas de esta plataforma, nos centramos primero en la versión cloud.
Para ello, nos registramos con el correo de la universidad y nos permite crear una url personalizada para el acceso. Una vez dentro, creamos un nuevo repositorio de código y especificaremos que queremos importar el que diseñamos inicialmente y se encuentra alojado en GitHub.

Una vez importado, podemos explorar el repositorio y ya finalmente abrirlo en el entorno de desarrollo remoto. Spaces permite definir y crear los entornos de desarrollo de varias formas, una automática en base al código y a un IDE concreto seleccionado y otra mediante ficheros de configuración de sintaxis YAML, devfile, Dockerfile y docker-compose.

Inicialmente, utilizamos el repositorio tal y como está para iniciar el entorno de desarrollo en su configuración básica en base al IDE seleccionado.

Ilustración 31. JetBrains Space - creación de entorno

Para este repositorio, seleccionamos la máquina básica y el IDE PyCharm en su última versión. Una vez confirmado, el sistema compila el entorno y en 2 minutos tenemos listo el entorno para ser ejecutado.

Spaces no tiene un editor web como tal integrado, por lo que requiere instalar su aplicación Gateway, la cual permite conectarse de forma remota a entornos de desarrollo suyos o de la competencia.

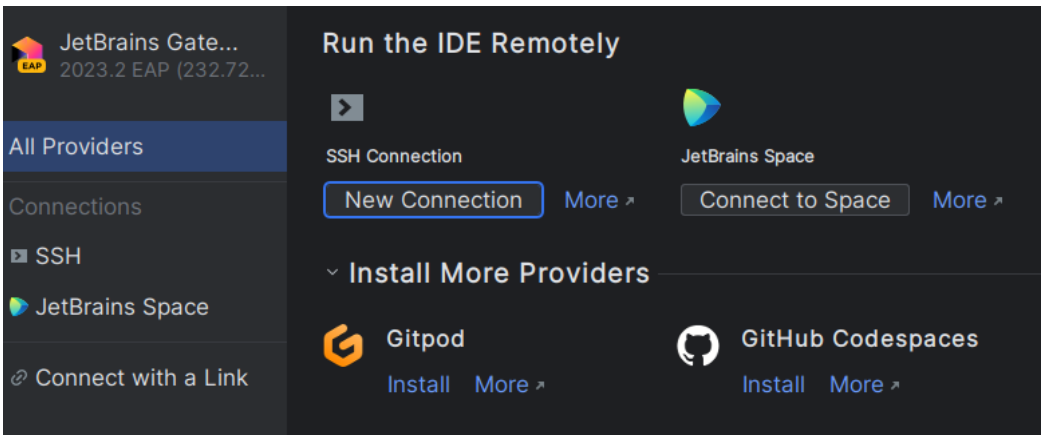


Ilustración 32. JetBrains Space - Selección de IDE

Nos conectamos al entorno y abre el editor PyCharm que tenemos instalado localmente con sus configuraciones y plugins.
Realizamos pruebas de ejecución del código de ejemplo Python y de escritura y se observa que el sistema sufre de un “input lag” bastante apreciable, al abrir el informe de la conexión el IDE nos da información en directo sobre la máquina remota que se está ejecutando y pese a no estar cargada e ir sobrada de recursos tiene una latencia entre 200 ms y 300 ms.

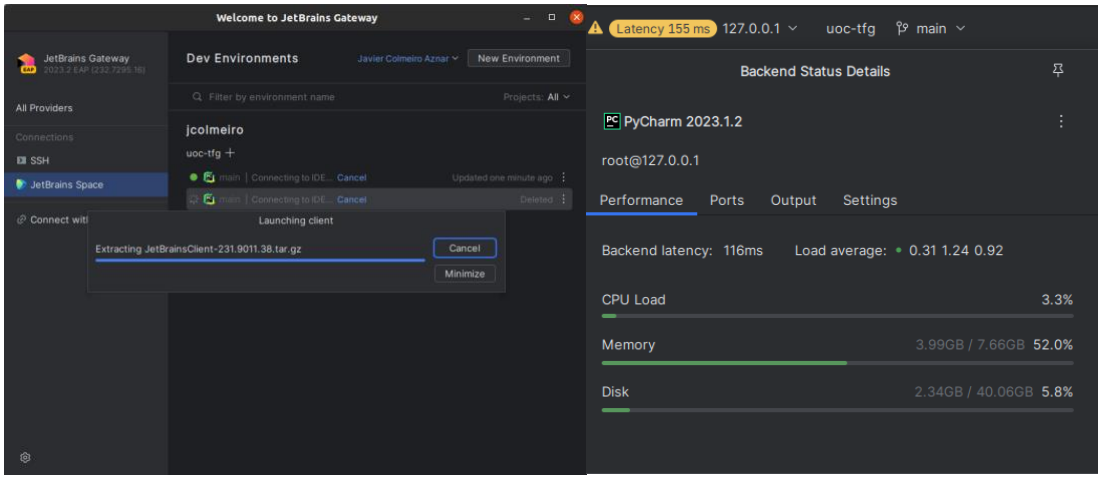
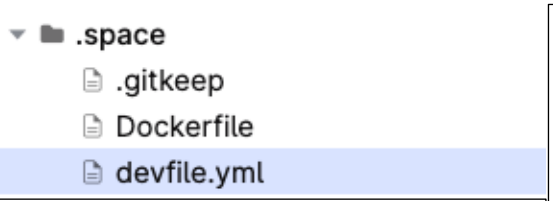


Ilustración 33. JetBrains Space - tiempos de carga y recursos

A continuación, realizamos una configuración manual del entorno de desarrollo. Para ello creamos el fichero devfile.yml dentro de la carpeta .space en nuestro repositorio con el siguiente contenido que define la configuración básica de que tipo de máquina usar, el editor de código y la imagen Docker a montar.




```yaml
schemaVersion: 2.2.0
attributes:
  space:
    instanceType: regular
    editor:
      type: PyCharm
      version: '2023.1.2'
      updateChannel: EAP
components:
- name: image-build
  image:
      imageName: myapp:latest
      dockerfile:
        uri: .space/Dockerfile
```

```
FROM ubuntu:20.04
ENV
DEBIAN_FRONTEND=noninteractive
ENV LC_ALL=C.UTF-8
RUN apt-get update && apt-get
install -y apt-utils apt-
transport-https
RUN apt-get install -y \
# Utilities \
curl unzip wget software-
properties-common socat man-db
gnupg2 pass lsof \
# VCS \
git \
# Docker
docker docker-compose \
# Required tools
cowsay
RUN apt update
RUN apt install postgresql
postgresql-contrib -f -y
```

Seguidamente crearemos un fichero Dockerfile con la imagen base que queremos utilizar, así como unos comandos iniciales básicos que nos instalar las

utilidades necesarias para instalar nueva paquetería y finalmente instalaremos el motor de base de datos postgresql.

Para terminar, eliminamos el entorno de desarrollo creado anteriormente y volvemos a crear uno nuevo. Esta vez el sistema detecta que tenemos ficheros de configuración específicos y compila el entorno de desarrollo en base a ellos. El tiempo de compilación ha sido similar para los dos casos cerca de los 2 minutos.

Como resumen final, JetBrains Spaces no es solo una plataforma de entornos de desarrollo remoto. Ofrece repositorios de código, gestión documental y entre otros, entornos de desarrollo. Permite utilizar cloud publica y privada con unos costes reducidos y 40 horas gratuitas de uso al mes. El interfaz gráfico es agradable aunque poco intuitivo y no permite modificar el código con un IDE online. El tiempo de compilación de los entornos es bueno pero el proceso de lanzar un entorno en el IDE local es un proceso largo con varios pasos y aplicaciones. Por último, la versión de su cloud pública sufre de una alta latencia que hacen que la experiencia de uso no sea adecuada.

Ideas clave de la solución evaluada:

¿Licencia de software? Privada
¿Tipo de infraestructura? Nube pública / Nube privada
¿Licencia de uso? Gratuita / de pago
¿Pago por uso? Si / No
¿Disponible en IDE local? No web, local mediante Gateway
¿Tecnología contenedores? Docker / Kubernetes
¿Fichero de configuración? Devfile.yaml
¿Docker-compose? Si
¿Dificultad de configuración? Media
¿Dificultad de uso? Media
¿Latencia de uso? Bastante
¿Tiempo de compilación del entorno de desarrollo? 2 minutos
¿Tiempo de inicio una vez compilado? 1minuto

4.4 Eclipse Che

La fundación eclipse es una organización creada originariamente por IBM en 2001 como un consorcio de vendedores de software. Su propósito fue el de promover el desarrollo del código libre y financiar y mantener un IDE llamado Eclipse. Con el tiempo, la fundación ha ido ampliando su espectro y en la actualidad apolla y financia multitud de proyectos entre los que se encuentra Eclipse Che, su aproximación a los entornos de desarrollo remotos en la nube.

Eclipse Che es una plataforma Opensource basada en Java y que permite mediante el uso de ficheros estándar Devfile crear la definición de entornos de desarrollo y su despliegue en una multitud de infraestructuras. Para funcionar, es necesario disponer de un cluster de Kubernetes y despliega en contenedores en formato Linux Containers (LXC) los 3 IDEs soportados VS Code, JetBrains y Eclipse Theia.

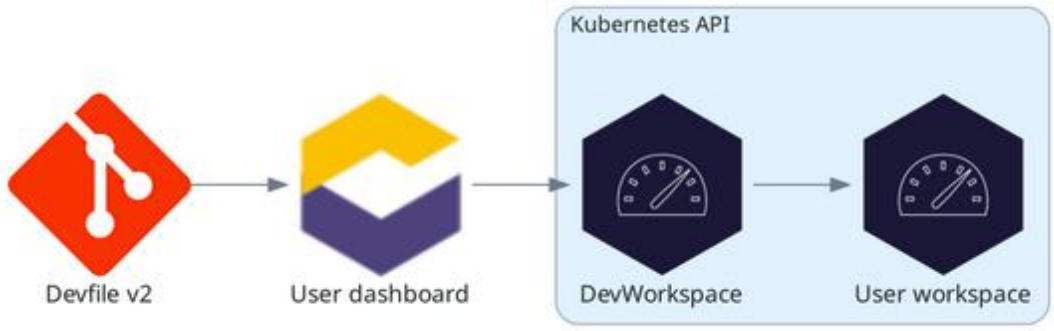


Ilustración 34. Eclipse Che – arquitectura

Eclipse Che no tiene coste asociados de por si ni de licencias ni de infraestructuras ya que tan solo proporcionan una plataforma para crear y planificar este tipo de entornos. Por este motivo no se puede probar como tal esta plataforma. Ofrece una versión de evaluación desplegada en su modalidad de Red Hat OpenShit Dev Spaces que se evaluará en la siguiente sección.

Aunque no se puede realizar la evaluación práctica de la solución directamente, podemos acceder a la extensa documentación donde dispone de muchas guías sobre todo enfocadas en su uso junto al cloud de Red Hat. Una de las secciones mas interesantes es la que enlaza a repositorio de código de che-incubator. En el podemos encontrar multitud de ejemplos y en especial, ficheros de configuración devfile.yaml que ayudan a comprender la sintaxis y como trabajar con ellos.

Para aprovecharnos de esta información se decide diseñar un devfile.yaml en base a estos ejemplos que nos permita disponer de un entorno de desarrollo

remoto con capacidades de compilar código Python y ejecutar el motór de base de datos PostgreSQL.

Ademas Eclipse es uno de los creadores, impulsores y mantenedor de la sintaxis de devfile por lo que tambien disponemos de buena documentación de estos ficheros en devfile.io/docs así ocmo un registro público de ficheros listos para utilizar.

Empezamos a configurar el fichero personalizado de la siguiente manera:

```
schemaVersion: 2.2.0
metadata:
  generateName: uoc-tfg red hat
attributes:
  controller.devfile.io/storage-type: ephemeral
```

En esta sección definimos la versión de esquema a utilizar, el nombre del contenedor y el atributo para utilizar almacenamiento efímero en vez de físico.

```
components:
  - name: tools
    container:
      image: quay.io/devfile/universal-developer-image:ubi8-latest
      volumeMounts:
        - name: m2
          path: /home/user/.m2
      memoryLimit: 6G
      memoryRequest: 512Mi
      cpuRequest: 1000m
      cpuLimit: 4000m
      mountSources: true
  - name: postgresql
    container:
      image:'quay.io/centos7/postgresql-13-
centos7@sha256:994f5c622e2913bda1c4a7fa3b0c7e7f75e7caa3ac66ff1ed70ccfe
65c40dd75'
      env:
        - name: POSTGRESQL_USER
          value: user
        - name: POSTGRESQL_PASSWORD
          value: password
        - name: POSTGRESQL_DATABASE
          value: food_db
        - name: PGDATA
          value: /tmp/pgdata
  - name: m2
    volume:
      size: 1G
```

A continuación, se definen 3 componentes, en el primero, definimos la imagen principal a utilizar, esta será la que contenga el entorno de desarrollo en si así como el editor de código. Podemos definir en esta también los limites de memoria, cpu y los volúmenes de disco que utilizaremos.
En el siguiente componente indicamos la configuración del contenedor que ofrecerá la base de datos. Por último definimos el contenedor para el almacenamiento y su límite de memoria.

```
commands:
  - id: run
    exec:
      label: "Run the application"
      component: tools
      workingDir: ${PROJECTS_ROOT}/python-hello-world
      commandLine: python -m venv .venv && . .venv/bin/activate && python
jcolmeiro_uoc_prime_numbers.py
      group:
        kind: run
  - id: deploypostgres
    exec:
      label: "Deploy Postrgres"
      component: tools
      workingDir: ${PROJECTS_ROOT}/uoc-tfg
      commandLine:    "oc    new-app    -e    POSTGRESQL_USER=user    -e
POSTGRESQL_PASSWORD=password       -e       POSTGRESQL_DATABASE=food_db
postgresql:10-el7 -n demo"
      group:
        kind: test
```

Por último, se definen 2 comandos que se ejecutaran cuando se compile el entorno de desarrollo. En el primero se ejecuta e inicializa el entorno virtual de Python y se lanza el fichero de ejemplo. En el segundo, se le indica al orquestador de OpenShift que cree una nueva aplicación de PostgreSQL e inicialice una base de datos con los datos indicados.

Con todo esto ya tendríamos una definición de entorno de desarrollo listo para ser usado en los servicios que fueran compatibles con devfile.yaml. En este casó se ha diseñado específicamente para la infraestructura PaaS de Red Hat OpenShift Dev Spaces y ejecutar el editor de código VS Code en su versión web.

Ideas clave de la solución evaluada:

¿Licencia de software? Opensource
¿Tipo de infraestructura? cluster Kubernetes.
¿Licencia de uso? Gratuita
¿Pago por uso? No
¿Disponible en IDE local? Si, Mediante extensión
¿Tecnología contenedores? Docker / Kubernetes
¿Fichero de configuración? Devfile.yaml
¿Docker-compose? Si
¿Dificultad de configuración? Alta
¿Dificultad de uso? Desconocida
¿Latencia de uso? Desconocida
¿Tiempo de compilación del entorno de desarrollo? desconocido
¿Tiempo de inicio una vez compilado? desconocido

4.5 Red Hat OpenShift Dev Spaces

Red Hat es una empresa americana de desarrollo de software fundada en 1993 y que se popularizó por ofrecer una distribución de Linux bajo la denominación de Red Hat Enterprise Linux. Este Linux estaba destinado a grandes empresas que necesitaban una capa extra de personalización y de soporte para sus servicios. En 2019 fue comprada por IBM aunque opera de forma independiente.

En 2011 irrumpieron en el mercado del PaaS con la comercialización de su producto OpenShift, un software de licencia comercial destinado al control y despliegue de contenedores basado en Kubernetes. Este software puede desplegarse en las cloud públicas de Azure, AWS o IBM así como en su cloud pública o bajo demanda. En 2019 publicaron su primera aproximación a los entornos de desarrollo remotos bajo el nombre de CodeReady Workspaces aunque en 2022 fueron renombrados a Openshift Dev Spaces para asociarlos a la marca orientada en servicios de contenedores.

Su servicio esta basado en Eclipse CHE y requiere de licencia de uso por cluster de Openshift así como de pago por uso de horas de computación si se utiliza su infraestructura privada. Estos costes se calculan por horas y son de $0,03 por hora para la licencia de uso y de 0,171 por hora de uso de computación de 4vCPU. En costes se traducirían en $1500 al año por un uso continuo de computación y de 263€ por la licencia de uso del cluster. Los costes por uso en otras clouds públicas como AWS son muy similares. A estos costes habrá que añadirle otros costes habituales que se pueda incurrir como costes por tráfico de datos, nodos, etc.

Para poder realizar las pruebas de campo, podemos crear una cuenta gratuita de OpenShift la cual ofrece una versión de evaluación con un crédito de uso ilimitado de tiempo de vCPU durante un mes. Para ello introducimos el correo, validamos los datos y se crea la cuenta de usuario.

Para poder hacer uso de los Red Hat OpenShift Dev Spaces podemos utilizar una cuenta sandbox de developer la cual nos permitirá crear entornos de desarrollo sin tener que configurar nada de la infraestructura inherente a este tipo de servicios cloud PaaS en OpenShift por lo que simplifica mucho el trabajo.

Para esto, hay que acceder a developer.redhat.com y se puede utilizar la versión web o un cliente nativo para nuestro sistema operativo. El interfaz web requiere que enlacemos nuestra cuenta de Red Hat y con este paso ya estará disponible para su uso.

Para comenzar con el Dev Space, podemos crear uno en base a una de las plantillas de uso que mantienen con los principales lenguajes de programación

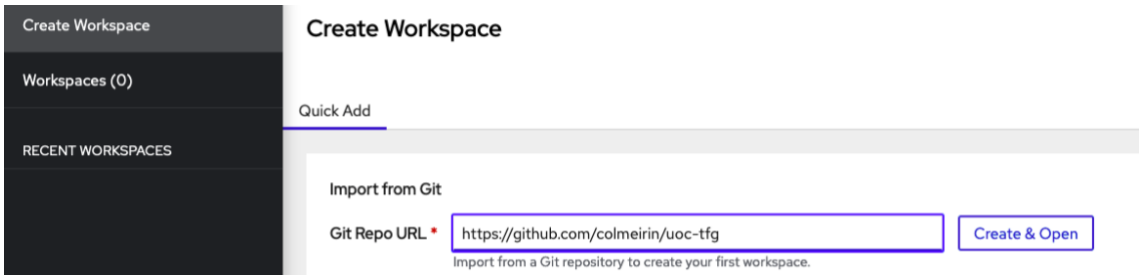


Ilustración 35. RedHat - Creación en base a repositorio

o podemos utilizar un fichero devfile.yaml específico. Como ya hemos diseñado un devfile.yaml válido cuando investigamos Eclipse CHE, podemos utilizarlo seleccionando la opción de importar desde repositorio. Introducimos la url del repositorio que creamos en el punto 3.1 y el sistema detectará el fichero y compilará el espacio.

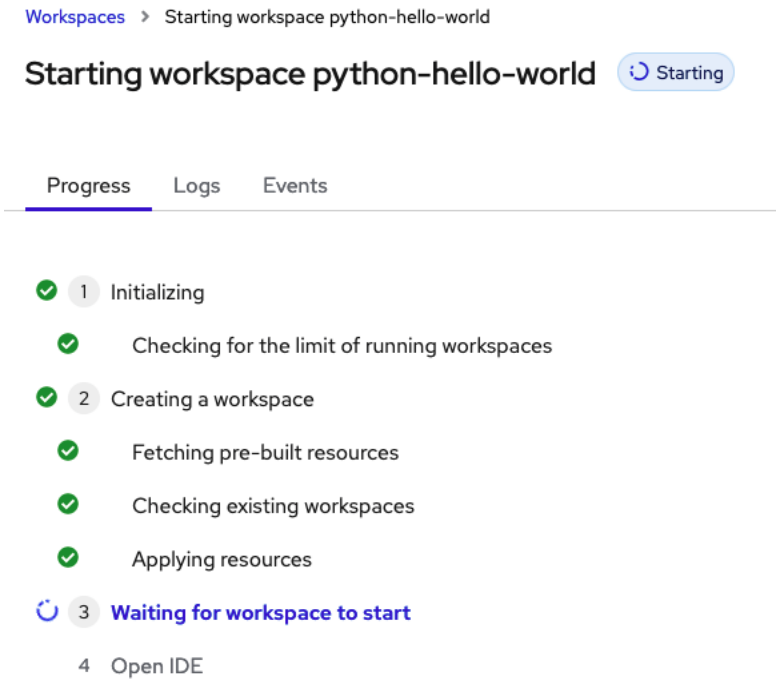


Ilustración 36. RedHat - construcción del contenidor

Una vez compilado, lanzará el IDE seleccionado, que en este caso es VS Code. El sistema responde bien, si latencia apreciable y es capaz de compilar el código Python de ejemplo.

Por último si queremos administrar el entorno de desarrollo, disponemos de una sección en la plataforma online de Red Hat OpenShift Dev Spaces donde podemos modificar manualmente el fichero Devfile importado o el Dev Workspace creado por Eclipse CHE.

Como resumen final, Red Hat OpenShift Dev Spaces es la forma mas sencilla de poder utilizar Eclipse Che en una nube. La versión de evaluación es sencilla de configurar y la interfaz de usuario es agradable. La compilación de los entornos puede tardar hasta 1 minuto y una vez compilado cuesta unos 20

segundos disponer de un entorno de desarrollo usable. Como puntos negativos, requiere de un pago de licencia de uso por cluster y también de pago por hora de computación en cualquiera de las modalidades cloud que ofrecen por lo que puede acarrear costes elevados de uso. La documentación sobre las operaciones es enorme pero confusa y cuesta encontrar la información concisa. Por último pese haber nacido como un proyecto Opensource en esta modalidad no dispone de licencia Opensource.

Ideas clave de la solución evaluada:

¿Licencia de software? Comercial
¿Tipo de infraestructura? Nube pública AWS, Azure, IBM o Red Hat
¿Licencia de uso? De pago por cluster
¿Pago por uso? Si
¿Disponible en IDE local? Si, Mediante extensión
¿Tecnología contenedores? OpenShift
¿Fichero de configuración? Devfile.yaml
¿Docker-compose? Si
¿Dificultad de configuración? Media
¿Dificultad de uso? Baja
¿Latencia de uso? Poca
¿Tiempo de compilación del entorno de desarrollo? 1 minuto
¿Tiempo de inicio una vez compilado? 20 segundos

4.6 Amazon CodeCatalyst Dev Environments

Amazon es una empresa americana que inicialmente se dedicaba a la venta de libros. Mas adelante tubo la idea de reutilizar los recursos extra que aprovisionaba para los picos de tráfico en su web y se ha convertido en el líder de la computación y servicios en la nube.

En diciembre de 2022 y bajo el paraguas de su servicio cloud AWS, presentó la herramienta Codecatalyst, un juego de herramientas y aplicaciones pensadas en diseñadores de software y orientada a cubrir todas sus necesidades para manejar el ciclo de vida del software. Esto incluye herramientas CI/CD, de repositorio de código, de gestión de tiques, aprovisionamiento de recursos y entornos de desarrollo en la nube. Este ultimo, llamado Codecatalyst Dev Environments es su apuesta en el entorno de los CDEs. Para ello hace uso de ficheros estándar devfile.yaml, y despliega en contenedores usando la infraestructura de AWS y clúster de Kubernetes.

Su modelo de negocio se basa en el pago por hora de computación. Dispone de una modalidad gratuita que permite crear un espacio de desarrollo por región geográfica de AWS y para cada espacio 60 horas de uso de Dev Environment en su modalidad 2 vCPU con 4GB de RAM y 16GB de almacenamiento. Esta modalidad no permite sobrepasar el limite ni pagar por uso por lo que habrá que cambiar la cuenta a la modalidad Standard.
En la modalidad Standard, que cuesta $4 por usuario al mes, se pueden crear 5 espacios de desarrollo por región de AWS y para cada espacio dispone de 200h de uso de Dev Envornment en modalidad de 2, 4, 8 y 16 vCPU, 4, 8, 16 y 32 GB RAM y 16, 32 y 64 GB de almacenamiento. Al superar los límites, se accede a la modalidad de pago por uso con un coste de $0,05/GB-mes por el almacenamiento extra y de $0.12/hora para el uso de máquinas 2 vCPU/4GB, $0.23/hora para 4 vCPU/8GB, $0.46/hora para 8 vCPU/16GB y $0.92/hora para 16 vCPU/32GB. Por ultimo, hay que tener en cuenta que se puede incurrir en costes extra por conceptos habituales de uso de las cloud como el pago por tráfico de datos.

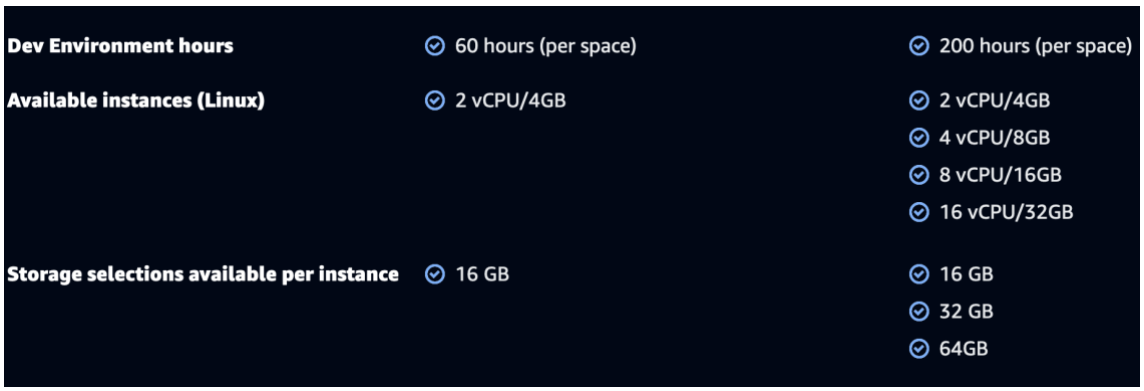


Ilustración 37. Amazon - características de cada licencia

Para comenzar con las pruebas, accederemos a la web de CodeCatalyst e iniciamos el registro gratuito.

1. Crear AWS Builder ID. Iniciamos el proceso, verificamos el email y asignamos una contraseña.

2. Crear alias CodeCatalyst space. Asignamos un alias para que se nos pueda reconocer en el sistema.

3. Crear un espacio. Creamos un espacio dándole un nombre.

4. Creamos una cuenta de AWS. Para esto seguimos el enlace proporcionado, verificamos el email, validamos la dirección postal, tarjeta de crédito, numero de teléfono y seleccionamos el nivel básico de soporte que es gratuito.

5. Esperar 24 horas a que la cuenta AWS se active.

6. Enlazar cuenta de CodeCatalyst con la cuenta de AWS. Introducimos el identificador de cuenta de AWS en el configurador de CodeCatalyst, este genera un token de verificación y lo pegamos en la consola de AWS en CodeCatalyst > Spaces > Verify Amazon CodeCatalyst space.

7. Crear un proyecto. El sistema tiene 3 opciones, utilizar un “blueprint” o plantilla de aplicación el cual esta pensado para traer todas las herramientas tipo de aplicaciones como aplicación .net, servicio REST, aplicación Node.js, etc. Como segunda opción, se puede elegir utilizar un repositorio de código existente. Como tercera opción se puede empezar un proyecto vacío y definir el espacio manualmente. Elegimos crear desde repositorio.

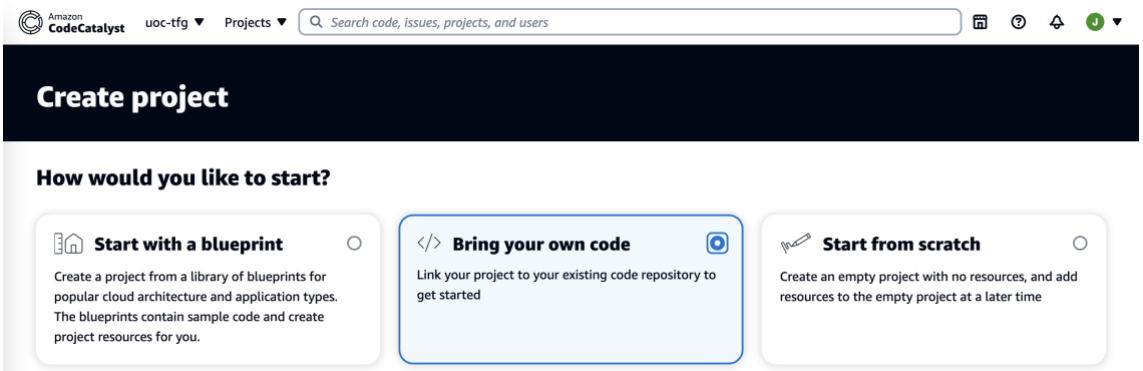


Ilustración 38. Amazon - Crear en base a repositorio

8. Vincular cuenta de GitHub. Vinculamos la cuenta de GitHub autorizando la aplicación y especificando a que repositorios tendrá acceso. Seleccionamos el repositorio creado anteriormente. Con este paso ya tenemos creado el proyecto.

9. Crear un Dev Environment. Para ello tenemos que elegir que editor de código se va a utilizar. Podemos seleccionar un editor web de Amazon u otros 4 famosos editores como VS Code o JetBrains.

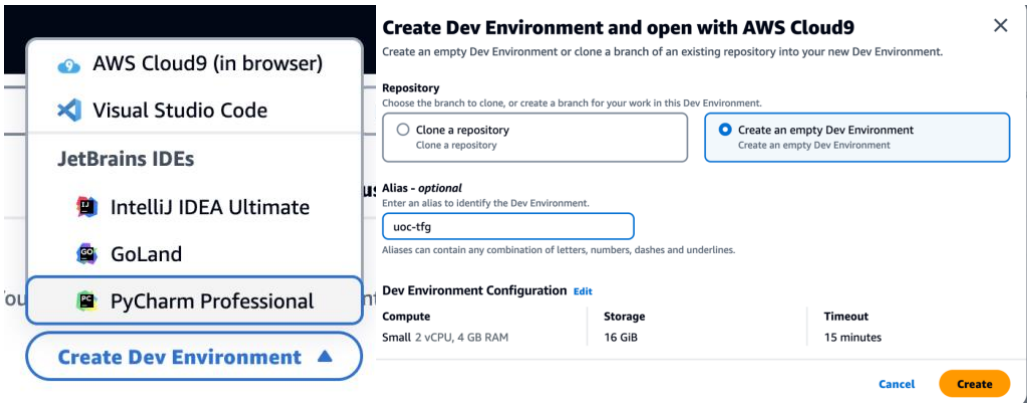


Ilustración 39. Amazon - Selección de IDE y creación

10. Usar el Dev Envornment. Para ello le damos a crear y esperamos 30 segundos a que se cree el contenedor. A continuación clicamos en el enlace y se abrirá el IDE seleccionado.

Una vez ejecutados los pasos se puede comprobar que funciona de forma similar al de JetBrains respecto a latencia y velocidad de respuesta.

Como resumen final, Amazon CodeCatalyst ofrece una solución de herramientas para desarrolladores que van mas allá de los entornos de desarrollo. Centrándonos en estos, la configuración es sencilla aunque larga y la creación, compilación y uso de los entornos es muy rápida. Permite utilizar editor web y locales y compartir las configuraciones de los entornos.
Como puntos negativos, la licencia no es Opensource, los entornos sufren una latencia bastante apreciable en ciertas operaciones, no se puede desplegar en otras nubes distintas a AWS y los costos de uso de esta son elevados.

Ideas clave de la solución evaluada:

¿Licencia de software? Privada
¿Tipo de infraestructura? Nube pública Amazon AWS
¿Licencia de uso? Gratuita / de pago
¿Pago por uso? Si
¿Disponible en IDE local? Si, Mediante extensión
¿Tecnología contenedores? Docker / Kubernetes
¿Fichero de configuración? Devfile.yaml
¿Permite Docker-compose? Si
¿Dificultad de configuración? Media
¿Dificultad de uso? Baja
¿Latencia de uso? Media
¿Tiempo de compilación del entorno de desarrollo? 30 segundos
¿Tiempo de inicio una vez compilado? 10 segundos

4.7 Coder

Coder es una empresa radicada en Estados Unidos. Nació en 2017 como una colaboración de 3 amigos que querían dedicarse a la monetización de plugins y servidores de Minecraft. Tras centrarse en las tecnologías pujantes de contenedores como Docker, en 2019 sacaron su producto Code-server como su aproximación para ofrecer una plataforma de gestión de entornos de desarrollo en la nube.

Inicialmente su objetivo era ofrecer un sistema que permitiera lanzar el editor de código VS Code en su versión web. Con el tiempo y al ser una empresa Opensource han ampliado la cantidad de IDEs compatibles y funcionalidades gracias a las aportaciones de la comunidad. Su tecnología se basa en Terraform como plataforma para aprovisionar recursos y gestionar despliegues en cualquier infraestructura en base a ficheros de configuración declarativos en formato Json y con el lenguaje HashiCorp. La ventaja frente al uso directo de Terraform es que no es necesario tener conocimientos complejos de este gracias a su interfaz de usuario y operaciones automatizadas.

Su modelo de negocio se basa en las licencias Enterprise. Estas ofrecen una capa de funcionalidades extra y soporte sobre la aplicación como gestionar grupos de permisos complejos, poder establecer cuotas de uso sobre recursos o configurar sistemas de alta demanda. Además, ofrecen su soporte y conocimientos para realizar despliegues complejos y distribuidos en múltiples capas y nubes distintas.

Respecto a los costes, Coder de por si no tiene ningún coste asociado ya que ofrecen su tecnología de gestión y administración de forma libre y sin licencias de uso. No disponen de infraestructura propia por lo que no se puede incurrir en costes de pago por uso de computación. Respecto a la licencia Enterprise no es posible obtener la información de costes de forma inmediata ya que se trata de un embudo de ventas pensado para organizaciones de mas de 50 empleados y con un proceso previo de análisis de las necesidades y requisitos específicos.

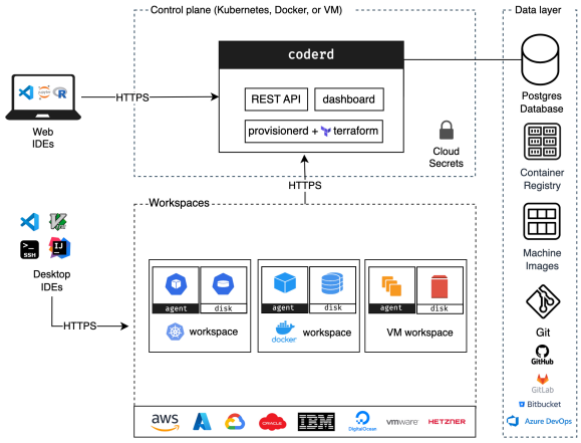


Ilustración 40. Coder – arquitectura

Para las pruebas de Coder, utilizaremos la infraestructura de nube privada creada anteriormente en Proxmox y lvm. Para ello tendremos que resolver primeramente las dependencias especificas necesarias para su instalación.
Instalamos el servidor de bases de datos PostgreSQL ya que es uno de los requisitos iniciales de Coder.

```
apt-get update && apt-get install -y postgresql
service postgresql start
```

A continuación, instalamos el servicio de Coder, creamos un usuario para ejecutarlo y lo lanzaremos.

```
curl -L https://coder.com/install.sh | sh
sudo adduser uoc
sudo usermod -aG sudo uoc
sudo usermod -aG docker uoc
sudo chmod 666 / var/run/docker.sock
su – uoc
code server
```

Al consultar la url de acceso nos encontraremos con la pantalla inicial donde podemos registrar la cuenta de administración del servicio.

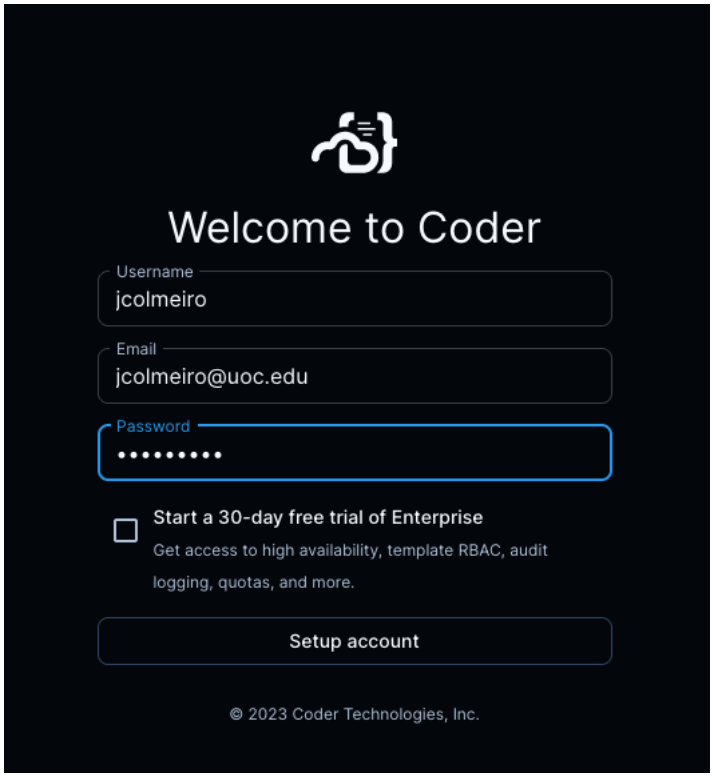


Ilustración 41. Coder - registro cuenta administrador

A continuación accedemos al sistema y creamos un nuevo espacio de trabajo, *workspace* basado en una *template* pre-configurada.

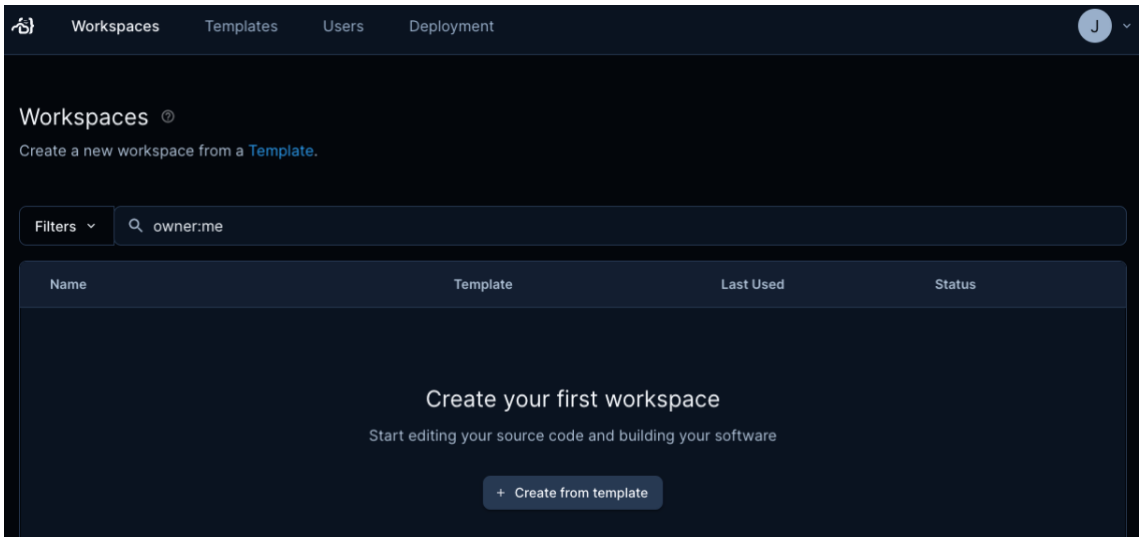


Ilustración 42. Coder - creación de entorno

Observamos que el sistema permite elegir como tipos de despliegue Docker, Kuberntes así como el uso de *clouds* públicas.

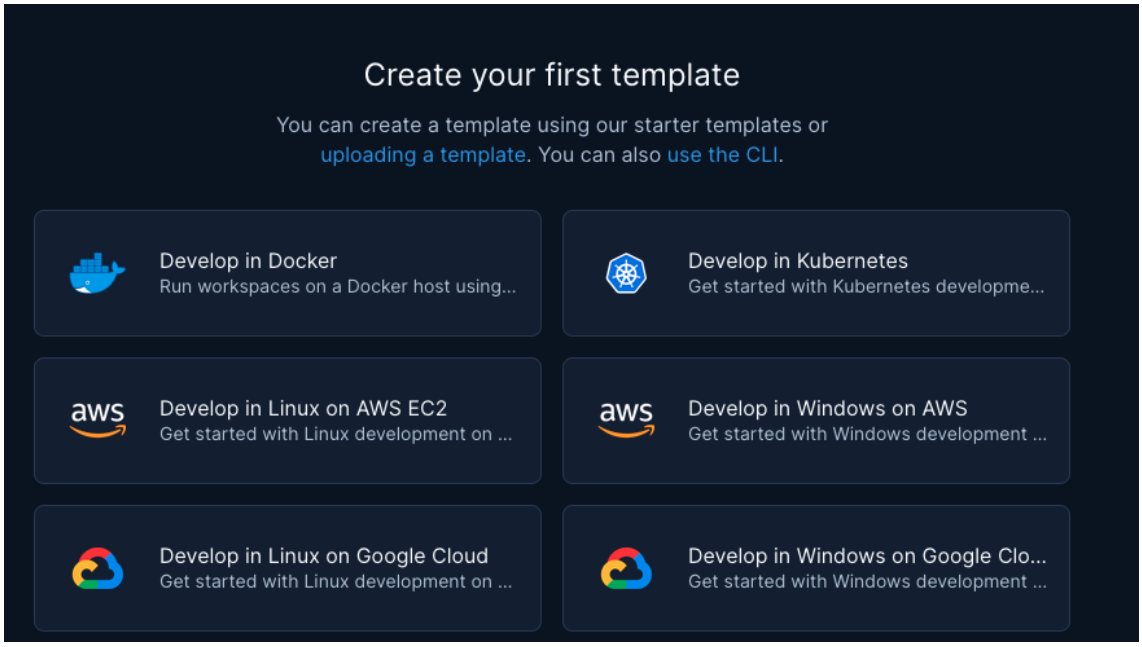


Ilustración 43. Coder - selección de infraestructura

Seleccionamos Docker y obtenemos las instrucciones de configuración de la template y procedemos introducirlas en la consola para inicializarlas. En estas inicializaremos una plantilla pensada en ser ejecutada en Docker y el sistema verificará que disponemos de permisos de administración mediante la generación de un token y la validación de este vía interfaz web.

```
coder templates init
coder login http://localhost:3000
cd ./docker && coder templates créate
```

Una vez completados estos pasos, dispondremos de una template recién creada para poder crear un workspace personal.

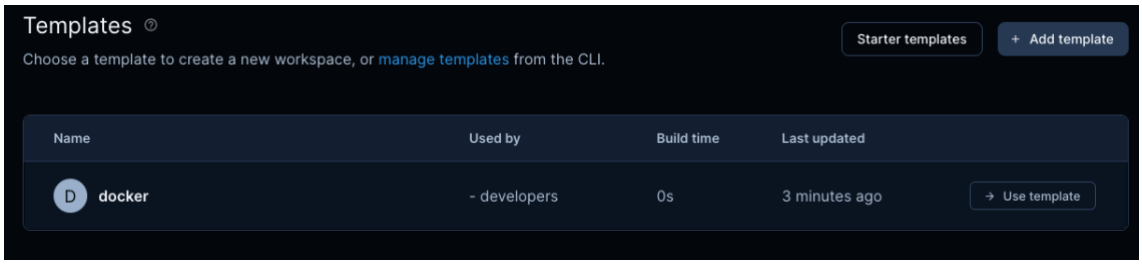


Ilustración 44. Coder - Creación de template

Creamos el workspace introduciendo los datos de nombre y del usuario al que lo asignamos.

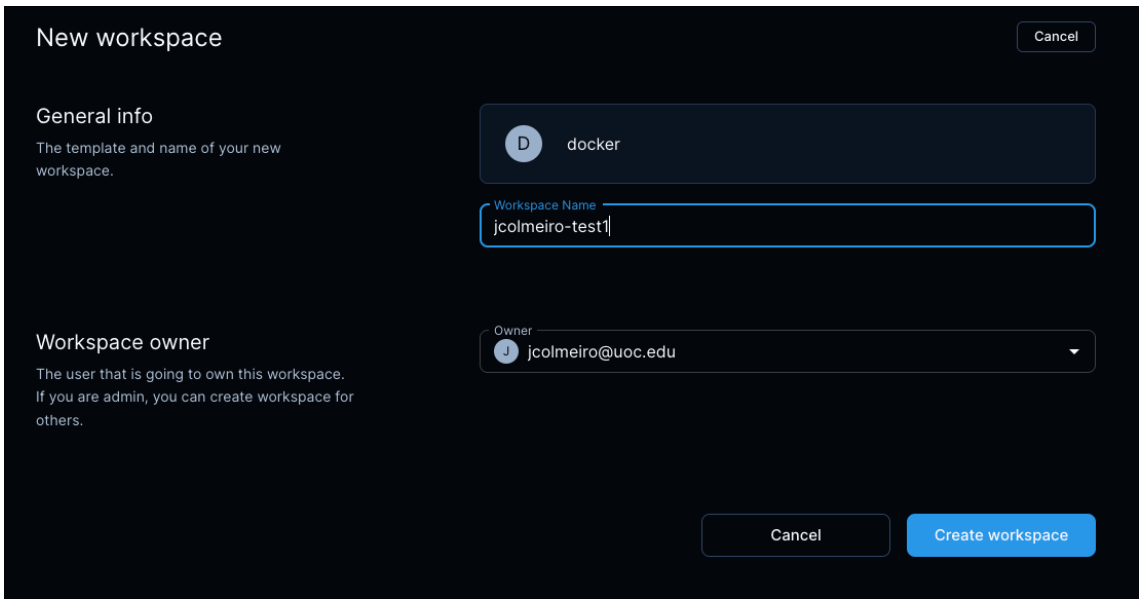


Ilustración 45. Coder - creación de entorno

Una vez creado, se construye automáticamente en un contenedor Docker en segundos y se inicializa. A partir de este momento, podemos abrir el elegir el espacio de trabajo a utilizar y dentro de este escoger el editor de código que queremos utilizar.

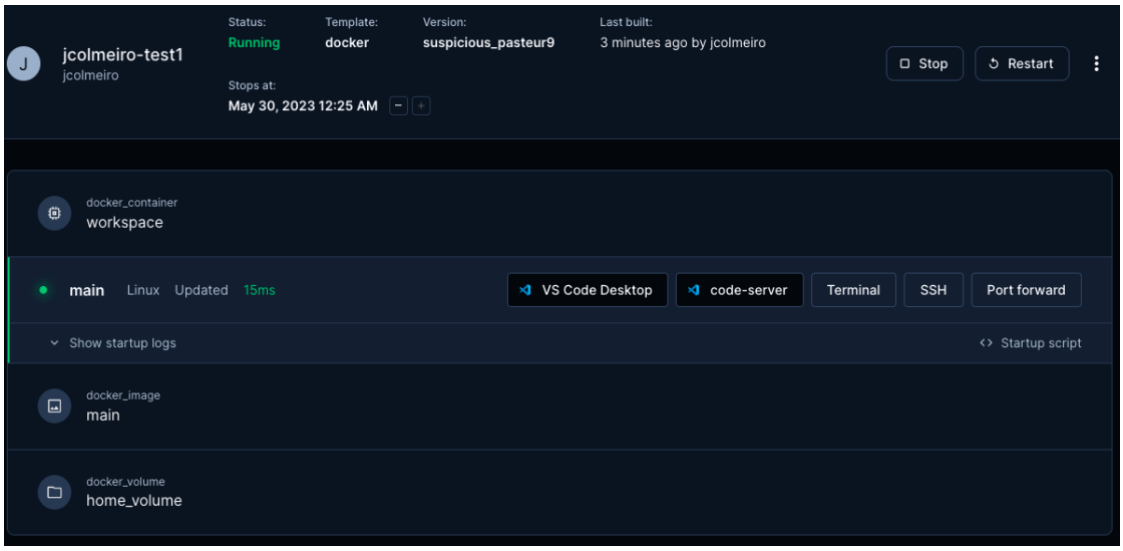


Ilustración 46. Coder - lanzador de entorno

Si elegimos la opción VS Code Desktop para editar el código de forma local, la primera vez pedirá instalar una extensión para poder utilizarlo.

Si seleccionamos code-server, nos iniciará VS Code en su versión web integrada por lo que no necesitaremos tener nada instalado en el dispositivo de acceso.

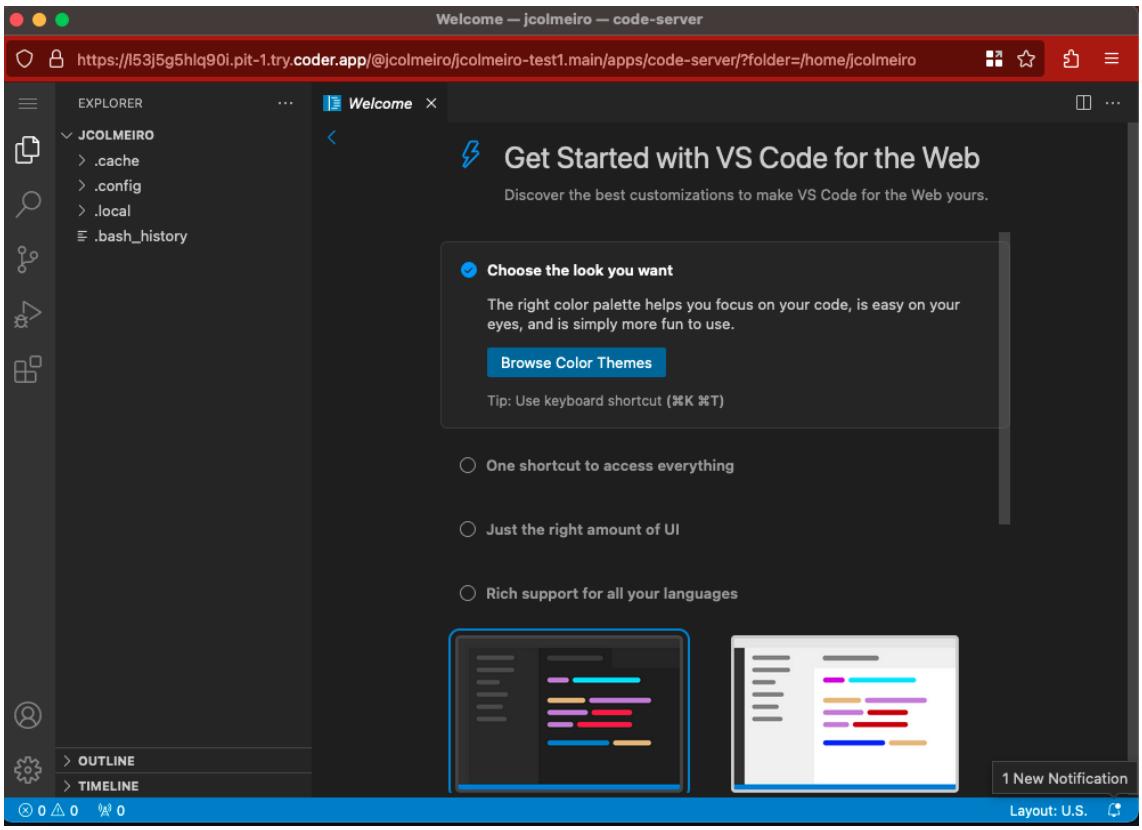


Ilustración 47. Coder - entorno ejecutándose

Finalmente, también permite conectarse por terminal o por ssh obteniendo una experiencia de uso como si se accediera a un servidor directamente.

Ilustración 48. Coder - consola del entorno

Como resumen final, Coder es una plataforma sencilla y Opensource que permite gestionar y administrar entornos de desarrollo remoto y desplegarlos en múltiples y diferentes infraestructuras. Además dispone de una gestión completa de usuario y permisos de acceso así como de personalización de las plantillas de los entornos. El sistema es sencillo y fácil de usar y los entornos desplegados

tienen una buena latencia. Como puntos negativos, no utiliza el standard de ficheros devfile.yaml sino docker-compose directamente, lo cual por otra parte es una ventaja en caso de ya disponer de estos ficheros creados.

Ideas clave de la solución evaluada:

¿Licencia de software? Opensource
¿Tipo de infraestructura? Nube pública, privada, servidor o local
¿Licencia de uso? Gratuita
¿Pago por uso? No
¿Disponible en IDE local? Si
¿Tecnología contenedores? Docker
¿Fichero de configuración? Docker-compose.yaml
¿Docker-compose? Si
¿Dificultad de configuración? Baja
¿Dificultad de uso? Baja
¿Latencia de uso? Poca
¿Tiempo de compilación del entorno de desarrollo? 5 minutos/
¿Tiempo de inicio una vez compilado? 10 segundos

4.8 CodeAnywhere

Codeanywhere nació en Croacia como empresa que se dedicaba a la creación entornos de desarrollo integrado (IDE) especializado en PHP. Inicialmente, su solución se basaba en un editor de código programado en Javascript y accesible por navegador. Además integraba un servicio FTP para poder subir el código a un servidor y ejecutarlo. Con la evolución y popularización de los servicios en la nube en 2022 publicó su aproximación a los entornos de desarrollo remoto por subscripción.

Su tecnología se basa en OpenVZ OS Containers que son contenedores Linux LXC, VEs o VPSs y se despliega sobre su propia infraestructura aprovisionando servidores virtuales privados con capacidad de proceso, memoria RAM y espacio en disco. Permite el uso de plantillas pre-configuradas para la mayoría de los lenguajes de programación (90 lenguajes) así como de motores de bases de datos (24 motores de bases de datos) .

Su modelo de negocio se basa en el pago por suscripción y dispone de 3 niveles distintos:

Basic permite tener 1 contenedor de 2GB de RAM y 10GB de almacenamiento por $6 al mes.

Standard permite tener 3 contenedores de 4GB de RAM y 15GB de almacenamiento por $15 al mes.

Premium permite tener 6 contenedores de 8GB de RAM y 20GB de almacenamiento por $40 al mes.

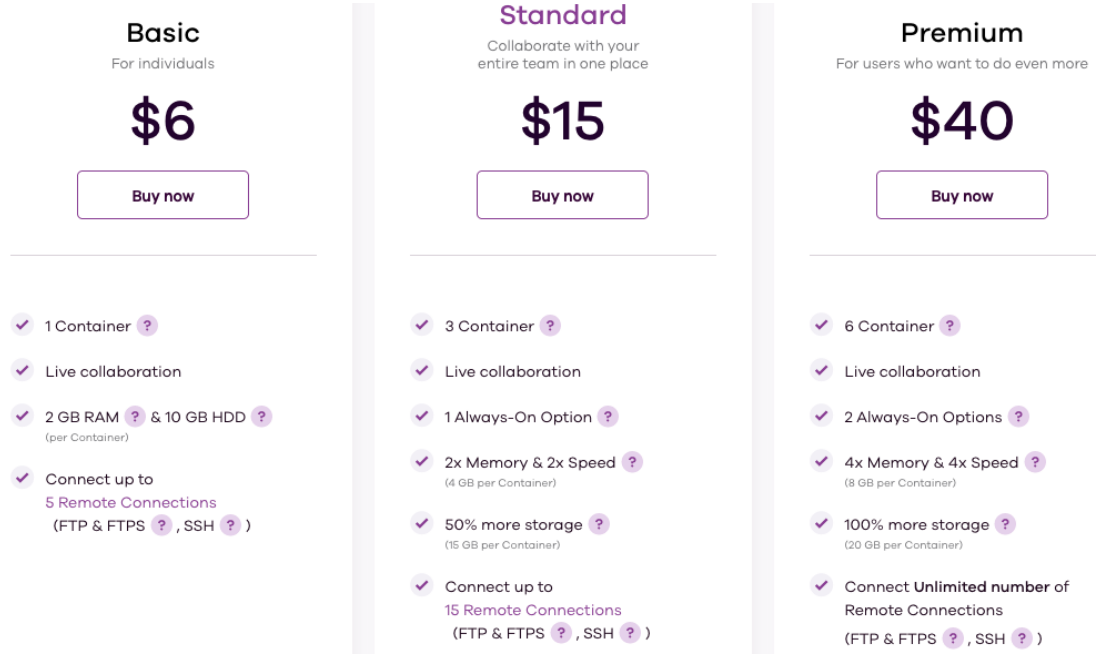


Ilustración 49. CodeAnywhere - precios y características de licencias

No tiene costes por hora de uso de computación o de almacenamiento.
Para probar la plataforma, nos registraremos con el correo de la universidad y directamente accederemos al panel donde se pueden crear los espacios de desarrollo. Accedemos a New Container y nos permite elegir una de las plantillas pre-compiladas disponibles.

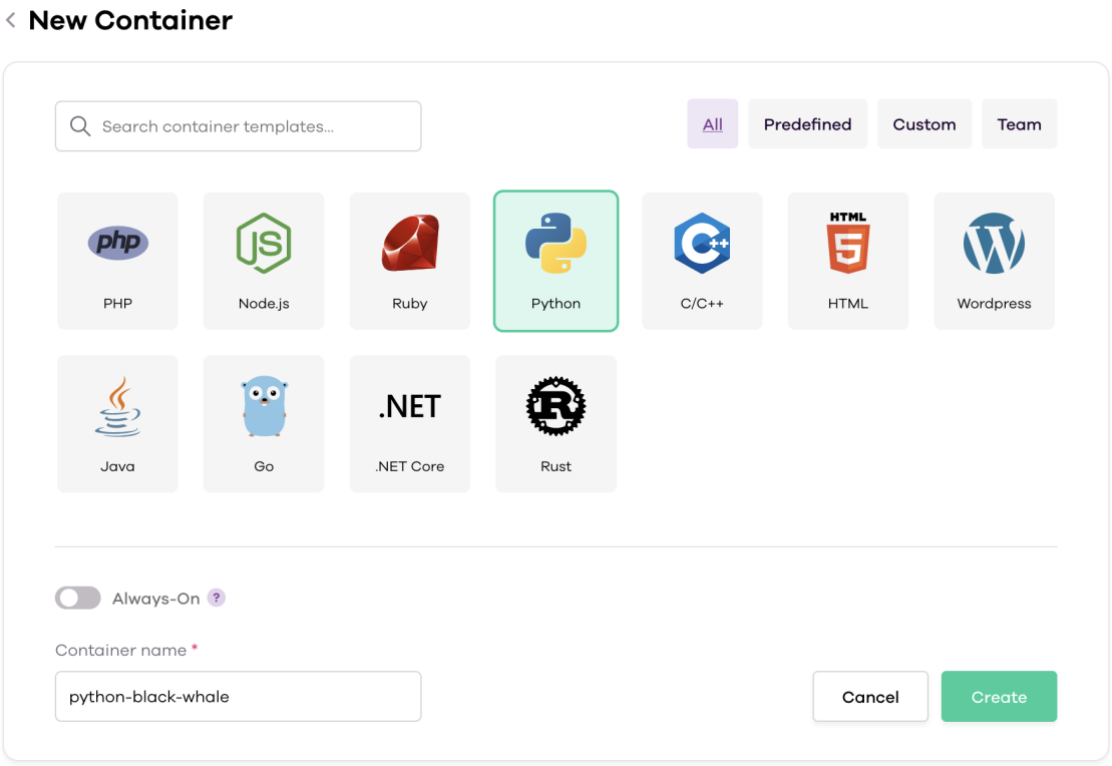


Ilustración 50. CodeAnywhere - selección de plantilla

Seleccionamos la de Python y le damos a crear y en 10 segundos tenemos el contenedor creado. A continuación accedemos a Open IDE y en 20 segundos abre la versión web de VS Code. Nos permite crear un nuevo proyecto o trabajar con un proyecto alojado en un repositorio Git. Para esta prueba seleccionamos el repositorio creado anteriormente y en 5 segundos lo ha descargado y lo tenemos listo para codificar y ejecutar.

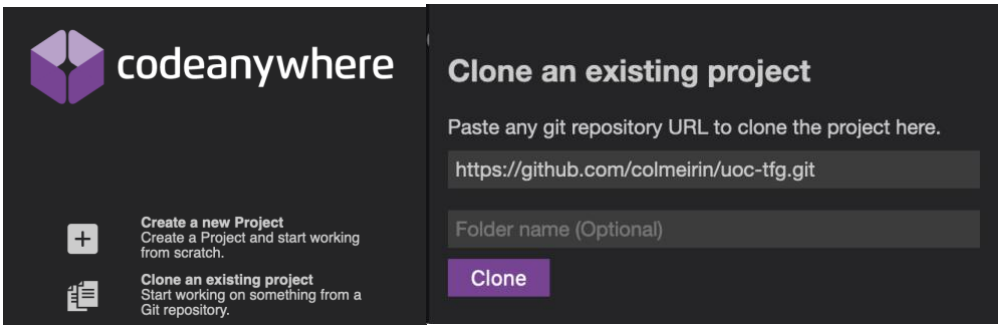


Ilustración 51. CodeAnywhere - creación a partir de repositorio

La plataforma ha creado un entorno de desarrollo con Python 3.8, pyenv para las versiones locales de Python y pip preinstalado. Disponemos de acceso sudo y

acceso a la terminal por ssh y por HTTP y Websocket Ports. Nos genera un enlace de acceso con el puerto 3000 expuesto donde podemos acceder a lo que se este ejecutando en el entorno, además también genera otro enlace público donde podemos acceder a cualquier puerto remplazando una sección de la url.
Si queremos utilizar el motor de bases de datos PostgreSQL hay que instalarlo como si se tratara de un servidor Linux cualquiera. Ejecutamos el siguiente comando para averiguar que distribución de Linux se esta ejecutando:

```
cat /etc/issue
Ubuntu 16.04 LTS
```

Observamos que esta ejecutando un Linux Ubuntu 16.04 el cual fue publicado en 2016 y podemos considerar que está bastante anticuado. En la documentación podemos encontrar los comandos recomendados de instalación para 8 motores distintos de bases de datos, elegimos PostgreSQL e introducimos los siguientes comandos para realizar la instalación:

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
sudo service postgresql status
```

Con ello el sistema instala la versión 9.5 de PosrgreSQL la cual coincide en el tiempo con la versión de Ubuntu ya que también fue publicada en 2016 y esta bastante desactualizada.

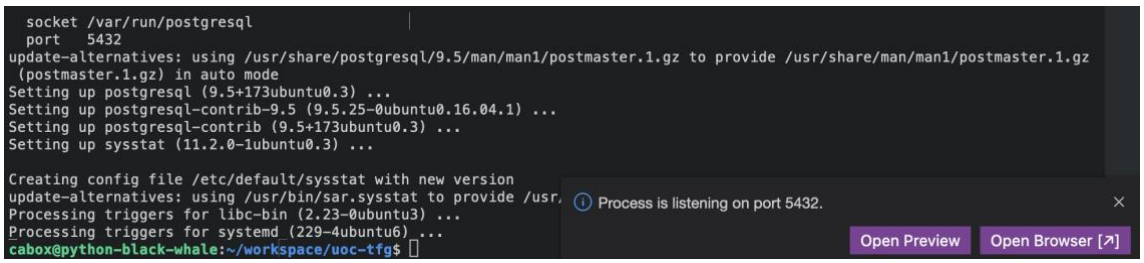


Ilustración 52. CodeAnywhere - instalación manual PostgreSQL en entorno

Con esto el sistema detecta que se encuentra disponible el servicio en el puerto por defecto y te permite ejecutarlo.

Por ultimo, en el panel de configuración podemos crear una nueva plantilla de contendor en base a las configuraciones que ya hemos hecho en esta y recrearla cuando queramos. También permite gestionar miembros del equipo y conexiones externas como SSH o Google Drive.

Como resumen final, Codeanywhere es una plataforma de entornos de desarrollo remoto muy sencilla de utilizar y configurar. Simplifican los costos con 3 tipos de licencia y el interfaz de usuario es agradable y sencillo. Los contenedores se crean de forma sencilla y rápida.
Como puntos negativos, solo permite utilizar los contenedores predefinidos sin docker-compose ni ficheros de configuración yaml o json. La potencia de los contenedores va asociada a la licencia por lo que si queremos mas puntualmente

habrá que incrementar la licencia de forma permanente.

Ideas clave de la solución evaluada:

¿Licencia de software? Privada
¿Tipo de infraestructura? Infraestructura de VM/VPS propietaria Codeanywhere
¿Licencia de uso? De pago
¿Pago por uso? No
¿Disponible en IDE local? No
¿Tecnología contenedores? OpenVZ OS Containers
¿Fichero de configuración? No
¿Docker-compose? No
¿Dificultad de configuración? Baja
¿Dificultad de uso? Baja
¿Latencia de uso? Poca
¿Tiempo de compilación del entorno de desarrollo? 10 segundos
¿Tiempo de inicio una vez compilado? 20 segundos

4.9 Microsoft Dev Box

Azure **Microsoft Dev Box**

Microsoft es una empresa que originariamente nació para el desarrollo y comercialización de sistemas operativos. Con el tiempo, la empresa ha migrado parte de su modelo de negocio al Cloud y la computación distribuida con su producto Azure y dentro de este contexto en agosto de 2022 presentaron una versión preliminar de su idea de entornos de desarrollo remotos e iniciaron pruebas de uso en un numero reducido de empresas. En mayo de 2023 anunciaron que ya estaba disponible de forma generalizada su producto Microsoft Dev Box en modo *preview* con características limitadas por lo que se evalúa también en este trabajo.

Su apuesta por los entornos de desarrollo remotos se basa en una tecnología propia desplegada en su nube pública y asociada a los centros de desarrollo de Azure. Permite configurar 2 tipos de máquinas y elegir una imagen base con Windows 11 o 10 y VS Code. De momento no se permite imágenes personalizadas y estos entornos son básicamente un sistema operativo Windows en la nube con aplicaciones de desarrollo de código y auxiliares instaladas. Por todo esto, podemos afirmar que ofrecen sus servicios como un PaaS.

Su modelo de negocio se basa en el pago por hora de computación y se tarifica directamente por euro hora. Actualmente, la tarificación para Europa permite elegir entre una máquina de 4vCPU y 16GiB de memoria por 0,50€ la hora o 8 vCPU y 32GiB de memoria por 0,999€ por hora. Incluyen 15 horas de uso gratuitas mientras se esta evaluando. Por otra parte, también requiere de un pago por uso de almacenamiento, teniendo en este área, 3 opciones. 256, 512 y 1024 GiB de disco SSD a 0,054 €, 0,107 € y 0,213 € por hora con 365 horas gratuitas por mes en la versión de evaluación.

Dev Box Compute

	Price
4 vCPU, 16 GiB Memory	**€0.50** per hour
8 vCPU, 32 GiB Memory	15 hours - free during preview (per month) **€0.999** per hour

Dev Box Storage

	Price
SSD 256 GiB	**€0.054** per hour
SSD 512 GiB	365 hours - free during preview (per month) **€0.107** per hour
SSD 1024 GiB	**€0.213** per hour

Ilustración 53. Microsoft - cotes por uso

Además requiere que todos los usuarios finales de estos entornos dispongan de una licencia activa de Windows 10/11 Enterprise, Microsoft Endpoint Manager y Azure Active Directory P1.

Al estar asociado al cloud de Azure, el proceso de configuración no está aislado e involucra conocimientos de gestión de esta. Así pues, ara poder configurarlo, hay que crear una cuenta gratuita en Azure y validarla con una tarjeta de crédito. Seguidamente hay que incrementar la cuenta al tipo de pago por uso para desbloquear todos los límites. A continuación, tenemos que seguir los siguientes pasos para poder obtener una Microsoft Dev Box desde cero:

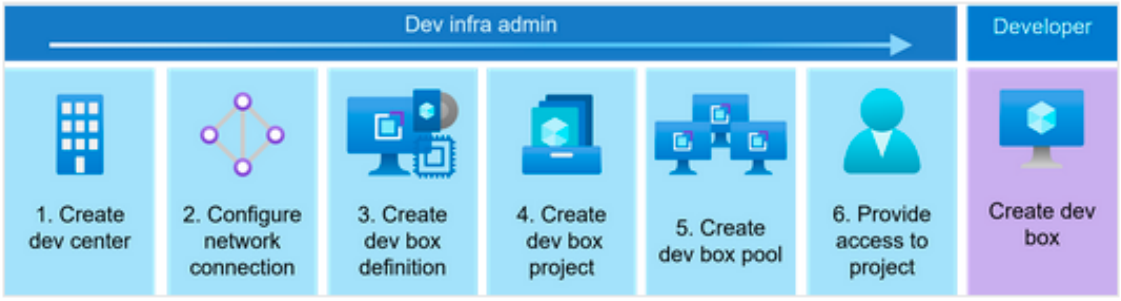


Ilustración 54. Microsoft - proceso resumido para crear entorno

1. Crear un centro de desarrollo. Para ello introducimos en el buscador “centro de desarrollo” lo abrimos y le damos a crear. En el asistente de creación tenemos que denominar el centro y asignar una región geográfica.

2. Crear una red virtual y subred. Para ello introducimos en el buscador “red virtual” y creamos una con los datos por defecto.

3. Crear una conexión de red. Para ello introducimos en el buscador “configuración de red” y creamos una con los datos por defecto en base a la red virtual anteriormente creada.

4. Asociar conexión de red a centro de desarrollo. Para ello introducimos en el buscador “centro de desarrollo”, accedemos al centro de desarrollo anteriormente creado y agregamos la conexión de red.

5. Crear la definición de un equipo de desarrollo. Seleccionamos Definiciones de equipo de desarrollo y creamos una nueva.

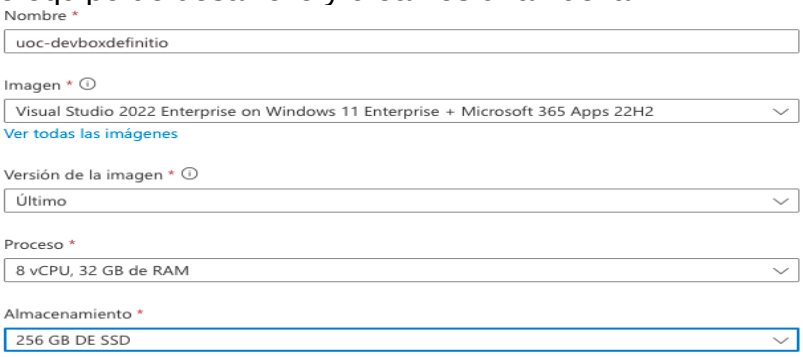


Ilustración 55. Microsoft - creación plantilla recursos

6. Crear un proyecto en la definición del equipo de desarrollo. Seleccionamos la definición de equipo de desarrollo y creamos un nuevo proyecto

7. Crear un grupo de equipos de desarrollo. Seleccionamos grupo de equipo de desarrollo y creamos uno nuevo

8. Crear un tipo de entorno. Seleccionamos la definición del equipo de desarrollo y creamos un nuevo tipo de entorno, llamado DEV.

9. Crear un token de acceso personal para repositorio. En GitHub, accedemos a la configuración de nuestra cuenta, y luego a la configuración de desarrollador y creamos un nuevo token de acceso personal que establece permisos de lectura a nuestro repositorio de GitHub creado anteriormente.

10. Crear un almacén de llaves. En el buscador introducimos “almacén de llaves” y creamos uno nuevo.

11. Crear un secreto para el token de acceso personal de GitHub. Creamos un nuevo secreto e introducimos el secreto del token generado anteriormente y configuramos los permisos de acceso a el para que este disponible.

12. Crear un catalogo. En el buscador introducimos “centros de desarrollo” y accedemos al creado anteriormente. Accedemos a Catálogos y creamos uno nuevo. Introducimos un nombre y la url pública de GitHub de nuestro repositorio, establecemos la rama main e introducimos la url de identificador secreto creado en el paso anterior.

Ilustración 56. Microsoft - creación catálogo en base a repositorio

13. Crear identidad para el centro de desarrollo. En el centro de desarrollo accedemos a Configuración > Identidad y activamos el parámetro de estado en Asignado por el sistema.

14. Crear asignaciones de control de acceso IAM y RBAC. Accedemos al almacén de llaves creado anteriormente y en el control de acceso (IAM) agregamos 2 roles distintos a nuestro usuario DevCenter Dev Box User y Administrador de Key Vault y un rol de Identidad administrada a la identidad creada anteriormente con el rol de Usuario de secretos de Key Vault.

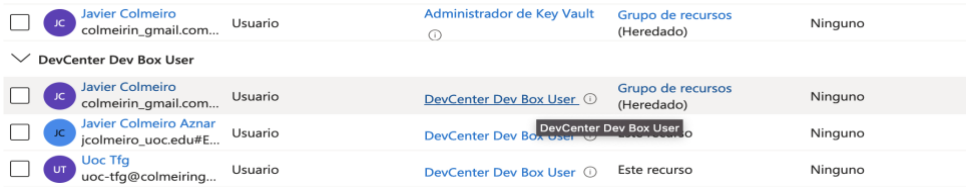


Ilustración 57. Microsoft - Creación de roles y permisos en Azure

15. A continuación accedemos al portal de desarrolladores de Microsoft en https://devportal.microsoft.com/ y añadimos una nueva dev box seleccionando las configuraciones de proyecto y dev box pool creadas anteriormente y le damos a añadir. Con esto comenzará el proceso de creación del entorno de desarrollo el cual puede tardar de 30 a 90 minutos según la carga del sistema.

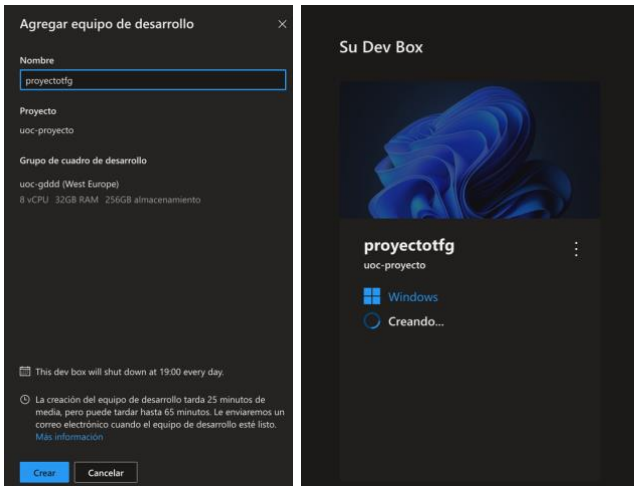


Ilustración 58. Microsoft - Creación de máquina virtual

Una vez compilado el entorno, podemos acceder al dev box via web o via Windows remote desktop ya que se trata de una máquina virtual que ejecuta un Windows 11 directamente. El acceso es lento y en primera instancia tiene una duración exagerada ya que se esta aplicando las configuraciones iniciales de todo el sistema. Una vez inicializado la latencia de uso es aceptable aunque en algún momento flaquea. Esto puede ser ocasionado por la región geográfica en la que se ubique el nodo de Azure.

Como resumen final, Microsoft ofrece una solución bastante limitada para acercarse a los entornos de desarrollo remotos. Como puntos a favor, esta integrada en el cloud de Azure el cual ya existe en muchas organizaciones. Como puntos negativos, técnicamente estos entornos tienen que ser ejecutados bajo Windows y solo permiten utilizar el editor de código VS Code. Además los costes por hora de computación son elevados y cada usuario debe disponer de licencias válidas de Windows 365. El proceso de configuración es largo y confuso y va asociado a conocimientos de configuración del cloud de Azure. Por último, no permite la parametrización de las imágenes usadas así como el uso de imágenes propias o externas ni tampoco alojarlo en una nube privada.

Ideas clave de la solución evaluada:

¿Licencia de software? Privada
¿Tipo de infraestructura? Nube pública Azure
¿Licencia de uso? De pago
¿Pago por uso? Si
¿Disponible en IDE local? No
¿Tecnología contenedores? VMs

¿Fichero de configuración? No
¿Docker-compose? No
¿Dificultad de configuración? Dificil
¿Dificultad de uso? Media
¿Latencia de uso? media
¿Tiempo de compilación del entorno de desarrollo? 35minutos
¿Tiempo de inicio una vez compilado? 60 segundos

4.10 Docker Dev Environment

Docker es un proyecto de código abierto que nació en 2013 y esta programado en Go. Utiliza tecnologías de virtualización del kernel de Linux basada en LXC (Linux Containters) para aislar y empaquetar las aplicaciones de una forma concisa y permitir ejecutrarlas de forma aislada así como partirlas de forma sencilla. Esta idea ha revolucionado la administración de sistemas y de aplicaciones con la idea de los microservicios y hoy en día prácticamente todas las aplicaciones de software disponen de una versión empaquetada de esta manera lista para ser utilizada y desplegada en cualquier infraestructura o nube pública en cuestión de segundos.

Para definir las características del microservicio o de aplicaciones mas complejas, hace uso de un fichero de texto plano llamado Dockerfile. Este fichero se utiliza para describir las características de la imagen empaquetada mediante unas palabras clave como FROM, ENV, COPY, USER, CMD, RUN,etc y así especificar las características así como las acciones de inicio a ejecutar que terminan de configurar la máquina.

Por otra parte, disponen de una herramienta llamada Docker Compose que permite definir ejecuciones de aplicaciones multi-contenedor de la misma manera. Esta hace uso de lenguajes de IaC (Infraestructure as Code) como YAML que almacena en un fichero llamado docker-compose.yml. En este fichero se describen los servicios con sus imágenes Docker, los volúmenes, ficheros de configuración, certificados o redes a utilizar De esta forma, con un simple comando podemos desplegar una aplicación que tenga distintas partes aisladas y configuradas pero que permita comunicarse entre si.

La mayoría de los proveedores de entornos de desarrollo remoto utilizan Docker como la tecnología de despliegue de dichos entornos. Así pues, Docker, como proyecto, ha decidido también ofrecer su visión sobre este paradigma y ha desarrollado el concepto de los Dev Environment, el cual actualmente se encuentra en modo Beta y en total desarrollo ya que se prevvee que en pocos meses liberen una nueva versión sobre este tema.

El cliente de escritorio de Docker nos permite configurar y empaquetar un entorno para desplegar de forma local o llevar a una nube o infraestructura propia. No requiere de ningún tipo de licencia de software ni pago por uso ya que simplemente permite con un asistente realizar la configuración.

Para iniciar el proceso, entramos en “create a Dev Environment”, asignamos un nombre e indicamos que queremos utilizar un repositorio Git. De esta manera el sistema será capaz de analizar el contenido del repositorio y crear una imagen de desarrollo acorde a los lenguajes de programación utilizados en el.

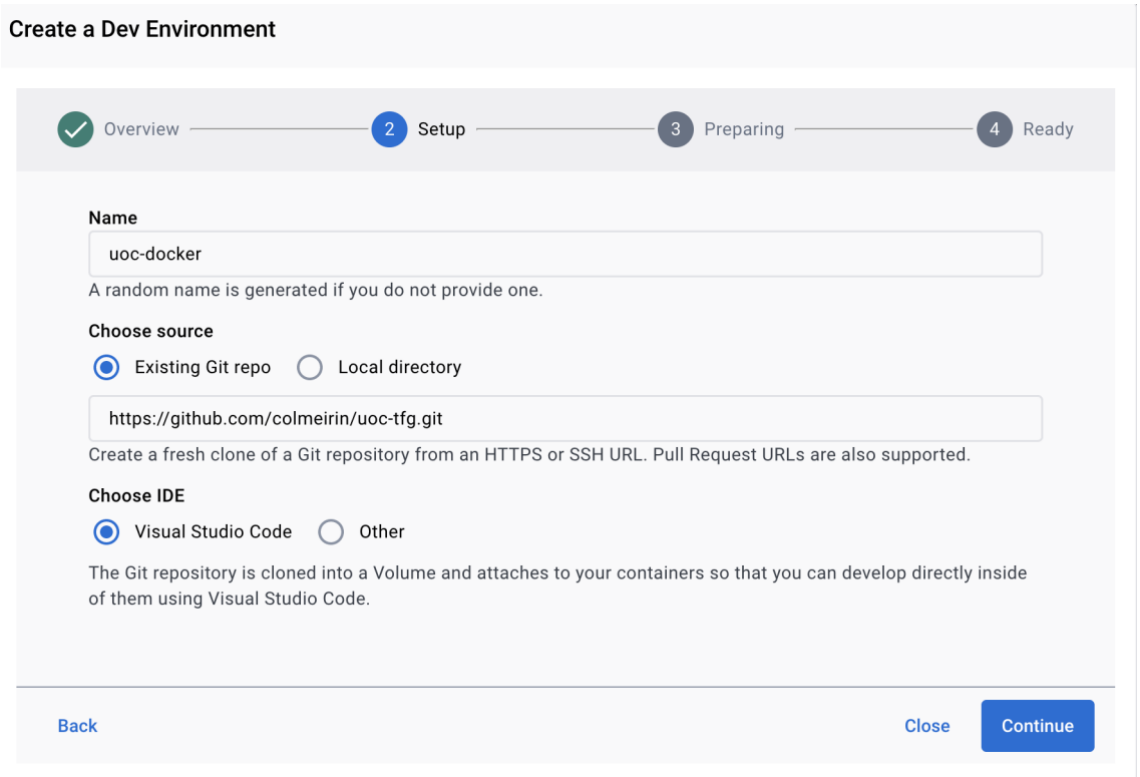


Ilustración 59. Docker - creación de entorno en base a repositorio

Inicialmente no existe fichero de configuración especifico en el repositorio por lo que el proceso termina creando un contenedor listo para utilizarse

Ilustración 60. Docker - entornos creado listo para ser lanzado

Al abrirlo, encontraremos un fichero de configuración llamado compose-dev.yaml el cual tendrá la definición básica del contenedor de desarrollo que ejecuta VS Code. Con esto ya podemos lanzar el editor de código y programar. A continuación se modifica el fichero anteriormente citado para añadir las imágenes de PostgreSQL y PgAdmin para disponer del motor de bases de datos.

Al recrear el Dev Environment, esta vez el Docker Compose crea los tres contenedores y los inicializa disponiendo de el entorno de desarrollo que cumple con todos los requisitos inicialmente planteados

Ilustración 61. Docker - Entorno creado con docker-compose

Una vez terminada la configuración, Docker dispone de una url o endpoint donde añadiendo la url del repositorio donde hemos almacenado el fichero de configuración, lanzará directamente el cliente de Docker local y compilará los contenedores. Por ejemplo, para el repositorio creado anteriormente con sus configuraciones la url será: https://open.docker.com/dashboard/dev-envs?url=https://github.com/colmeirin/uoc-tfg

Como resumen final, Docker ofrece una herramienta de configuración de entornos de desarrollo remoto en contenedores fácil y sencilla. Esta es capaz de leer el contenido de repositorios remotos y preparar las características necesarias. Además si existen ficheros de configuración específicos los leerá y aplicará. La interfaz es limpia y sencilla y permite ejecutar el editor de código incluido o lanzar uno local cualquiera.
Como puntos negativos, esta solución no está directamente integrada en la nube, y habrá que crear toda la infraestructura que permita gestionarlo y desplegarlo de forma remota así como definir la parametrización de usos de recurso, red, gestión de usuarios, etc.

Ideas clave de la solución evaluada:

¿Licencia de software? Opensource
¿Tipo de infraestructura? Cualquiera.
¿Licencia de uso? Gratuita
¿Pago por uso? No
¿Disponible en IDE local? Si
¿Tecnología contenedores? Docker / Kubernetes
¿Fichero de configuración? Compose-dev.yaml
¿Docker-compose? Si
¿Dificultad de configuración? Media
¿Dificultad de uso? Baja
¿Latencia de uso? Poca
¿Tiempo de compilación del entorno de desarrollo? 2 minutos
¿Tiempo de inicio una vez compilado? 30 segundos

5. Comparativa y elección.

5.1 Licencia de software, infraestructura de despliegue y tecnología.

Iniciamos la comparativa y resumen de las soluciones centrados en el tipo de licencia de software que disponen, el tipo de infraestructura válida para el despliegue y la tecnología de contenedores que utiliza cada una.

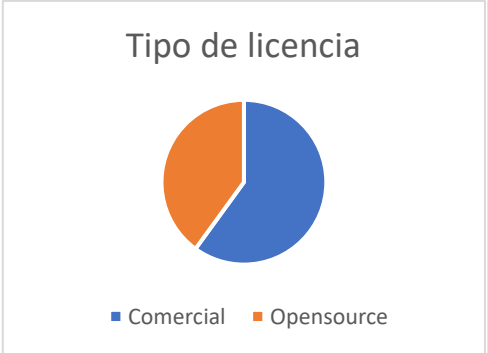
Plataforma	Licencia	Tipo de infraestructura	Tecnología
Amazon Dev Environments	Comercial	Nube pública AWS	Docker / Kubernetes
Codeanywhere	Comercial	Nube Codeanywhere	OpenVZ
Coder	Opensource	Nube, servidor o local	Docker
Docker Dev Environment	Opensource	Nube, servidor o local	Docker
Eclipse Che	Opensource	Nube pública / privada	Kubernetes
Git Hub Codespaces	Comercial	Nube pública Git Hub	Docker / Kubernetes
Gitpod	Opensource	Nube pública Gitpod	Docker / Kubernetes
JetBrains Spaces	Comercial	Nube pública / privada	Docker / Kubernetes
Microsoft Dev Box	Comercial	Nube pública Azure	VMs
Red Hat Dev Spaces	Comercial	Nube pública	OpenShift

Ilustración 62. Comparativa - licencia, nube, tecnología

En este aspecto observamos que el 60% de las soluciones evaluadas disponen de una licencia comercial todas asociadas a uno de los gigantes de la nube pública. Estos se basan en su gran mayoría en tecnologías Opensource pero han decidido no liberar sus productos a la comunidad. Respecto a los actores del Opensource, Docker Dev Envirnoments se encuentra en plena refactorización de su solución por lo que es posible que en un futuro cercano presenten una solución actualizada y mas en la línea del resto de soluciones.

Respecto al tipo de infraestructura se aprecia que la mayoría de las soluciones comerciales están pensadas para ser desplegadas en su propia infraestructura. En este aspecto son una excepción las soluciones de JetBrains y de Red Hat que pese a estar diseñadas para ser desplegadas en su infraestructura, permiten utilizar otras pagando la licencia de uso.

Respecto a la tecnología inherente de contenedores, Docker y Kubernetes dominan el espectro con un 80% de ellas si tenemos en cuenta que OpenShift internamente usa Kubernetes con Eclipse CHE.



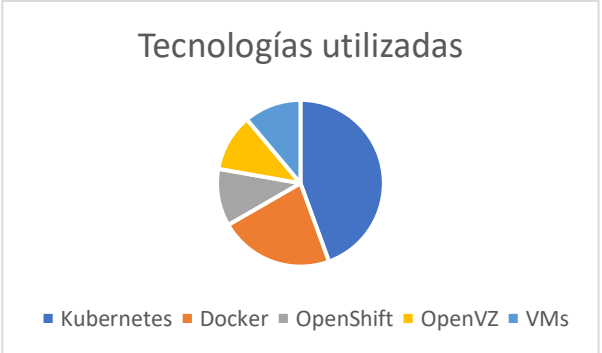


Ilustración 63. Comparativa - usos de licencia y tecnologies

5.2 Licencia de uso, coste mensual y coste de hora de computación.

A continuación se analizan los costes inherentes al uso de cada una de las plataformas, estos se separan en coste de licencia de usuario y de coste por hora de computación. Se han tomado los datos de referencia para el tipo de máquina mas habitual e inicial que se compone de 4vCPU y 8GB de RAM. En la mayoría de los casos si queremos aumentar la potencia, simplemente multiplicaremos por 2 el coste por hora de computación en cada salto.

Además hay algunas soluciones que disponen de cuentas gratuitas con mas o menos recursos pero en esta comparativa se ha tomado de base la licencia de pago inicial. Por último, cabe destacar que el coste de la licencia de uso de Red Hat Dev Spaces es en realidad por cluster de despliegue, por lo que será independiente al numero de usuarios.

Plataforma	Licencia de uso	Licencia / mes	Pago por uso	Coste / hora
Amazon Dev Environments	Gratuita/de pago	4	Si	0,12
Codeanywhere	De pago	15	No	0
Coder	Gratuita	0	No	0
Docker Dev Environment	Gratuita	0	No	0
Eclipse Che	Gratuita	0	No	0
Git Hub Codespaces	Gratuita/de pago	4	Si	0,19
Gitpod	Gratuita/de pago	9	Si	0,036
JetBrains Spaces	Gratuita/de pago	10	Si / No	0,41
Microsoft Dev Box	De pago	32,4	Si	0,55
Red Hat Dev Spaces	De pago	22	Si	0,17

Ilustración 64. Comparativa - costes licencias y computación

Se aprecia que las soluciones Opensource no suelen tener asociada un licencia de uso y como el despliegue es independiente, no se pueden computar los costes de cálculo por hora.

Dentro de las soluciones de pago, Amazon, Gitpod y GitHub son las mas baratas y con costes de coste por uso mas reducido. Además, Gitpod es la que mas horas de computación gratuita ofrece en este nivel de licencia por lo que es posible que no se incurriera en costes extras para un uso normal.

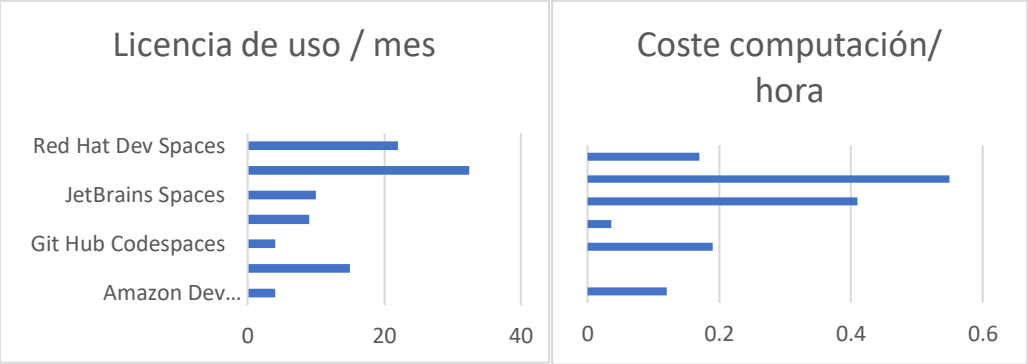


Ilustración 65. Costes de licencia de uso y computación

Microsoft es la solución mas cara tanto en licencia de uso como de coste de computación. No obstante, esta licencia incluye Windows 365 y Office 365 así como almacenamiento en la nube y una multitud de aplicaciones y esta integrada en la nube de Azure por lo que la comparativa de precio de forma aislada referente a los entornos de desarrollo es injusta.

Por último, CodeAnywhere ofrece una tarifa plana mensual que aunque parezca elevada incluye todas las horas de computación que se requieran y es la que tiene un interfaz de usuario y de configuración mas sencillo y directo. De hecho para entidades educaciones ofrece una licencia reducida de $1,5 y esta opción es utilizada por varias universidades incluyendo la Universidad de Yale o la Universidad de Hardvard.

5.3 Formato fichero configuración, docker-compose y uso de IDE local.

En esta sección analizamos las características técnicas que ofrecen las distintas soluciones respecto a los ficheros de configuración el uso de la tecnología de docker-compose y la capacidad de ser ejecutados en un IDE local.

Plataforma	Formato fichero	Docker-compose	IDE local
Amazon Dev Environments	Devfile.yaml	Si	Si
Codeanywhere	No	No	No
Coder	docker-compose.yaml	Si	Si
Docker Dev Environment	docker-compose.yaml	Si	Si
Eclipse Che	Devfile.yaml	Si	Si
Git Hub Codespaces	devcontainer.json	No	Si
Gitpod	gitpod.yaml	Si	Si
JetBrains Spaces	Devfile.yaml	Si	No
Microsoft Dev Box	No	No	Si
Red Hat Dev Spaces	Devfile.yaml	Si	Si

Ilustración 66. Comparativa - ficheros, docker-compose e IDE local

Inicialmente, se observa que CodeAnywhere y Microsoft Dev Box se encuentran

en un segmento del mercado totalmente distinto al resto ya que ofrecen sus entornos como una solución cerrada y que no permite configuración y herramientas externas. Esto es debido a que priman el uso de su editor de código online propietario y que las configuraciones se realicen de forma local como si se tratara de una máquina virtual y luego se utilicen estas como plantillas para crear otras.

Respecto a las soluciones que si permiten ficheros de configuración, se aprecia que hay una clara tendencia, 40%, a utilizar el estándar de Devfile.yaml promovido por la Cloud Native Computing Foundation, CNCF y apoyado por los grandes actores de la industria del cloud. Seguidamente se encontraría el uso de ficheros docker-compose.yaml apoyados por Dockerfile que utilizan el paradigma de configuración estándar de contenedores Docker.

Por último, hay algunos actores como GitHub y Gitpod que han optado por promover el uso de sus propios ficheros de configuración. Estos son híbridos entre los ficheros de docker-compose y los ficheros de Devfile con sus particularidades especificas en cada uno de ellos.

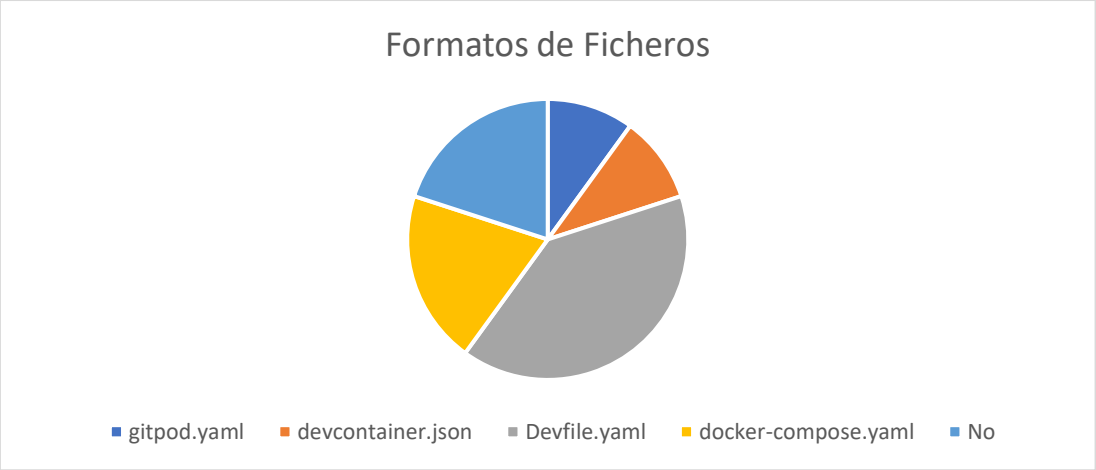


Ilustración 67. Comparativa - tipos de ficheros

5.4 Dificultad de configuración y uso, tiempos de carga y latencia.

Por último, en esta sección se comparan los resultados obtenidos en las pruebas de validación práctica de cada una de las soluciones, analizando la dificultad de configuración, la dificultad de uso, la latencia de uso, el tiempo de construcción de los entornos de desarrollo, así como del tiempo de inicio del entorno de desarrollo una vez construido y arrancado.

Plataforma	Configuración	Uso	Latencia	Construcción	Inicio
Amazon Dev Environments	Media	Baja	Media	30	10
Codeanywhere	Baja	Baja	Poca	10	20
Coder	Baja	Baja	Baja	300	10
Docker Dev Environment	Media	Baja	Poca	120	30
Eclipse Che	Alta	Media		0	0
Git Hub Codespaces	Baja	Baja	Poca	1500	10
Gitpod	Baja	Baja	Poca	420	5
JetBrains Spaces	Media	Media	Alta	120	60
Microsoft Dev Box	Alta	Media	Media	2100	20
Red Hat Dev Spaces	Media	Baja	Poca	60	20

Ilustración 68. Dificultad configuracion y uso, latencia y tiempos

Se aprecia que Microsoft Dev Box y Eclipse Che tienen la mayor dificultad de configuración, esto es debido a que Microsoft Dev Box esta asociado a requerir una configuración completa del cloud público Azure, el cual involucra una cantidad enorme de pasos y no dispone de una versión simplificada para su uso como es el caso de Amazon con su AWS o RedHat con OpenShift. Respecto a Eclipse Che la dificultad alta radica en que requiere de un cluster de Kubernetes y de una cantidad de recursos de hardware y de almacenamiento basta elevado. El resto de las soluciones se encuentran en un nivel de dificultad de configuración similar. Cabe destacar la solución de CodeAnywhere por su propuesta sencilla y desatendida y la solución de Coder que pese a requerir de una infraestructura propia los pasos a seguir para su configuración son pocos y sencillos.

Respecto a la dificultad de uso por parte del usuario final, prácticamente todos incluyen una interfaz de acceso sencillo, claro y actual y las operaciones de arranque del entorno de desarrollo así como de mantenimiento son simples.

Referente a la latencia de uso, la mayoría de las soluciones tienen una respuesta aceptable excepto JetBrains que sufre de un retardo de entrada o inputlag muy elevado.

Respecto a los tiempos de construcción de los entornos de desarrollo la mayoría de las soluciones son capaces de construirlo en menos de 5 minutos. En este aspecto cabe destacar la solución de CodeAnywhere que es la mas rápida ya que los entornos están pre compilados. En la opciones de despliegue de nube pública destaca Amazon como la mas rápida y GitHub y Microsoft como los mas lentos, tardando este último mas de 35 minutos.

Por último respecto al tiempo de arranque del entorno de desarrollo una vez construido, la mayoría de las soluciones ofrecen tiempos cerca de los 20 segundos. En este área cabe destacar Gitpod por ser la que mas se desmarca con un tiempo de arranque de 5 segundos.

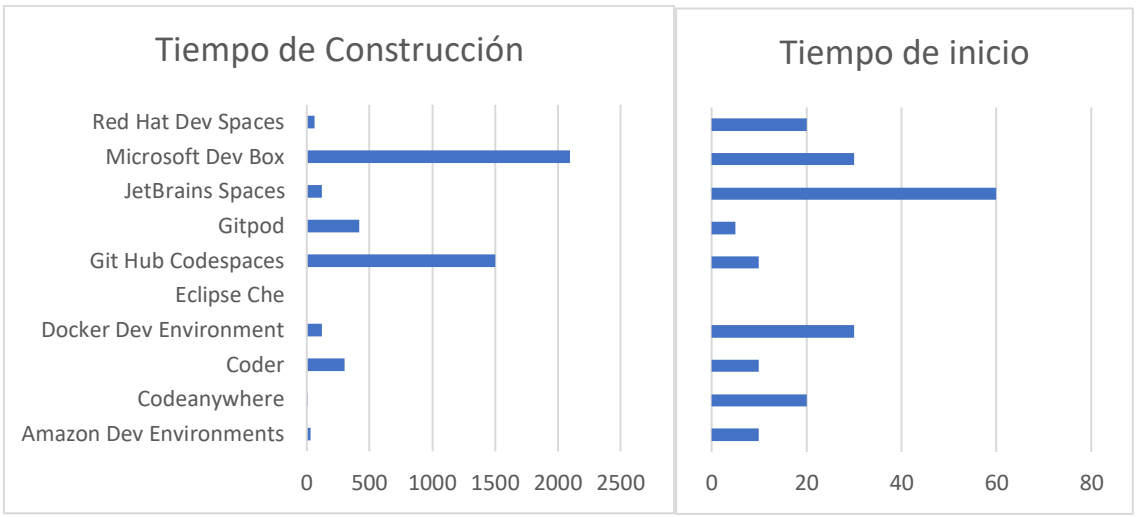


Ilustración 69. Comparativa - tiempos de creación y de arranque

5.5 Requisitos, elección y justificación de la solución en el contexto educacional

Para la elección de plataforma que se debería utilizar en una organización educacional se ha tenido en cuenta el contexto inicial planteado en el trabajo, el cual quiere ofrecer entornos de desarrollo estandarizados remotos a los alumnos de una institución educacional. Para cumplir con este propósito, he considerado que la solución debe además cumplir los siguientes requisitos para ser una opción válida.

- **Licencia de software.** La licencia del software debe ser Opensource. De esta manera se puede asegurar y revisar el código fuente utilizado, modificarlo y personalizarlo sin tener restricciones al respecto. Además, promover el uso del código libre en las instituciones educativas debería ser una prioridad para asegurar el futuro de las tecnologías abiertas.

- **Licencia de uso.** La licencia de uso debe ser gratuita o no suponer un importe mensual muy elevado ya que este coste tendrá que recaer en los alumnos o en la institución, si decide financiarlo.

- **Versatilidad de despliegue.** Debe permitir ser desplegado tanto en nube pública como en nube privada para poder adaptarse al tamaño de la institución y sus recursos actuales de infraestructura y personal de administración de esta pero poder crecer con el tiempo. Además, para ser versátil y facilitar las tares de desarrollo, extensión y mantenimiento debe ofrecer diferentes tipos de despliegue que se puedan adecuar a las necesidades de cada momento. Estos deben ser Docker, clúster de orquestación Kubernetes, servidor, contenedores linux o máquina local.

- **Facilidad de configuración.** La configuración y personalización de los entornos debe poder realizarse de una manera sencilla y concisa. Por esto, se espera que utilice una sintaxis de descripción de servicios como puede ser YAML.

- **Gestión de usuarios.** Debe permitir el registro, administración y personalización de permisos y credenciales de usuarios. De esta manera aseguramos que podamos compartimentar por ejemplo los espacios de los alumnos de los profesores, entre distintos alumnos o entre distintos cursos.

- **Buena documentación y comunidad.** Por último, se valorará que la solución este bien documenta, actualizada y que disponga de una comunidad de usuarios detrás que dan confianza a esta. Además, esto nos asegura que la resolución de problemas o las futuras actualizaciones serán mas fáciles de realizar debido al interés general.

Teniendo todo esto en cuenta y la comparativa realizada anteriormente, se ha elegido Coder como la mejor solución a implementar.

Coder es la plataforma de desarrollo en la nube que mas se adapta a los requisitos planteados ya que permite a los desarrolladores usar contenedores para escalar la infraestructura. Entre las opciones analizadas, Coder destaca por su flexibilidad en el despliegue, accesibilidad a los entornos y la capacidad de gestión administrativa eficaz de la plataforma.

La licencia de software es código abierto y de uso gratuito, aunque dispone de una versión Enterprise con soporte y alguna funcionalidad avanzada. Así pues esto concuerda con la filosofía a seguir planteada y asegura la transparencia, personalización y accesibilidad.

Ofrece despliegues en un click en nube pública así como el uso de nube privada y despliegue en contenedor, en Kuberntes, en servidor directo, en máquina virtual e incluso offline. Con esto permite adaptarse a cualquier situación presente o futura.

Permite la configuración de los entornos mediante una sintaxis sencilla y además garantiza el acceso a los entornos de desarrollo mediante web, consola y local, soportando los software de desarrollo de código offline mas utilizados como el VS Code y el Jetbrains.

El interfaz web para gestión de la plataforma, usuarios y tareas administrativas es sencilla e intuitiva.

Dispone de una amplia documentación del código, de los procesos, distintos tipos de despliegue, configuraciones de plantillas, casos de uso, etc. Además su repositorio principal de código tiene mas de sesenta mil seguidores así como cinco mil forks en GitHub.

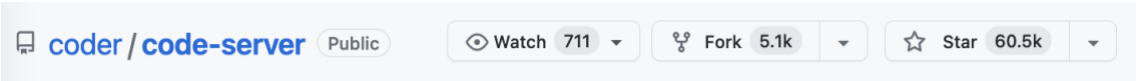


Ilustración 70. Coder - estado repositorio oficial

No obstante, si se quisiera optar por una solución con licencia comercial, la mas sencilla de implementar y de mantener de forma desatendida sería CodeAnywhere.

Por último, si se quisiera optar por una solución con licencia comercial y de despliegue en nube pública, la opción de Amazon CodeCatalyst es la que ofrece mejor rendimiento, configuraciones y precio reducido de licencias y de coste por hora de computación.

6. Conclusiones

6.1 Conclusiones del trabajo

Tras la extensa investigación sobre las soluciones actuales en el mercado para realizar entornos de desarrollo remoto estandarizados en la nube se ha podido comprobar que pese a ser una tecnología relativamente creciente dispone de un gran ecosistema de soluciones.

En el, los principales actores y competidores del cloud público ofrecen su visión sobre la tecnología de maneras distintas aunque similares. Además existen otros actores que aportan una alternativa a lo que es la tónica general en la industria; el uso de Docker y Kubernetes como orquestación.

Se ha podido observar también que hay una tendencia impulsada por algunas entidades a intentar generar un estándar respecto a los ficheros de configuración de los entornos de desarrollo así de promover su uso y integración en el resto de las tecnologías.

Se ha podido evaluar de forma práctica las distintas soluciones en su modalidad de PasS o de nube privada y se han adquiridos conocimientos auxiliares respecto al manejo de contenedores, los ficheros que los definen así como los ficheros específicos que definen en los entornos de desarrollo.

Por último, se ha podido seleccionar una tecnología que parece adecuada para ser desplegada en un entorno educacional gracias a los datos recabados y a la comparativa de soluciones realizada.

Ilustración 71. Nube de logotipos de actores en los CDEs

6.2 Objetivos del trabajo

Ilustración 72. Objetivos, explorar, analizar, comparar

Los objetivos planteados para la realización del trabajo eran:

1. Realizar una investigación exhaustiva sobre la tecnología su estado actual y los actores así como sus soluciones y características.

2. Realizar una prueba práctica con un prototipo funcional mínimo para evaluar su capacidades a nivel de usuario.

3. Realizar una comparativa general para resumir la información y tomar una decisión informada sobre que plataforma elegir en un contexto educacional.

Respecto al primer objetivo, se han investigado y evaluado 9 tecnologías distintas de distintos actores tanto gigantes de la industria del cloud y del PaaS así como pequeñas empresas que se han convertido en referentes en el sector. Esta investigación ha permitido obtener una visión general sobre el estado de la tecnología, descubrir muchas soluciones distintas, así como identificar los componentes de software o hardware que suelen construirlas. Además se ha podido recabar información actualizada sobre licencias, precios y características por lo que considero que el objetivo se ha cumplido satisfactoriamente.

Respecto al segundo objetivo, se ha podido probar de forma eficaz todas las plataformas investigadas. Se ha creado satisfactoriamente el prototipo funcional que permitía desarrollar código en Python y ejecutar el motor de bases de datos PostgreSQL en todas ellas. Además, se ha podido verificar la latencia y la respuesta así como los tiempos de ejecución y de depuración, las capacidades funcionales a nivel de interfaz de usuario y de interoperabilidad entre distintos editores de código nativo. Por todo esto, se considera que se ha alcanzado el objetivo planteado.

Respecto al tercer y último objetivo, se ha realizado una extensa comparativa y resumen en base a el tipo de infraestructura necesario, tipo de licencia de software, costes de licencias mensuales, costes de uso de computación así como de características clave de su configuración y personalización. Esto ha permitido que en base a los requisitos planteados iniciales para la toma de decisión de la tecnología a desplegar en un entorno educacional se haya podido tomar de forma satisfactoria, por lo que se considera que este punto también se ha cumplido.

6.3 Análisis de planificación y metodología seguida.

Respecto a la planificación y la metodología seguida una vez completado el trabajo y adaptado a las situaciones que se han ido viviendo, puedo asegurar que se ha seguido una metodología Waterfall con algunos tintes de Agile. Desde la planificación inicial, se separó el trabajo en etapas de forma guiada por la estructura temporal de las PECs de la asignatura. De esta manera, se ha seguido una metodología en la que al terminar una investigación tecnológica se continuaba con la siguiente investigación.

No obstante, en algunos casos, las características de configuración de alguna de las tecnologías o de los ficheros que las definen, eran desconocidos cuando se realizaba estas primeras investigaciones. Al obtener estos conocimientos en semanas posteriores, se ha procedido en algunos casos, a volver a revisar el contenido generado y adaptar, actualizar o re ejecutar ciertas pruebas de uso en base a la experiencia adquirida.

Por todo esto, se considera que la planificación ha sido correcta y la metodología seguida acertada.

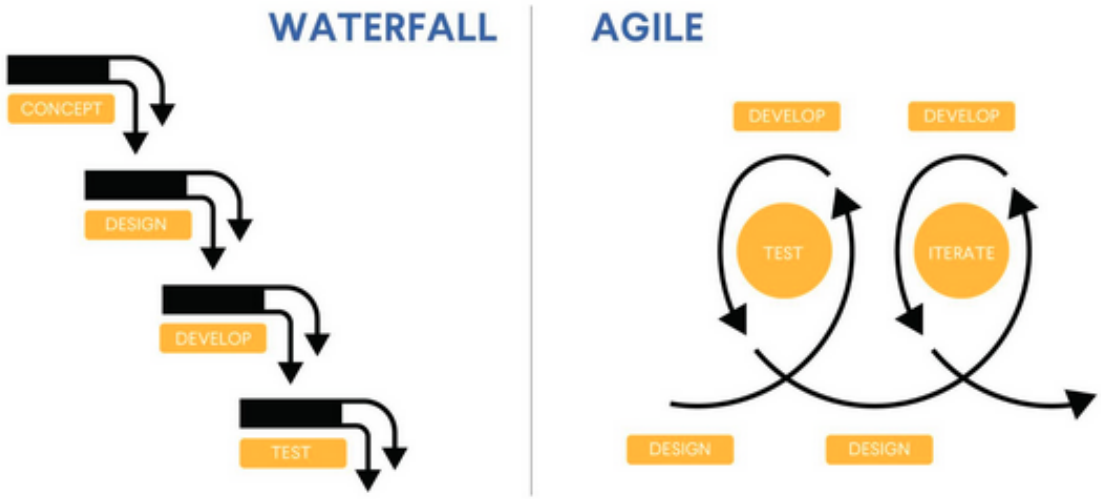


Ilustración 73. Waterfall vs Agile

6.4 Líneas de trabajo futuras

Ilustración 74. Futuro actualizarse, definir camino y planificar

Al tratarse de una tecnología en plena eclosión y con una cantidad de actores presentes enorme, es necesario realizar actualizaciones continuas sobre el estado y las capacidades de esta. Por ejemplo, durante el desarrollo de las investigaciones aparecieron soluciones como la de CodeCatalyst o Microsoft que

dejaron de ser betas cerradas y se volvieron disponibles en nuestra región, por lo que pudieron entrar en la investigación.

Por otra parte, las tecnologías de contenedores se han posicionado como el estándar actual en el despliegue y en el desarrollo de aplicaciones en la red y es un paso lógico hacer uso de ellas también para el desarrollo remoto. Tomar la decisión de implantar estas tecnologías en un entorno educacional de una manera informada y con resultados reales de pruebas da un valor añadido al estudio.

Respecto al contexto educacional, habría que aprovechar la información recabada en este trabajo para establecer una línea de investigación en la que poder identificar que asignaturas pueden beneficiarse en mayor medida de esta tecnologías y así poder crear un grupo de trabajo que trate de llevarlas a cabo.

Por último, respecto a la solución propuesta, Coder, ofrece un gran potencial de configuración y personalización pero si se quisiera optar por una opción de cloud privada porque ya se está usando y se tienen conocimientos específicos de uso y configuración de ella, también se podría beneficiar de la información aquí recogida para iniciar una investigación sobre su implantación específica.

7. Glosario

AGPLv3
GNU Affero General Public License, 14
AWS
Amazon Web Service, nube pública de Amazon, 12, 27, 40, 42, 43, 44, 45, 68
Azure
Nube pública de Microsoft, 5, 12, 27, 40, 42, 56, 57, 58, 59, 66, 68, 80, 81
baremetal
servidor al cual se accede a sus recursos sin ninguna capa de virtualización como si se tratara de un ordenador local, 4
CDEs
Cloud Development Environment, 5, 8, 43, 72
CI/CD
Continuous Integration / Continuous Deployment, 9, 43
cloud
Nube pública o privada, i, ii, 1, 3, 11, 20, 22, 34, 36, 37, 40, 42, 43, 57, 59, 67, 68, 72, 73, 75
cluster
agrefgación de servidores, iii, 20, 32, 37, 39, 40, 42, 65, 68
commit
mensaje que describe los cambios que se guardan en un repositorio de código, 29
desarrollo remoto
paradigma que permite la edición, ejecución y depuración de código en un editor que no se encuentra en el dispositivo local, 1, 2, 8, 10, 12, 15, 22, 24, 25, 30, 32, 34, 36, 38, 50, 52, 54, 61, 63, 72, 75
Devfile
formato de fichero que define una sintaxis para describir entornos de desarrollo con lenguaje YAML, 36, 37, 39, 41, 42, 45, 67, 79
devfile.yaml
fichero de configuración que usa Devfile, 4, 37, 39, 41, 43, 51
Docker
Tecnología de contenedores que permite empaquetar, editar, distribuir y ejecutar contenedores de software, i, ii, iii, 5, 11, 13, 17, 19, 20, 22, 23, 25, 27, 30, 31, 32, 35, 36, 39, 42, 45, 46, 48, 49, 51, 55, 60, 61, 62, 63, 64, 67, 69, 72, 79, 80, 81
docker-compose
tecnología que permite describir mediante un fichero la creación y ejecución de varios contenedores Docker, iii, 5, 19, 25, 30, 32, 34, 51, 54, 61, 62, 66, 67
Dockerfile
fichero de configuración de contenedor Docker, 4, 23, 24, 25, 34, 61, 67, 80
Enterprise
refrente a empresas grandes, 22, 32, 33, 40, 46, 57, 70
Git
tecnología de repositorios de código, 32, 53, 61
GPUs
Graphics Process Unit, Tarjetas de video, 9
hardware
elementos físicos que componen un ordenador, 1
IA
Inteligencia Artificial, 11
IAs
Inteligencias Artificiales, 13
IDE
Integrated Development Environment, iii, iv, 5, 25, 29, 30, 31, 32, 34, 35, 36, 37, 39, 41, 42, 45, 51, 52, 53, 55, 59, 63, 66, 79
IDEs
Integrated Development Environments, 37, 46
Json
Vease JSON, 46
JSON
JavaScript Object Notation, fichero de intercambio de datos, 27, 28
Kubernetes
Tecnología de orquestación de contenedores, iii, 13, 20, 22, 25, 31, 32, 36, 37, 39, 40, 43, 45, 63, 64, 68, 69, 72
Linux
sistema operativo de código abierto y uso de kernel, 16, 37, 40, 52, 54, 61
linux container
Contenedores linux, lxc o lvm. tecnología de contenedores con kernel compartido, 4
máquinas virtuales
Tecnología que permite empaquetar un sistema operativo completo y distribuirlo para su uso en cualquier requisito de Hardware, 1
Tecnología que permite empaquetar un sistema operativo completo y distribuirlo para su uso en culaquier requisito de Hardware, i, 8
Marketplace

8. Bibliografía

SALAS, R.P. y HO, J., 2021. A Remote/Virtual Robotics Lab. 2021 IEEE Frontiers in Education Conference (FIE). S.l.: IEEE, pp. 1-4. ISBN 9781665438513. DOI 10.1109/FIE49875.2021.9637340. Disponible en: https://ieeexplore.ieee.org/document/9637340

David J, Malan, 2022. *Standardizing Students' Programming Environments with Docker Containers.* Hardvare University, July 2022, DOI:10.1145/3502717.3532164 Conference: ITiCSE 2022: Innovation and Technology in Computer Science Education. Disponible en: https://cs.harvard.edu/malan/publications/iticse22.pdf

A Complete Guide to Agile Product Management. (s. f.). Recuperado 17 de junio de 2023, de https://ryanseamons.com/agile-product-management/

About billing for GitHub Codespaces—GitHub Docs. (s. f.). Recuperado 17 de junio de 2023, de https://docs.github.com/en/billing/managing-billing-for-github-codespaces/about-billing-for-github-codespaces

About—Coder v2 Docs. (s. f.). Recuperado 17 de junio de 2023, de https://coder.com/docs/v2/latest

Amazon CodeCatalyst. (s. f.). Recuperado 17 de junio de 2023, de https://codecatalyst.aws/explore/dev-environments

Architecture—Coder v2 Docs. (s. f.). Recuperado 17 de junio de 2023, de https://coder.com/docs/v2/latest/about/architecture

Awesome-compose/postgresql-pgadmin at master · docker/awesome-compose · GitHub. (s. f.). Recuperado 17 de junio de 2023, de https://github.com/docker/awesome-compose/tree/master/postgresql-pgadmin

Cloud IDE · Online Code Editor · Codeanywhere. (s. f.). Recuperado 17 de junio de 2023, de https://codeanywhere.com/

Cloud Native Computing Foundation. (s. f.). Recuperado 17 de junio de 2023, de https://www.cncf.io/

Codeanywhere. (2023). En *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Codeanywhere&oldid=1131302368

Coder—Your Self-Hosted Remote Development Platform. (s. f.). Recuperado 17 de junio de 2023, de https://coder.com/

Colecciones—SkylarDunn-0040 | Microsoft Learn. (s. f.). Recuperado 17 de junio de 2023, de https://learn.microsoft.com/es-es/users/skylardunn-0040/collections/q87ku5ox21ggg4?wt.mc_id=build23_breakout_collection_azuremkt_BRK251H

Dev Container metadata reference. (s. f.). Recuperado 17 de junio de 2023, de https://containers.dev/implementors/json_reference/

Devfile | JetBrains Space Documentation. (s. f.). Recuperado 17 de junio de 2023, de https://www.jetbrains.com/help/space/devfile.html#auto-create-devfile

Devfile schema—Docs. (s. f.). Recuperado 17 de junio de 2023, de https://devfile.io/docs/2.0.0/devfile-schema

Docker Compose overview | Docker Documentation. (s. f.). Recuperado 17 de junio de 2023, de https://docs.docker.com/compose/
Docker/awesome-compose at e6b1d2755f2f72a363fc346e52dce10cace846c8. (s. f.). Recuperado 17 de junio de 2023, de https://github.com/docker/awesome-compose/tree/e6b1d2755f2f72a363fc346e52dce10cace846c8
Dockerfile reference | Docker Documentation. (s. f.). Recuperado 17 de junio de 2023, de https://docs.docker.com/engine/reference/builder/
Eclipse Che Incubator. (s. f.). Recuperado 17 de junio de 2023, de https://github.com/che-incubator
GitHub Codespaces. (s. f.). Recuperado 17 de junio de 2023, de https://github.com/features/codespaces
GitHub Codespaces overview—GitHub Docs. (s. f.). Recuperado 17 de junio de 2023, de https://docs.github.com/en/codespaces/overview
Gitpod: Always ready-to-code. (s. f.). Recuperado 17 de junio de 2023, de https://www.gitpod.io/
Gitpod for Education. (s. f.). Recuperado 17 de junio de 2023, de https://www.gitpod.io/for/education
Gitpodifying—The Ultimate Guide. (s. f.). Recuperado 17 de junio de 2023, de https://www.gitpod.io/guides/gitpodify
Google launches a GitHub Copilot competitor | TechCrunch. (s. f.). Recuperado 17 de junio de 2023, de https://techcrunch.com/2023/05/10/google-launches-a-github-copilot-competitor/
Home | Eclipse Che. (s. f.). Recuperado 17 de junio de 2023, de https://www.eclipse.org/che/
How To Get Started With Space Cloud Dev Environments | The Space Blog. (s. f.). Recuperado 17 de junio de 2023, de https://blog.jetbrains.com/space/2022/10/26/get-started-with-space-dev-environments/
Inicio rápido: Configuración de Microsoft Dev Box—Microsoft Dev Box | Microsoft Learn. (s. f.). Recuperado 17 de junio de 2023, de https://learn.microsoft.com/es-es/azure/dev-box/quickstart-configure-dev-box-service?tabs=AzureADJoin
Install Docker Engine on Debian | Docker Documentation. (s. f.). Recuperado 17 de junio de 2023, de https://docs.docker.com/engine/install/debian/#install-using-the-repository
Installing Che: Eclipse Che Documentation. (s. f.). Recuperado 17 de junio de 2023, de https://www.eclipse.org/che/docs/stable/administration-guide/installing-che/
JetBrains Space: A Complete Software Development Platform. (s. f.). Recuperado 17 de junio de 2023, de https://www.jetbrains.com/space/
Microsoft Dev Box – Dev Workstation in the Cloud | Microsoft Azure. (s. f.). Recuperado 17 de junio de 2023, de https://azure.microsoft.com/en-us/products/dev-box/
Openshift Dev Spaces Overview | Red Hat Developer. (s. f.). Recuperado 17 de junio de 2023, de https://developers.redhat.com/products/openshift-dev-spaces/overview
Overview | Docker Documentation. (s. f.-a). Recuperado 17 de junio de 2023, de https://docs.docker.com/desktop/dev-environments/

Overview | Docker Documentation. (s. f.-b). Recuperado 17 de junio de 2023, de https://docs.docker.com/compose/compose-file/

Python Program to Check Prime Number. (s. f.). Recuperado 17 de junio de 2023, de https://www.programiz.com/python-programming/examples/prime-number

Qué es la metodología waterfall y cuándo utilizarla [2022] • Asana. (s. f.). Recuperado 17 de junio de 2023, de https://asana.com/es/resources/waterfall-project-management-methodology

Reimagining Dev Workstations with Microsoft Dev Box. (s. f.). Recuperado 17 de junio de 2023, de https://techcommunity.microsoft.com/t5/azure-developer-community-blog/reimagining-developer-workstations-with-microsoft-dev-box/ba-p/3825055

RoseHJM. (2023, mayo 10). *Create and configure a dev center—Azure Deployment Environments*. https://learn.microsoft.com/en-us/azure/deployment-environments/quickstart-create-and-configure-devcenter

Self-Hosted Remote Development Environments. (2023). [Go]. Coder. https://github.com/coder/coder (Obra original publicada en 2021)

Set up a dev environment | Docker Documentation. (s. f.). Recuperado 17 de junio de 2023, de https://docs.docker.com/desktop/dev-environments/set-up/

Terraform by HashiCorp. (s. f.). Recuperado 18 de junio de 2023, de https://terraform.io

Top 9 GitHub Copilot alternatives to try in 2023 (free and paid). (s. f.). Recuperado 17 de junio de 2023, de https://www.tabnine.com/blog/github-copilot-alternatives/

Try Docker Compose | Docker Documentation. (s. f.). Recuperado 17 de junio de 2023, de https://docs.docker.com/compose/gettingstarted/

Welcome! | minikube. (s. f.). Recuperado 17 de junio de 2023, de https://minikube.sigs.k8s.io/docs/

Workspace Image—Gitpod Docs. (s. f.). Recuperado 17 de junio de 2023, de https://www.gitpod.io/docs/configure/workspaces/workspace-image