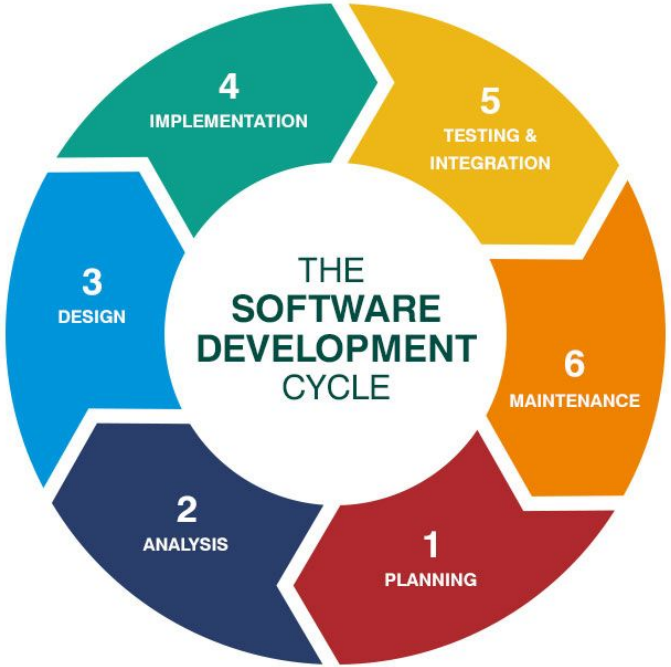
DevOps II - Aplicación en desarrollo de software

ADS

DevOps en Desarrollo de software



Con respecto al desarrollo de software, también se aplican muchas herramientas DevOps. Vamos a ir separando los distintos pasos de acuerdo al círculo que está en esta diapositiva:

- **Planeamiento general del proyecto**
- **Análisis de requisitos**
- **Diseño**
- **Implementación**
- **Testeo e integración**
- **Mantenimiento**

Obviamente, se van a repetir muchas herramientas que ya vimos con respecto a las prácticas en tecnología de la información.

Herramientas para planeamiento general del proyecto

Acá entran en juego todas las metodologías de desarrollo; SCRUM, XP, modelo evolutivo, por prototipos, etc..

El planeamiento desde la génesis del proyecto, donde se detallan los pasos a seguir para intentar conseguir que el mismo supla a la empresa con un sistema de calidad, es el que contiene la metodología de desarrollo subyacente.

Más allá de la metodología de desarrollo, hay herramientas para **organización** y **automatización del seguimiento de tareas** que entran en esta categoría de planeamiento general del proyecto.

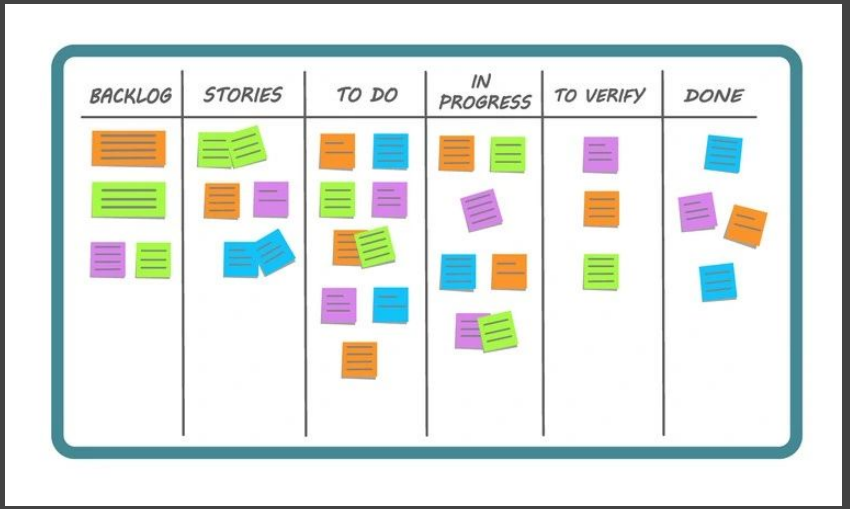
Herramientas para planeamiento general del proyecto

Kanban: si bien Kanban se puede entender como metodología ágil en general, en realidad vamos a ver que en el contexto de DevOps, junto con cualquier otra metodología de desarrollo ágil (como vimos con SCRUM), permite **visualizar** y **gestionar el trabajo** para lograr lo que todas las metodologías ágiles pretenden: mejorar la eficiencia, reducir el tiempo de entrega y asegurar un flujo continuo de trabajo.

Kanban – Introducción

El principio básico de Kanban (看板) es **visualizar el trabajo** dividido en columnas que representan diferentes etapas o estados del flujo de trabajo. Siempre están estas tres mínimo:

- **To do**: a hacer, pero todavía no comenzamos.
- **In Progress**: en progreso, trabajando actualmente.
- **Done**: tareas completadas.



Kanban - Tarjetas

Cada tarea o elemento de trabajo se representa con una **tarjeta** en el tablero. Las tarjetas pueden tener ciertos detalles:

- Un título
- Una descripción
- Asignaciones
- Fechas límite
- Prioridad

Kanban - WIP Limits

Una característica de Kanban es la de limitar la cantidad de trabajo que hay en la columna de trabajo en progreso, a esto se lo conoce como **WIP Limits**.

La idea es que el equipo no se sobrecargue y termine las tareas ya asignadas antes de comenzar nuevas.

Durante el desarrollo, es fácil decir “Bueno, voy a parar de hacer esto un segundo y arranco con otra cosa”, sin embargo esto es muy ineficiente; toma tiempo y quita concentración de cada tarea, dividiéndola en dos.

*Es así que si el **WIP Limit** para ‘In Progress’ es 3, entonces solo podemos tener 3 tareas en esa columna, y si todas las tareas en esa columna están ocupadas, el equipo no puede comenzar una nueva hasta que terminen una de las que ya está en progreso.*

Kanban - Mejora Continua

Obviamente, al ser una metodología ágil, y aplicar leyes de Lehman con relación a la autorregulación, Kanban promueve la **mejora continua** del flujo de trabajo.

Por lo general, se recolectan métricas como **lead time** (tiempo que tarda una tarea en moverse de una columna “To Do” a “Done”), y ajustar prácticas para mejorar la eficiencia.

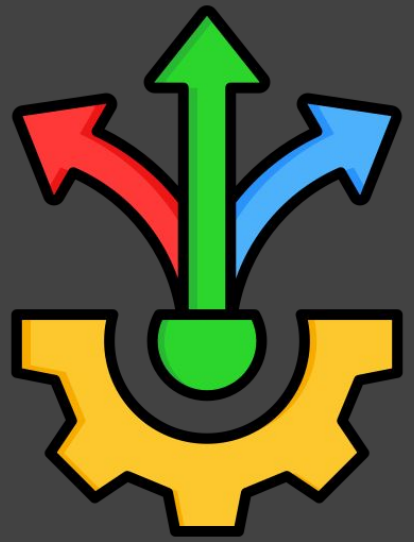
Por ejemplo, si se están acumulando muchas tareas en “In Progress”, hay que modificar ya el WIP Limit porque estamos siendo ineficientes.

Kanban - Prioridades flexibles

Kanban permite **priorización dinámica**; las prioridades pueden cambiar a medida que surgen nuevas necesidades al principio de cada sprint (o incluso durante los sprints con los ***Kanban de emergencia***).

Los Kanban de Emergencia se emiten de modo temporal cuando se requiera hacer frente a partes defectuosas del código, trabajos extraordinarios o esfuerzos especiales, si la situación lo amerita.

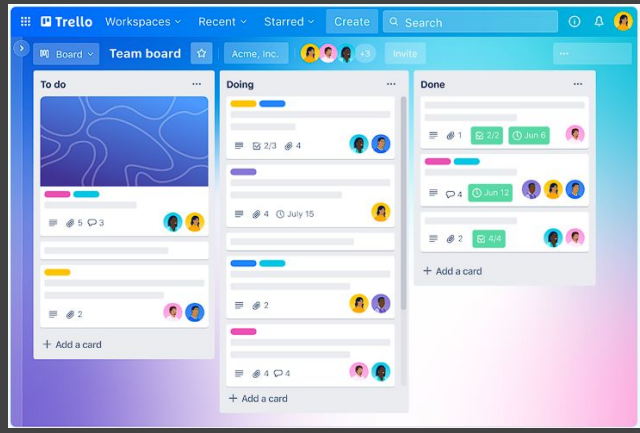
Por ejemplo, si aparece una tarea urgente, se agrega al tablero en la columna “To Do” y se trabaja en eso sin esperar a la finalización de un ciclo completo.

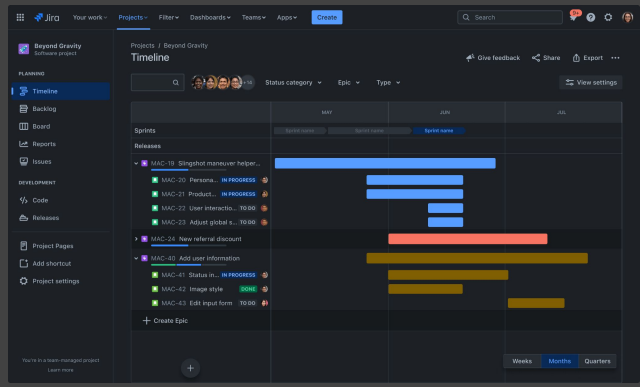


Herramientas para aplicar Kanban

Las herramientas más utilizadas para aplicar Kanban son:

- **Trello**, que dispone de un tablero visual para pequeñas y medianas empresas.
- **Jira**, una herramienta de gestión usada en empresas mucho más grandes.
- **Gitlab** y **Azure** también tienen sus propios tableros Kanban integrados para poder manejar el flujo de trabajo.





Herramientas para el análisis de requisitos

Lo más relevante en las metodologías de desarrollo de hoy en día para realizar un análisis de requisitos satisfactorio es dividir los mismos en historias manejables para facilitar la entrega iterativa y la priorización en ciclos de desarrollo: **historias de usuario**.

Más allá de las historias de usuario y las metodologías de desarrollo que las promueven, existen algunas herramientas que permiten la realización de un análisis de requisitos de manera más cómoda...

Herramientas para el análisis de requisitos

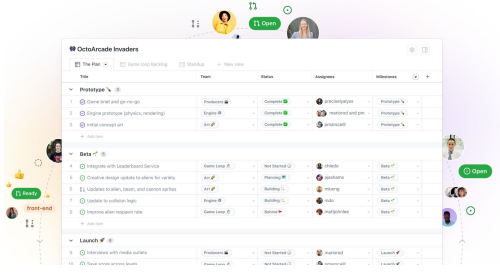
Confluence: permite crear, organizar y compartir documentos dentro de un equipo o empresa; funciona como una *wiki* empresarial para almacenar y gestionar información relevante.

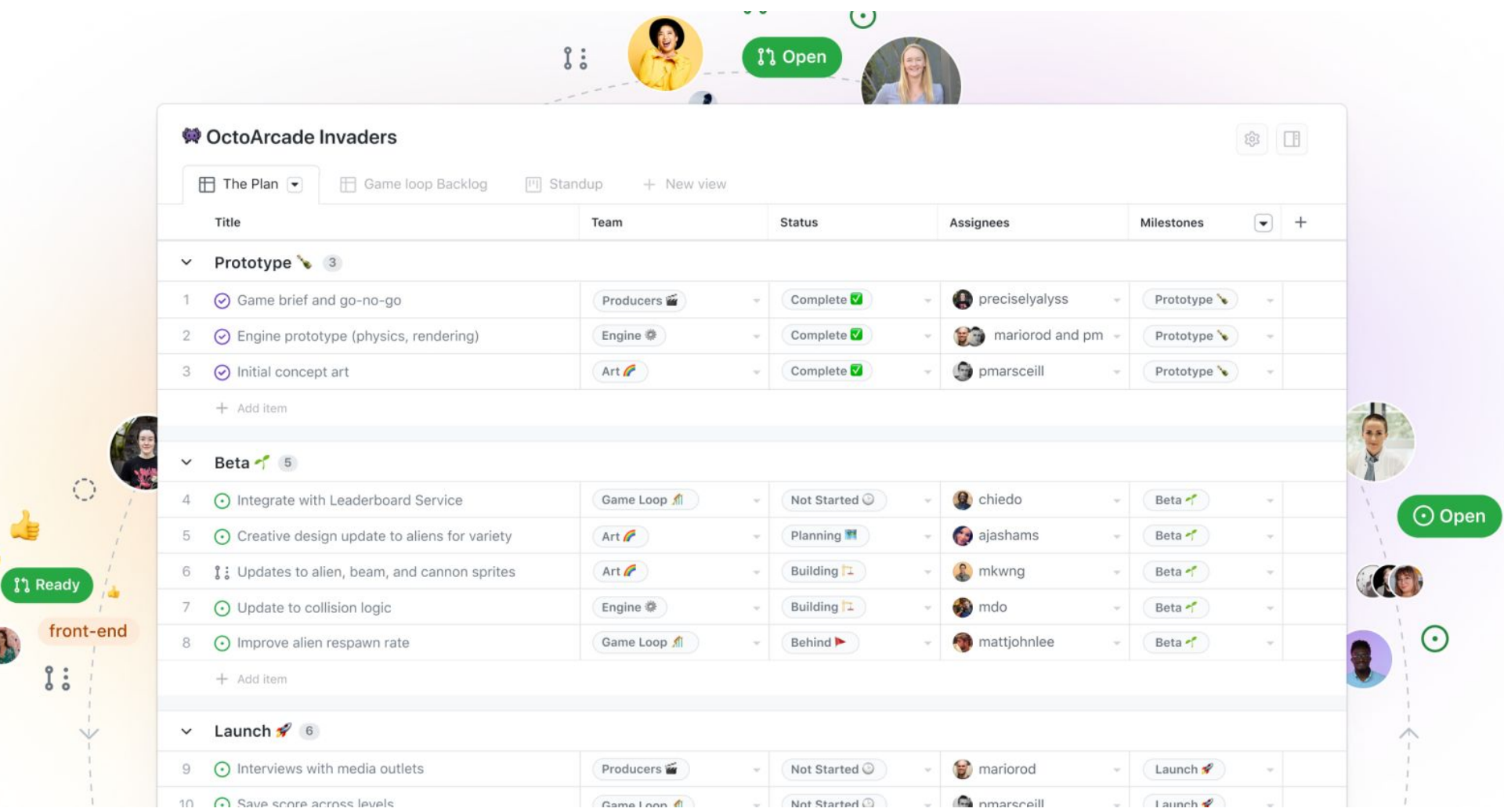
- **Documentación de requisitos**: Creamos páginas para documentar los requisitos del proyecto de manera detallada, ya sean **funcionales** como **no funcionales**.
- **Bien organizado**: Los requisitos se organizan en una jerarquía clara, utilizando títulos, subtítulos y links internos (como Wikipedia!!).
- **Historial de versiones**: Confluence guarda versiones anteriores de los documentos, permitiendo ver el historial de cambios; útil para mantener un registro de cómo fueron evolucionando los requisitos.

Herramientas para el análisis de requisitos

GitHub Issues: Es mucho más simple que Confluence; permite gestionar principalmente bugs y mejoras. Muy útil para proyectos que se manejan en GitHub.

- **Issues**: ‘incidentes’. Éstos pueden crearse para cada requisito, conteniendo una descripción detallada del mismo, links, discusiones, entre otros.
- **Asignación**: Los issues pueden ser asignados a desarrolladores o equipos para hacerse responsables del análisis o implementación del mismo, permitiendo una buena organización.
- **Commits y pull requests**: Al estar integrado directamente con el repositorio de código en GitHub, podemos referenciar commits o pull requests directamente desde los issues.




Open
OctoArcade Invaders
The Plan
Game loop Backlog
Standup
New view
Title
Team
Status
Assignees
Milestones
Prototype 3
1 Game brief and go-no-go
Producers
Complete
preciselyalyss
Prototype
2 Engine prototype (physics, rendering)
Engine
Complete
mariorod and pm
Prototype
3 Initial concept art
Art
Complete
pmarsceill
Prototype
Add item
Beta 5
4 Integrate with Leaderboard Service
Game Loop
Not Started
chiedo
Beta
5 Creative design update to aliens for variety
Art
Planning
ajashams
Beta
6 Updates to alien, beam, and cannon sprites
Art
Building
mkwng
Beta
7 Update to collision logic
Engine
Building
mdo
Beta
8 Improve alien respawn rate
Game Loop
Behind
mattjohnlee
Beta
Add item
Launch 6
9 Interviews with media outlets
Producers
Not Started
mariorod
Launch
Ready
front-end
Open

Herramientas para el diseño

Una herramienta que ayuda mucho al diseño, y vimos la clase pasada es **Terraform**, ya que permite estructurar el proyecto de manera automática.

Otro puede ser **AWS CloudFormation**, que permite modelar, aprovisionar y gestionar recursos en la nube de AWS usando JSON o YAML.

CloudFormation usa un archivo de **plantilla** que describe los distintos **recursos** y su **configuración**. Cuando se manda el archivo a la nube, CloudFormation crea y gestiona los recursos en el orden adecuado, garantizando que se manejen las dependencias entre ellos.

Al conjunto completo de recursos que definió la plantilla, lo llamamos **pila** o **stack**.

Herramientas para el diseño

AWS CloudFormation contiene:

- **IaC (Infraestructura como Código)**: Permite que toda la infraestructura de un proyecto se defina como código en un archivo de plantilla, convirtiéndo a la infraestructura y al código fuente en repetible y visionable.
- **Despliegue de recursos**: Los recursos definidos se van a crear y configurar de manera consistente, ya que la plantilla está automatizada, minimizando los riesgos y errores.
- **Gestión de dependencias**: Se hace automáticamente; si hay dependencias en el proyecto entre sí, las mismas van a gestionarse de manera automática para que los recursos se creen en el orden correcto.

Herramientas para la implementación

Se puede hacer una relación entre seguridad (TI) e implementación; aplicaciones como **SonarQube** o **Snyk**, que realizan revisiones automáticas de calidad y seguridad del código aplican tanto para tecnología de la información como para desarrollo de software.

Git, por otra parte, al tener un funcionamiento muy amplio como software de control de versiones, también puede aplicar para control de implementación, permitiendo el acceso al código a cualquier desarrollador del proyecto.

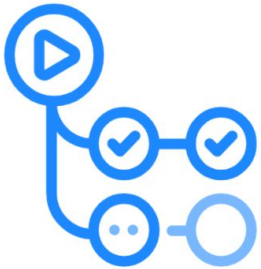
Con relación a integración continua y despliegue continuo (CI/CD), también nos vamos a referir a la implementación del código: herramientas que vimos la clase pasada como **Jenkins** pueden ser de gran ayuda para desarrollo de software también.

Herramientas para la implementación

GitHub Actions es otra herramienta que puede ser esencial para la implementación si estamos trabajando con repositorios en GitHub. Este nos permite automatizar, customizar y ejecutar los flujos de trabajo del desarrollo de software en nuestro repositorio.

La página web de GitHub actions lo explica perfectamente:

https://docs.github.com/es/actions/about-github-actions/understanding-github-actions



GitHub Actions

Herramientas para testeo e integración

Hay tres vertientes por las cuales se puede analizar el testeo e integración de una aplicación:

- A través de **pruebas automatizadas**, creando así pruebas unitarias, de integración, de regresión y funcionales para validar automáticamente el comportamiento del código en cada etapa.
- A través de **pruebas en entornos de staging**, permitiendo configurar pruebas en entornos que simulan producción (staging).
- A través de **monitorización de logs**, capturando y analizando logs en tiempo real durante el testing y después de la integración.

Herramientas para pruebas

JUnit es un framework de pruebas unitarias para Java. Permite validar que cada unidad de código (métodos, clases, funciones) funcionen de manera correcta en aislamiento.

JUnit tiene la capacidad de realizar **pruebas unitarias** y **pruebas de regresión**; esta última nos permite detectar si los cambios en el código realizados para arreglar pruebas unitarias causan errores en otras partes que anteriormente funcionaban.

JUnit dispone de:

- **Anotaciones**: definen el comportamiento de los métodos de prueba. Algunas anotaciones pueden ser:
  - @Test: marca un método como una prueba.
  - @Before y @After: definen acciones antes y después de cada prueba.
  - @BeforeClass y @AfterClass: ejecutan configuraciones y limpiezas a nivel de clase.

JUnit Herramientas para pruebas

JUnit también dispone de **Assertions**. Estas permiten verificar condiciones esperadas. Si una condición falla, la prueba se marca entonces como fallida:

- assertEquals()
- assertTrue()
- assertNotNull()

Ejemplo:

```
public class Calculadora {

     public int Suma(int a, int b) {

          return a + b;

     }

}

//en base a este método, hago un test…
```

```
public class TestCalculadora {

     @Test

     public void testSuma() {

          Calculadora c = new Calculadora();

          int resultado = c.Suma(2, 3);

          assertEquals(5, resultado, “La
suma da 5”);

     }

}


//acá creamos un test que utiliza el método
Suma de la clase Calculadora, y verifica si
el resultado es correcto.
```

Herramientas para pruebas

Selenium se utiliza para pruebas de interfaz de usuario (UI) en el front-end de páginas de internet. Permite automatizar la interacción del usuario con el browser.

Selenium simula el comportamiento del usuario en aplicaciones web, permitiendo ser ejecutado en **diferentes navegadores** (google chrome, mozilla firefox, microsoft edge, etc.), y se integra con otros frameworks de pruebas como JUnit, TestNG o cualquiera que usemos.

Esta herramienta usa una API llamada **WebDriver** para poder interactuar con el browser. La misma nos permite localizar elementos de la página mediante selectores (*id*, *name*, *class*, *CSS selector*, *etc.*), para simular un clic, envío de texto, o verificación de estado.

Herramientas para pruebas

Ejemplo:

```
public class LoginTest {

    @Test

    public void testLogin() {

        WebDriver driver = new
ChromeDriver();

        driver.get(“http://app.com/login”);

            WebElement usuario =
driver.findElement(By.id("usuario"));

        WebElement contrasenia =
driver.findElement(By.id("contrasenia"));

        WebElement botonLogin =
driver.findElement(By.id("botonLogin"));

        usuario.sendKeys(“admin”);

        contrasenia.sendKeys(“111111”);

        botonLogin.click();

        assertTrue(driver.getCurrentUrl().contai
ns(“/paginaPrincipal”));

        driver.quit();

        }

}
```

//Acá lo que hicimos fue simular el ingreso de un usuario llamado “admin” con contraseña “111111”. En el caso de que el login haya funcionado correctamente, debería habernos redirigido a la paginaPrincipal, entonces utilizamos la ruta del mismo como assertTrue para verificar.

Herramientas para pruebas

Postman sirve para hacer pruebas de **APIs**. Nos deja enviar solicitudes HTTP y verificar las respuestas sin necesidad de programar.

Las respuestas son verificadas de dos maneras:

- Podemos ver si contienen los datos esperados
- Verificamos el estado HTTP.
- Podemos crear colecciones de pruebas y ejecutarlas automáticamente.

Supongamos que queremos realizar una petición GET con una API de Pokemon o con la de países y banderas...

POSTMAN

Herramientas para pruebas

Vamos a usar este URL: https://pokeapi.co/api/v2/pokemon/pikachu

... o este para el de los países: https://restcountries.com/v3.1/name/argentina

Vamos a utilizar **GET**.

Podemos usar la versión online de Postman también!!

Después esto se testea de manera automática:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
pm.test("Name is Pikachu", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.name).to.eql("pikachu");
});
```

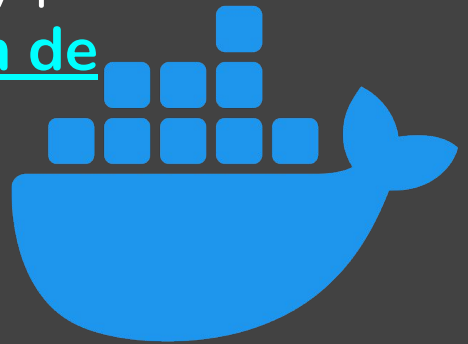
```
pm.test("Type is Electric", function () {
    var jsonData = pm.response.json();
    var type = jsonData.types[0].type.name;
    pm.expect(type).to.eql("electric");
});
```

```
//verificamos estado HTTP, nombre del pokemon y qué tipo es
```

Docker - Introducción

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de **contenedores de software**, en lugar de virtualizar hardware (como lo hacen las máquinas virtuales).

En el contexto de DevOps y desarrollo de software, Docker se convierte en una herramienta importantísima para simplificar la integración y entrega continua y para estar seguros de que **las aplicaciones se ejecuten de manera consistente**.



Docker - Origen

En la era moderna del desarrollo de software, los entornos de trabajo estandarizados y la capacidad de empaquetar aplicaciones junto con todas sus dependencias son requisitos fundamentales. A medida que los proyectos de software crecen y se vuelven más complejos, la necesidad de un **entorno uniforme** se hace cada vez más evidente, y Docker es una tecnología que responde a esta necesidad.

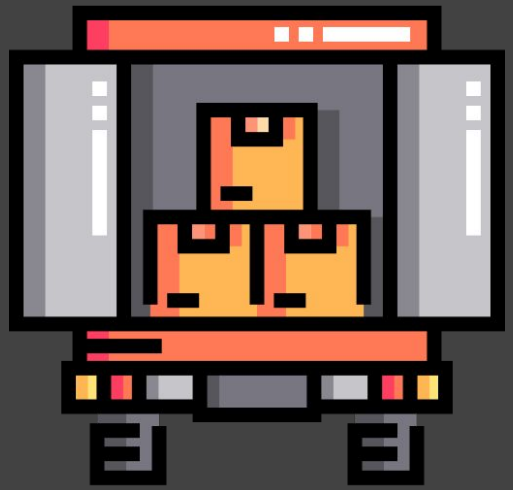
Imaginemos la situación común en la que una aplicación funciona perfectamente en la computadora de un desarrollador, pero al desplegarse en otro entorno (como en el servidor de producción o en la máquina de otro desarrollador), aparecen errores o se empieza a comportar de manera inesperada.

Ahí es donde tenemos que intentar manejar entornos complejos y dependencias entre sistemas; Docker nos provee un entorno para que repliquemos el sistema en cualquier lugar sin importar la arquitectura, utilizando algo llamado **contenedores**.

Docker - Contenedores vs. Máquinas Virtuales

Para entender Docker, primero hay que entender en qué se diferencian los contenedores de las máquinas virtuales:

Recordemos que una **máquina virtual** emula todo un sistema operativo completamente de arriba a abajo con su propio kernel y recursos (CPU, RAM, espacio en disco), lo que las hace muy pesadas en términos de recursos porque requieren su propio kernel y todos los drivers del sistema operativo para funcionar.

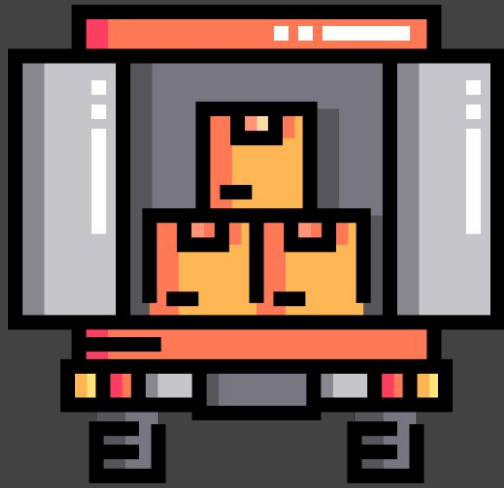


Por otra parte, un **contenedor** comparte el kernel del sistema operativo anfitrión, pero crea un entorno aislado que contiene solo la aplicación y sus dependencias. Al no emular el sistema operativo, los contenedores son **significativamente más ligeros** y **pueden ejecutarse a gran velocidad** con menos consumo de recursos.

Docker - Qué ventajas nos dan los contenedores

Gracias a los contenedores, podemos trabajar en un entorno controlado, predecible y replicable. Sus características ventajosas incluyen:

- **Consistencia entre entornos**: podemos asegurar que la aplicación funcione de la misma manera en cualquier entorno.
- **Escalabilidad**: las aplicaciones escalan en función de la demanda, ya que cada componente es muy fácil de desplegar y replicar, lo que lo convierte en crucial en aplicaciones de alta disponibilidad que necesiten adaptarse a fluctuaciones de tráfico (como vimos en DevOps I - herramientas para escalabilidad).
- **Automatización y CI/CD**: se integran los contenedores en pipelines de integración y entrega continua. Esto permite desplegar nuevas versiones rápidamente y con menos riesgo de errores.
- **Aislamiento de aplicaciones:** es posible ejecutar múltiples instancias de aplicaciones en el mismo sistema sin preocuparse de conflictos de dependencias ya que cada contenedor funciona de manera aislada.



Docker - Funcionamiento interno

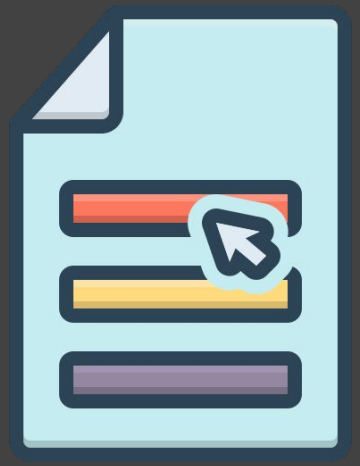
Docker combina varias tecnologías:

- **Namespaces**: aíslan los recursos del sistema para que los contenedores parezcan entornos independientes.
- **Control Groups**: Limitan y distribuyen los recursos del sistema (CPU, RAM, etc.), a cada contenedor.
- **Union File Systems**: Nos proveen con un sistema de archivos en capas que hace que las imágenes de Docker sean eficientes en almacenamiento.
- **Networking**: Configura redes virtuales para que los contenedores puedan comunicarse entre sí y con el exterior sin interferir con el sistema anfitrión.

Kernel de Linux - Namespaces

Los namespaces son una funcionalidad del kernel de Linux que Docker usa para aislar diferentes aspectos de los contenedores. En Docker, los namespaces permiten que cada contenedor se ejecute en su propio “universo” sin interferir con otros contenedores o con el sistema anfitrión. Los namespaces más usados son:

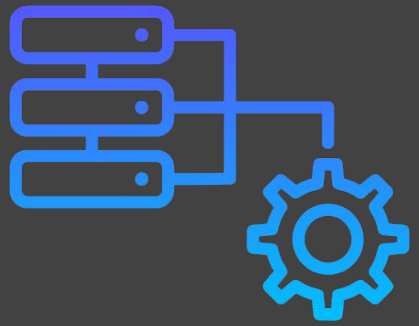
- **PID (Process ID):** Aísla el espacio de los procesos para que cada contenedor tenga su propia numeración de PID. Esto significa que un proceso dentro de un contenedor no puede ver ni interferir con los procesos de otros contenedores ni del anfitrión.
- **Mount**: Controla el sistema de archivos que cada contenedor puede ver. Esto permite que un contenedor vea únicamente su sistema de archivos en lugar de todo el sistema de archivos del anfitrión.
- **Network**: Aísla las interfaces de red. Cada contenedor puede tener su propio conjunto de interfaces de red y su propio espacio de red, incluyendo puertos y direcciones IP.
- **IPC**: Aísla la comunicación entre procesos, de modo que los procesos dentro de un contenedor solo puedan comunicarse con otros procesos dentro del mismo contenedor.
- **User**: Permite que los contenedores tengan su propio espacio de usuarios, creando un entorno seguro en el que los usuarios dentro del contenedor pueden no tener privilegios en el sistema anfitrión, independientemente de su privilegio dentro del contenedor.



Kernel de Linux - CGroups

Los cgroups o “grupos de control” son otra funcionalidad del kernel de Linux que Docker utiliza para asignar y limitar los recursos que cada contenedor puede utilizar. Mediante los cgroups, Docker puede imponer límites estrictos en la cantidad de CPU, memoria RAM, disco y otras métricas de recursos que cada contenedor puede consumir.

Por ejemplo, si un contenedor ejecuta un proceso que consume mucha CPU, los cgroups pueden limitar el uso de CPU de ese contenedor, asegurando que otros contenedores o el sistema anfitrión no se vean afectados por este consumo excesivo.



Docker - UnionFS

Docker implementa UnionFS, que es un sistema que permite que cada imagen de Docker se construya sobre capas que representan cambios individuales en el sistema de archivos, en lugar de copiar archivos completos. Cada capa es inmutable, lo cual significa que no se modifica una vez creada. En lugar de eso, cada cambio crea una nueva capa.

Cuando Docker crea un contenedor a partir de una imagen, apila las capas y crea una capa adicional, llamada la capa de contenedor o capa de escritura. Solo esta última capa es modificable y contiene los cambios específicos del contenedor.

Docker - Redes

Docker tiene una infraestructura de red propia que es muy flexible. Los modos de red principales son:

- **Bridge**: Crea una red virtual privada donde los contenedores pueden comunicarse entre sí.
- **Host**: El contenedor comparte la red del anfitrión en lugar de tener su propia red aislada, sin ofrecer aislamiento de red (al contrario de **bridge**).
- **Overlay**: Permite crear redes que abarcan múltiples hosts, ideal para clústeres Docker en producción.
- **None**: Desactiva la red del contenedor.

Obviamente, Docker no prescinde de funcionalidades de Linux en términos de las redes, utilizando de hecho iptables (reglas de Firewall de Linux).

Docker - ¿Cómo lo hacemos andar?

Dockerfile: Primero, creamos un archivo llamado Dockerfile que construye la imagen del contenedor, definiendo el sistema operativo base, las dependencias de la aplicación, configuraciones específicas, variables de entorno y comandos a ejecutar.

Build: Ejecutamos ‘docker build’, procesando el Dockerfile y construyendo la imagen a partir del mismo. Cada instrucción en el Dockerfile crea una nueva capa en la imagen, usando UnionFS para apilar las capas de manera eficiente.

Run: Ejecutamos ‘docker run’, lanzando un contenedor a partir de una imagen tomada. Docker crea una capa de escritura temporal para almacenar los cambios realizados durante la ejecución y usa namespaces y cgroups para aislar el contenedor del sistema anfitrión.

Docker - Cliente

Docker usa arquitectura cliente-servidor. El **Docker Daemon** (*dockerd*) es un proceso en segundo plano que se ejecuta en el sistema anfitrión y maneja todas las operaciones de construcción, ejecución y supervisión de contenedores. El cliente Docker (*docker*) es una interfaz de línea de comandos que los usuarios utilizan para interactuar con el Daemon.

Cuando se ejecuta un comando como *docker run*, el cliente Docker se comunica con el Docker Daemon mediante una API para ejecutar la tarea solicitada.

Docker

Concluímos en que usar Docker para crear y gestionar contenedores puede simplificar la creación de sistemas altamente distribuidos, permitiendo que múltiples aplicaciones, las tareas de los trabajadores y otros procesos funcionen de forma autónoma en una única máquina física o en varias máquinas virtuales.

Docker permite una plataforma como servicio o *PaaS* de estilo de despliegue, además que simplifica la creación y el funcionamiento de las tareas de carga de trabajo.

El enfoque de Docker basado en contenedores impulsa la adopción general de filosofía DevOps, facilitando la colaboración entre desarrollo y operaciones y agilizando los ciclos de vida de CI/CD.

Herramientas para entornos de producción o staging

Kubernetes es una herramienta muy completa que se utiliza en muchas etapas de DevOps. Sin embargo, ayuda principalmente a administrar aplicaciones en contenedores en entornos de **producción** o **staging**.

- **Orquestación**: Kubernetes administra el ciclo de vida de los contenedores (inicio, paro, reinicio) en un clúster. Coordina el despliegue de las aplicaciones y asegura que la cantidad deseada de réplicas de los contenedores esté siempre en ejecución.
- **Rolling deploy**: Podemos realizar una actualización de una aplicación sin tiempo de inactividad mediante despliegues *rolling*, donde los contenedores se actualizan gradualmente.

Herramientas para entornos de producción o staging

Supongamos que tenemos una aplicación web distribuida y nos falta desplegarla en staging para pruebas y después en producción:

Primero **configuramos** un *cluster* en un proveedor de nube y lo organizamos en namespaces: *production* y *staging*.

Hacemos un **pipeline** para desplegar los cambios automáticamente en el namespace que hicimos antes, *staging*, cada vez que haya un merge a la rama de desarrollo. Después de eso, lo desplegamos en *production*.

En este momento, podemos ir probando nuevas versiones en el namespace *staging* y después mandamos un ***Canary release*** (para una cantidad limitada de gente) en *production*, así vemos qué piensan los usuarios y si es estable o no.

Configuramos alguna de las herramientas que vimos la clase pasada relacionada a monitoreo para ir verificando la respuesta de usuarios y la estabilidad antes de redirigir todo el tráfico.

Herramientas para monitoreo de logs

ELK Stack permite recolectar, almacenar y visualizar datos de logs. El mismo tiene tres componentes (E, L y K - Elasticsearch, Logstash y Kibana)

Elasticsearch es el núcleo del stack, una base de datos de búsqueda y análisis que permite almacenar y buscar grandes volúmenes de datos en tiempo real. Indexa los logs que vienen de múltiples fuentes y permite búsquedas rápidas en los datos históricos y actuales, ayudando a identificar patrones de errores y métricas de rendimiento.

Logstash es un motor de procesamiento de datos que ingiere, transforma y envía los logs a Elasticsearch. Se conecta a diversas fuentes de datos (servidores, aplicaciones, bases de datos), transforma los logs al formato deseado y los envía a Elasticsearch, permitiendo almacenar datos complejos antes de almacenarlos.

Kibana es la herramienta de visualización y dashboard de ELK, que facilita el análisis de datos almacenados en Elasticsearch mediante gráficos y paneles.

Herramientas para monitoreo de logs

Grafana Loki trabaja en conjunto con Grafana. Está optimizado para entornos en contenedores y se integra muy bien con clusters de Kubernetes.

Loki es el servicio de almacenamiento y gestión de logs. A diferencia de ELK, Loki almacena logs en texto plano y sólo indexa metadatos, lo que hace que consuma menos recursos. Es ideal para entornos de microservicios, especialmente Kubernetes, ya que puede recolectar logs directamente de los contenedores y organiza los logs en base a los labels de Kubernetes, como el nombre del namespace.

Herramientas para mantenimiento

Con respecto al mantenimiento, podemos destacar 4 campos distintos que son muy relevantes:

Despliegue continuo: Se pueden usar pipelines para garantizar el despliegue continuo, esto consiste principalmente en actualizaciones frecuentes y controladas en entornos de producción sin inactividad.

Monitoreo y alertas: Herramientas más relacionadas a TI como las que vimos la clase pasada, para monitorear el rendimiento de la aplicación en producción y recibir alertas por si pasó algo:

- **Prometheus**
- **Grafana**
- **Datadog**

Gestión de parches y actualizaciones: Automatización de instalación de parches de seguridad y actualizaciones de software, como por ejemplo:

- **Ansible** (apareció la clase pasada)
- **Chef**

Automatización de backups: Herramientas como las que ya vimos que realizan backups del sistema, intentando minimizar el riesgo por si ocurre algún desastre, como:

- **Veeam**
- **AWS Backup**