

# MAQUINAS VIRTUALES VS CONTENEDORES

Huichi Contreras, Franklin Carlos (2016056193), Huillca Umpiri, Willian (2015053793), Robles Flores, Anthony Richard (2016056192)

*Escuela Profesional de Ingeniería de Sistemas*

*Universidad Privada de Tacna*

*Tacna, Perú*

---

## Abstract

In this article we will learn concepts, differences and uses of virtualization and containers, this will allow us to choose the most recommendable option for the performance of a system that is being developed or to make changes in it.

## Keywords:

virtualization, containers, tools, processes, simulation, resources.

---

## 1. Resumen

En este artículo aprenderemos conceptos, diferencias y usos de la virtualización y contenedores, esto nos permitirá elegir la opción más recomendable para el desempeño de un sistema que se esté desarrollando o para realizar cambios en él.

## Palabras clave:

virtualización, contenedores, herramientas, simulación, procesos, recursos.

## 2. Introducción

Antes de ahondar en el concepto de contenedores, volvamos unos años atrás para recordar el nacimiento de la virtualización. A medida que el hard-

ware se hacía más poderoso nos encontramos con que el software no ocupaba todas las capacidades de la maquina física donde se encontraba siendo ejecutada (en muchos casos ni siquiera una fracción de estos recursos). Dado lo anterior se crearon recursos “virtuales” para simular el hardware base sobre el cual se ejecuta el software, permitiendo que múltiples aplicaciones puedan ser ejecutadas al mismo tiempo, cada una usando una fracción de los recursos del hardware físico disponible. A a esta “simulación” que permite de compartir recursos la denominamos comúnmente “virtualización”.

La mayoría de nosotros cuando escuchamos el concepto de virtualización pensamos inmediatamente en máquinas virtuales, pero es importante entender que este es solo un tipo de virtualización en el cual se habilita un sistema operativo el cual tiene la ilusión de que posee recursos dedicados para operar. Entendiendo lo anterior podemos ahora definir a los Contenedores como creadores de la percepción de un ambiente aislado exclusivo para la aplicación mientras que en la virtualización “tradicional” de máquinas virtuales la aplicación se ejecuta en un sistema operativo virtualizado donde convive con otros aplicativos.

### **3. Marco teorico**

#### *3.1. ¿Que es la virtualización?*

La virtualizacion es la creación de una versión virtual (no física) de algo. Esta basada en software, se puede aplicar a sistemas operativos, almacenamiento, servidores, aplicaciones, redes, etc. y es una manera de reducir gastos y aumentar eficiencia y agilidad en las empresas.

##### *3.1.1. Virtualizacion de servidores*

La virtualizacion de servidores ayuda a evitar ineficiencias ya que permite ejecutar varios sistemas operativos en una maquina fisica con maquinas virtuales con acceso a los recursos de todos. Tambien permite generar un cl uster de servidores en un unico recurso para así mejorar mucho mas la eficiencia y la reduccion de costes. Tambien permite el aumento de rendimiento de las aplicaciones y la disponibilidad al aumentar la velocidad en la carga de trabajo.

##### *3.1.2. Virtualizacion de escritorios*

La implementacion de escritorios virtualizados permite ofrecer a las sucursales o empleados externos de forma rapida y sencilla un entorno de trabajo y una reduccion de la inversion a la hora de gestionar cambios en estos.

### *3.1.3. Virtualizacion de red*

Se trata de reproducir una red fisica completa mediante software para poder ejecutar los mismos servicios que una red convencional y sus dispositivos. Cuentan con las mismas características y garantías que las redes físicas con las ventajas que nos ofrece la virtualizacion ademas de la liberacion del hardware.

### *3.1.4. Almacenamiento definido por software*

La virtualizacion del almacenamiento permite prescindir de los discos de los servidores. Los combina en depositos de almacenamiento de alto rendimiento y los distribuye como software. Este nuevo modelo permite aumentar la eficiencia en el guardado de datos.

## *3.2. Ventajas de la virtualizacion*

Como se ha podido apreciar en los tipos de virtualizacion presentados anteriormente, esta conlleva una mejora considerable tanto en el rendimiento, agilidad, flexibilidad, escalabilidad, etc. como en una reduccion considerable de los costes economicos y de tiempo y una simplificacion en la gestion de la infraestructura.

- Reduce los costes de capital y los gastos operativos.
- Minimiza o elimina los tiempos de inactividad
- Aumenta la productividad, la eficiencia, la agilidad y la capacidad de respuesta
- Implementa aplicaciones y recursos con mas rapidez.
- Garantiza la continuidad del negocio y la recuperacion ante desastres.
- Simplifica la gestion del centro de datos.

## *3.3. Contenedores*

Un contenedor de software se puede considerar como una aplicación para el servidor. Estos se encargan de proporcionar a las aplicaciones archivos, variables y bibliotecas que sean necesarios para ejecutarse y maximiza su portabilidad. Para poder instalar una aplicación, el contenedor se carga en el ordenador en un formato portable o imagen (Image) que incluye todos los datos necesarios para su funcionamiento y, en el ordenador, se inicia en

un entorno virtual. Los contenedores permiten que estos equipos de desarrollo alcancen una eficiencia muy alta en la entrega y el despliegue de software, al solucionar muchos de los retos que presenta la virtualización tradicional.

Beneficios:

- Ahorro de espacio y consumo. No se tendrá la necesidad de crear máquinas virtuales en las que instalarlas.
- Reutilización. Se puede crear tantas instancias como necesitemos, destruirlas y reproducir el entorno inicial.
- No “ensucia”. No se tendrá que instalar dentro de nuestro equipo con la problemática que ello conlleva en algunos casos.
- Compartir. Estas imágenes las podremos distribuir comodamente entre los componentes de nuestro equipo.
- Se obtiene mayor modularidad. El desarrollo con contenedores es ideal para un enfoque basado en microservicios para el diseño de aplicaciones.

Ventajas de los contenedores:

Los contenedores de aplicaciones “empaquetan” los recursos necesarios para el funcionamiento de una aplicación sin embargo, las mayores ventajas de tales contenedores radican, sobre todo, en la gestión y en la automatización de software basado en contenedores.

- Instalación más sencilla: los contenedores de software se inician a partir de imágenes o representaciones portables de un contenedor, incluyendo un programa y todos los componentes requeridos. De esta manera se compensan las diferencias entre sistemas operativos. Su instalación se reduce a la introducción de una línea de comando.
- Independiente de la plataforma: las imágenes se pueden transportar cómodamente de un sistema a otro y se caracterizan por una considerable independencia de la plataforma. Lo único que se necesita para iniciar un contenedor desde una imagen es un sistema operativo que soporte contenedores.

- Pérdidas por virtualización mínimas: con un Linux y Docker container, la instalación de contenedores requiere alrededor de 100 MB y unos pocos minutos, aunque no es solo esto a lo que se oponen los administradores de sistemas. Mientras que la virtualización de hardware trae consigo una pérdida de rendimiento para el hipervisor y otros sistemas operativos, los contenedores, al prescindir de todo esto, reducen esta pérdida al mínimo.
- Aplicaciones aisladas: cada programa funciona independientemente de otros contenedores, de forma que aplicaciones con requerimientos opuestos pueden funcionar en paralelo en el mismo sistema.
- Administración y automatización unitarias: debido a que en una plataforma como Docker todos los contenedores son gestionados con las mismas herramientas, es posible automatizar todas las aplicaciones de manera centralizada. Por esto, estas soluciones están indicadas sobre todo para arquitecturas de servidor en las cuales los componentes están distribuidos en varios servidores, de forma que se carga con los pesos de instancias diferentes. En estos ámbitos de aplicación, el Docker container dispone de herramientas con las cuales configurar automatismos. Esto posibilita, por ejemplo, iniciar instancias nuevas de forma automática en momentos puntuales de sobrecarga.

Al final, el uso de los contenedores es muy conveniente, también puede traer un nivel bajo de seguridad estaríamos dejando de usar sistemas operativos separados. Pues, los contenedores no son tan herméticos como las máquinas virtuales con sistema operativo propio. En consecuencia, aunque los contenedores constituyen una alternativa para la virtualización de hardware, de momento no la pueden sustituir por completo.

## 4. Funcionamiento de Docker

### 4.1. *Arquitectura*

Docker usa una arquitectura cliente servidor. El cliente de Docker se comunica con el Daemon de Docker para crear, ejecutar y distribuir los contenedores. Tanto el cliente como el Daemon pueden estar en el mismo sistema o pueden conectarse remotamente. Como Docker usa el kernel de Linux para

su ejecución, si el sistema operativo del sistema no es este, se deberá usar una pequeña capa extra en la arquitectura de tipo VM (boot Docker) para poder correr Docker en la máquina.

#### 4.2. *Cliente de Docker*

Es la principal interfaz de usuario para Docker. Acepta los comandos del usuario y se comunica con el Daemon de Docker.

#### 4.3. *Imágenes de Docker (Docker Images)*

Las imágenes de Docker son plantillas de solo lectura, que nos permitan crear contenedores basados en su configuración.

#### 4.4. *Registros de Docker (Docker Registries)*

Los registros de Docker guardan las imágenes. Estos son repositorios públicos o privados donde se pueden subir o descargar imágenes. Sería similar a GitHub para imágenes de Docker (Docker Hub).

### 5. Creación de Imágenes

Como se comenta en el punto anterior, las imágenes de Docker son las plantillas para poder levantar los contenedores. Por eso la importancia de saber crear imágenes y personalizarlas ya que solo permiten lectura y los cambios que hagamos en los contenedores no se verán reflejados en estas. La manera más sencilla de crear una imagen es descargarla del Docker Hub con el comando explicado anteriormente:

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Este comando nos permite descargar una imagen en una versión concreta o tag dependiendo de nuestras necesidades. Por defecto, si no se pone nada, descargará la última.

En la figura podemos apreciar cómo se descarga la última versión de la imagen de MongoDB y se genera la imagen. Una vez obtenida la imagen se

```
alfredo@alfredo:~$ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo

8ad8b3f87b37: Pull complete
5947be99d359: Pull complete
d5a4577c6007: Pull complete
acb97586a200: Pull complete
d11260d069a3: Pull complete
bf102d35e390: Pull complete
f4964f6a9bfa: Pull complete
8b392ba3e8bf: Pull complete
c023b73abe56: Pull complete
Digest: sha256:8ff7bd4acdb123e3922a7fae7f73efa35fba35af33fad0de946ea31370a23cc4
Status: Downloaded newer image for mongo:latest
alfredo@alfredo:~$
```

(a) Docker pull

```
alfredo@alfredo:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mongo	latest	48b8b08dca4d	2 weeks ago	366.4 MB

```
alfredo@alfredo:~$
```

(b) Docker images

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

pasara a levantar el contenedor para poder ejecutar el servicio con otro de los comandos explicados.

En la figuras 2.4 se puede apreciar como utilizando el comando `docker ps` que permite ver que contenedores estan levantados, no hay ninguno (a) y como al inicializar con el comando `docker run mongo` levanta el servicio (b) y esta vez sí aparece el contenedor (c).

La segunda manera de crear y personalizar imagenes es mediante un DockerFile, que es un documento de texto donde se encuentran los comandos que se deben ejecutar para generar nuestra imagen.

El comando `docker build` comunica al Daemon de Docker que debe de leer el DockerFile del directorio actual y seguir las instrucciones linea por linea para la creacion de nuestra imagen. Este proceso va pintando los resultados por pantalla y generando imagenes intermedias para obtener asi una cache que nos permitira en caso de errores, una vez corregido el DockerFile,

```
alfredo@alfredo:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
alfredo@alfredo:~$
```

(a) Docker ps

```
alfredo@alfredo:~$ docker run mongo
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=d62f7dce38c2
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] db version v3.2.9
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] git version: 22ecae3b40c85fc7cae7d56e7d6a82fd01188c
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1t  3 May 2016
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] allocator: tcmalloc
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] modules: none
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] build environment:
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten]   distmod: debian1
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten]   distarch: x86_64
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten]   target arch: x86_64
2016-09-19T14:54:09.623+0000 I CONTROL [initandlisten] options: {}
2016-09-19T14:54:09.631+0000 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=3G,session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoints=(wait=60,log_size=2G),statistics_log=(wait=0),
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] **           We suggest setting it to 'never'
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten]
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten] **           We suggest setting it to 'never'
2016-09-19T14:54:10.421+0000 I CONTROL [initandlisten]
2016-09-19T14:54:10.422+0000 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'
2016-09-19T14:54:10.422+0000 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-09-19T14:54:10.492+0000 I NETWORK [initandlisten] waiting for connections on port 27017
```

(b) Docker run

```
alfredo@alfredo:~$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
eee379c1602   mongo     "/entrypoint.sh mongo"   4 seconds ago Up 3 seconds  27017/tcp    kickass_pike
alfredo@alfredo:~$
```

(c) Docker ps

continuar desde el punto conflictivo.

```
FROM docker/whalesay:latest
CMD echo "Proyecto CoIoTe" | cowsay
```

Aquí tenemos un ejemplo sencillo de DockerFile que nos servirá para explicar de una manera rápida como crearlos.

FROM indica la imagen base que va a utilizar para seguir futuras instrucciones. Buscamos si la imagen se encuentra localmente, en caso de que no, la descargamos. En nuestro ejemplo utilizamos la última versión de la imagen docker/whalesay.

## 6. Docker Compose

Docker Compose es un orquestador que nos permite ejecutar aplicaciones que utilicen varios contenedores a la vez. Se crea un archivo docker-



compose.yml donde se configuraran todos los servicios necesarios para nuestra aplicacion. Una vez ejecutado este archivo nos generara todas las imagenes y con estas los contenedores especificados a la vez que arrancara la aplicacion. Los comandos que se utilizaran seran similares a los utilizados en la creacion de imagenes.

Para ejecutar el servicio y levantar todos los contenedores.

```
docker-compose up
```

Para detener el servicio y detener los contenedores.

```
docker-compose stop
```

## 7. Conclusiones

- Los Contenedores son un método de virtualización de un sistema operativo que permite ejecutar una aplicación junto con sus elementos dependientes, en un ambiente aislado e independiente.
- La virtualizacion lo que hace es crear a traves del software una version virtual de un recurso, como virtualiza de manera completa un software, consume los recursos del hardware tal como lo hicieramos en una maquina física reduciendo el rendimiento y el performance de la maquina que aloja a la virtual

## 8. Autores

## 9. Planteamiento del problema

Lorem Ipsum

Roles	Integrantes
Product owner	Gonzales Cave, Angel Gabriel
Scrum master	Huichi Contreras, Franklin Carlos
Equipo scrum	Condori Quispe, Yhónn Joel — Pastor Mendoza, José Edilberto

Cuadro 1: Integrantes del equipo

### 9.1. ***Problema***

Lorem Ipsum

### 9.2. ***Justificacion***

Lorem Ipsum

### 9.3. ***Alcance***

Lorem Ipsum

### 9.4. ***General***

Lorem Ipsum

### 9.5. ***Especificos***

- Analizar cuáles son las peticiones de soporte técnico mas solicitadas por los usuarios, para obtener la informacion necesaria y definir los requerimientos para el correcto desarrollo del sistema
- Mejorar la productividad de la empresa informando a los usuarios sobre como solucionar los problemas comunes.

## 10. **Referentes Teoricos**

Lorem Ipsum

A continuación se describe la conformación actual de ITIL v3 :

- Libro 1 Estrategia del Servicio Propone tratar la gestión de servicios no sólo como una capacidad sino como un activo estratégico
- Libro 2 Diseño del Servicio Cubre los principios y métodos necesarios para transformar los objetivos estratégicos en portafolios de servicios y activos.

- Libro 3 Transición del Servicio Cubre el proceso de transición para la implementación de nuevos servicios o su mejora
- Libro 4 Operación del Servicio Cubre las mejores prácticas para la gestión del día a día en la operación del servicio.
- Libro 5 Mejora Continua del Servicio Proporciona una guía para la creación y mantenimiento del valor ofrecido a los clientes a través de un diseño, transición y operación del servicio optimizado.

[3]

## 11. Desarrollo de la propuesta

### 11.1. *Tecnología de información*

- NoSQL el término fue inicialmente utilizado en el año 1998, y fue para denominar una base de datos relacional que no utilizaba el lenguaje SQL para funcionar. Características principales :

Evitar la complejidad innecesaria Las bases de datos relacionales proporcionan gran cantidad de funcionalidades y restricciones para mantener la consistencia de los datos, en ciertos casos, mucho más de lo necesario.

Alto rendimiento Muchas bases de datos NoSQL proporcionan un rendimiento superior al que ofrecen los sistemas RDBS convencionales.

[2]

- Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.
- Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.

### 11.2. *Metodología, técnicas usadas*

SCRUM es una metodología para gestión, mejora y mantenimiento de un sistema nuevo o existente. SCRUM se concentra en cómo los miembros del equipo deberían funcionar a fin de producir un sistema flexible en un entorno que cambia constantemente. [1]

### 11.2.1. *Inicio*

- Planeamiento: Consiste en establecer la visión, el presupuesto, forma de financiamiento y el backlog del producto. En esta fase se selecciona que funcionalidad es la mas apropiada para desarrollo inmediato. También se establece el equipo de trabajo, se evalúan las herramientas de desarrollo y se define la fecha de entrega (es una fecha aproximada). [1]
- Arquitectura : Esta fase consiste en la conceptualización y análisis. Si el proyecto se trata de la mejora de un nuevo sistema, sólo se realiza un análisis limitado. Se realiza un diseño de alto nivel para actualizar los modelos del dominio y reflejar el contexto del nuevo sistema y los requerimientos y las modificaciones necesarias de la arquitectura del sistema. Los diseñadores y arquitectos dividen el proyecto en paquetes basándose en los Ítems del backlog. En la jerga de SCRUM se llaman "paquetes" a los objetos o componentes que necesitan cambiarse en cada iteración. [1]

### 11.2.2. *Desarrollo*

En esta etapa se realiza el desarrollo propiamente dicho. También se la conoce como "Ingeniería concurrente". La misma se divide en iteraciones que proveen como resultado funcionalidades incrementales al fin de cada una de ellas. Dichas iteraciones se llaman sprints.

Un sprint dura aproximadamente entre una semana y 30 días. Cada sprint incluye las fases tradicionales del desarrollo de software: requerimientos, análisis, diseño, desarrollo y entrega. Durante un sprint no se utilizan diagramas de gantt para seguimiento de tareas (estos parten del supuesto que las tareas de un proyecto se pueden identificar y ordenar), debido a que el desarrollo es semi-caótico y cambiante como para que se le aplique un proceso definido.

Durante un sprint no se pueden cambiar los miembros del equipo scrum. Tampoco pueden introducirse cambios durante un sprint (si surge algún cambio se incluya en el backlog del producto, pero no en el del sprint).

El scrum master mantiene el sprint backlog. Actualiza las tareas finalizadas y para las que no lo están, el tiempo que el equipo piensa que tomara para terminarlas. [1]

### 11.2.3. *Reuniones Scrum*

Durante un sprint, todos los días se realizan reuniones llamadas "SCRUM". El objetivo de las mismas es quitar los impedimentos que le surgen a los

miembros del equipo scrum. Cada una de ellas dura aproximadamente 15 minutos. A cada miembro del equipo scrum se le pregunta:

- ¿Qué hizo durante las ultimas 24 horas?
- ¿Qué planea hacer las proximas 24 horas?
- ¿Qué obstaculos se le han presentado en las últimas 24 horas ?

Estas reuniones deben realizarse obligatoriamente.[1]

#### 11.2.4. Cierre

Esta etapa comienza cuando el equipo de management decide que las variables de entorno, tales como los requerimientos se han completado. En esta etapa se genera la documentación final, se realiza el testing pre-lanzamiento y el lanzamiento propiamente dicho. [1]

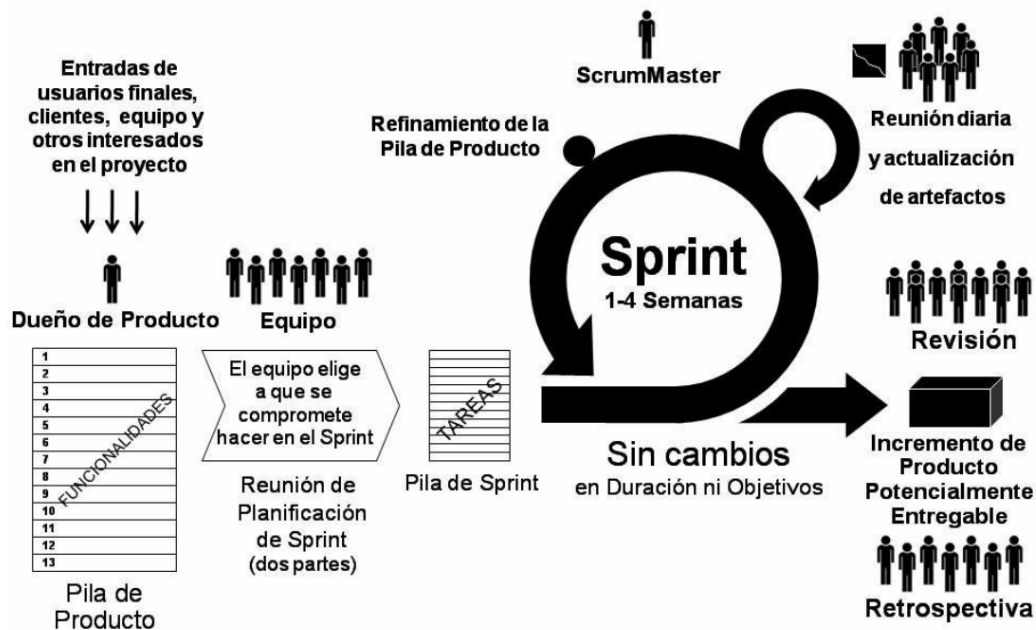


Figura 1: Cronograma del proyecto

## 12. Cronograma

## Referencias

- [1] Caso, N. (2004). Scrum development process. Recuperado de <https://bit.ly/2ouTR57>. Accedido el 09-10-2019.
- [2] Herranz, R. (2014). Bases de datos nosql : Arquitectura y ejemplos de aplicacion. Recuperado de <https://core.ac.uk/download/pdf/44310803.pdf>. Accedido el 10-09-2019.
- [3] Ríos, S. (2007). *ITIL v3*. Biagle Management.

## Cronograma del proyecto

Powered by  **monday.com**  
Click here to start your free trial

### Especificar requisitos y evaluar proyecto

Name	Owner	Status	Timeline - Start	Timeline - End	Priority
Recolección de requisitos	Huichi		2019-10-09	2019-10-12	High
Incorporar mejoras en requisitos	Huichi, Gonzalez		2019-10-13	2019-10-14	Medium
Elaborar el cronograma	Huichi		2019-10-15	2019-10-16	High
Conseguir aprobación para continuar	Huichi		2019-10-17	2019-10-17	High
Asegurar los recursos necesarios	Condori		2019-10-18	2019-10-20	Medium

### Análisis y Diseño

Name	Owner	Status	Timeline - Start	Timeline - End	Priority
Análisis de requisitos	Huichi, Gonzalez		2019-10-21	2019-10-24	High
Diseño de arquitectura	Huichi, Gonzalez		2019-10-25	2019-10-27	Medium
Desarrollo de prototipos	Huichi, Gonzalez		2019-10-27	2019-11-01	Medium
Revisión de diseño	Condori		2019-10-27	2019-11-01	Medium
Incorporar mejoras en el diseño	Huichi, Gonzalez		2019-11-02	2019-11-05	Low
Conseguir aprobación para aprobar	Huichi		2019-11-06	2019-11-06	High

### Programación

Name	Owner	Status	Timeline - Start	Timeline - End	Priority
Revisión en especificación de diseño	Condori, Pastor		2019-11-03	2019-11-08	High
Diseño detallado de módulos	Angel		2019-11-07	2019-11-10	High
Codificación	Condori, Pastor, Gonzalez		2019-11-10	2019-11-15	Medium
Pruebas de programación	Condori		2019-11-10	2019-11-15	Medium

### Pruebas

Name	Owner	Status	Timeline - Start	Timeline - End	Priority
Probar módulos y sus componentes	Condori		2019-11-16	2019-11-19	Medium
Detectar fallos	Condori		2019-11-16	2019-11-22	High
Corregir fallos	Condori		2019-11-16	2019-11-22	High
Volver a probar código modificado	Condori		2019-11-16	2019-11-22	High

### Despliegue

Name	Owner	Status	Timeline - Start	Timeline - End	Priority
Establecer estrategia de despliegue	Gonzalez, Pastor		2019-10-27	2019-11-09	High
Desarrollar metodología	Huichi, Gonzalez		2019-10-27	2019-11-22	High
Desplegar el software	Condori, Pastor		2019-11-23	2019-11-26	Medium

Figura 2: Cronograma del proyecto