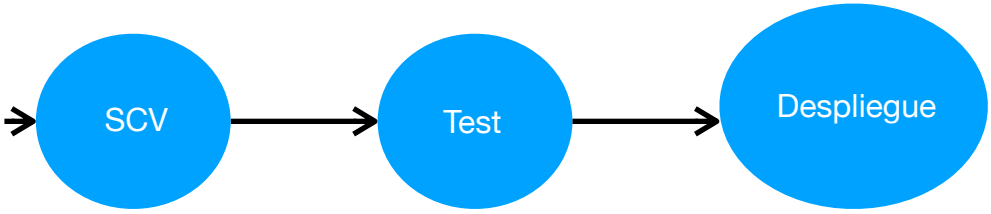
Charla de 0 a Cloud



Sistema de Control de Versiones (SCV) —> Ejemplo: git

- Cuando haces un cambio en el repositorio —> quieres que ese cambio se añada a la rama maestra (merge)
- *Call reviews* (distintos tipos) —> como pasarela

Tests

Una vez que los cambios están revisados se pasan los tests. Tú tienes una forma de correr los test, se comprueban si han pasado o no (añadido a la historia de ese *pull request* e indica si se han pasado). Al pasar los tests, los tests se corren en una plataforma completamente distinta a la de los desarrolladores, por eso, si falla es para todos y viceversa. Los tests se corren en el momento en el que quieres que ese código se ponga en la rama maestra.

Commits

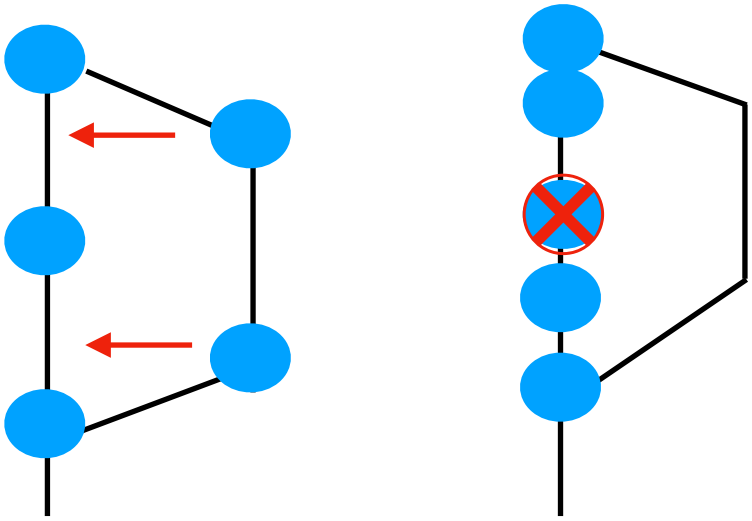


FIGURA 1: DIAGRAMA VERSIONES GIT

Se pasan los tests, una vez que ya se añade todo ya se vuelve a pasar un test a la totalidad del proyecto.

- Comprar máquina más grande: escalado vertical
- Coger una y tener más pequeñitas (dividir): escalado horizontal

Entiendo la nube

Software-as-a-Service (SaaS)

Básicamente se trata de cualquier servicio basado en la web. Ejemplos populares de SaaS son Google Docs, Dropbox o Gmail. En este tipo de servicios nosotros accedemos normalmente a través del navegador sin atender al software. Todo el desarrollo, mantenimiento, actualizaciones, copias de seguridad es responsabilidad del proveedor. En este caso tenemos poco control, nosotros nos situamos en la parte más arriba de la capa del servicio. Si el servicio se cae es responsabilidad de proveedor hacer que vuelva a funcionar.

Platform-as-a-Service (PaaS)

En este caso nuestra única preocupación es la construcción de nuestra aplicación, ya que la infraestructura nos la da la plataforma. Es un modelo que reduce bastante la complejidad a la hora de desplegar y mantener aplicaciones ya que las soluciones PaaS gestionan automáticamente la escalabilidad usando más recursos si fuera necesario. Ejemplos populares son Google App Engine que permite desarrollar aplicaciones en Java o Python desplegándolas en la infraestructura que provee Google, cosa que también hace Heroku con Rails y Django.

Infraestructure-as-a-Service (IaaS)

En este caso con IaaS tendremos mucho más control que con PaaS, aunque a cambio de eso tendremos que encargarnos de la gestión de infraestructura. El ejemplo perfecto es el proporcionado por Amazon Web Service (AWS) que no provee una serie de servicios como EC2 que nos permite manejar maquinas virtuales en la nube o S3 para usar como almacenamiento.

FIGURA 2: ENTENDIENDO LA NUBE

Demo de Python con Flask

- Create repository

```
$ git clone git@github.com:Gecofer/holaEtsiit.git
```

- Create issue

- Create nueva rama

```
$ git checkout -b feature/1
Cambiado a nueva rama 'feature/1'
```

- Instalar python3 sino está instalado

```
$ python3 --version
```

- Crear entorno virtual

```
$ virtuales -p python3 env
$ pip3 install virtual
```

```
$ virtualenv -p python3 env
```

- Hacer uso del microframework Flask de Python: http://flask.pocoo.org

```
$ pip install flask
```

- Cogemos el código que viene en el enlace y lo pegamos en un fichero nuevo “server.py”:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!”

Importa flask del paquete que hemos
```

```
$ git add server.py
$ git commit -m “#1 Running something”
$ git push --set-upstream origin feature
```

- Create pull request

- File Changes -> te aperece “server.py” y te aparece un + y le das al + en “@app.route”

Vamos hacer un test

- Para ello usaremos unittest con Python: https://docs.python.org/3/library/unittest.html

- Nos vamos a la rama master —> Nueva issue

```
$ git checkout master
$ git pull
$ git checkout feature/2
```

- Crear *Vim test.py:* basado en el ejemplo básico de https://docs.python.org/3/library/unittest.html (testear nuestro servidor)

```
import unittest
import server # server.py

class TestMyServer(unittest.TestCase):

      def test_first(self): # añadimos un test
            response = server.hello()
            self.assertIn("Hello", responde, "Hello was not found in response")
# comprobamos si hello va a estar dentro

if __name__ == '__main__':
    unittest.main()
```

```
$ bat server.py
```

- Assert te sirve de testeo

```
$ git status
$ git push --set-upstream origin feature/2
$ git commit -m "Closes #2 Bye"
```

- Closes #2: cierra el issue

- Hacemos uso de Travis: lanza un contenedor en el cambio del código

```
$ vim travis.yml
$ git add travis.yml
```

- Hacemos uso de Heroku

```
$ git npm install -g heroic
// sino la creamos con el -g, nos va a crear una carpeta
```