

# INGENIERÍA MECATRÓNICA



DI\_CERO

DIEGO CERVANTES RODRÍGUEZ

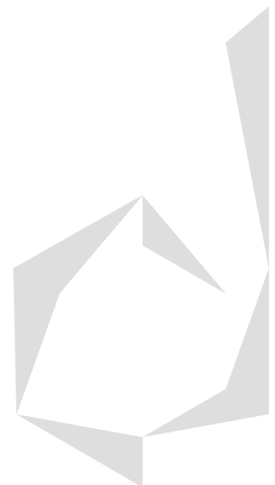
PROGRAMACIÓN: MANEJO DE DATOS Y RECURSOS

PYTHON 3.9.7, DJANGO, DOCKER, KUBERNATES, LINUX, ETC.

## Introducción al Lenguaje de Programación Python

# Contenido

Python .....	2
Django y su Arquitectura MTV (Model - Template – View) .....	2
API REST (Aplication Programming Interface) con Python: .....	2
Jinja 2 – Aditivo a Django para la creación de Interfaces Web con Python:.....	3
Django Admin.....	5
Postman .....	6
Docker: Contenedores de distintos OS .....	6
Docker - Kubernetes .....	8
Python - Distribuciones de Linux .....	9
Python - TDD (Test Driven Development) con frameworks .....	9
Pruebas Unitarias y Pruebas de Integración .....	10



# Python

Python es uno de los lenguajes de programación más utilizados en el mercado ya que es muy versátil y fácil de usar para ya sea inteligencia artificial, operaciones matemáticas complejas (matriciales, derivativos, integrales, etc.), visión artificial, análisis de datos, manejo de Backend, etc.

## Django y su Arquitectura MTV (Model - Template – View)

Existen varios usos y frameworks de Python para el manejo de datos, uno de los más importantes para el manejo de datos en el Backend es Django, que a su vez usa una arquitectura llamada MVT (Model - View - Templates) o a veces igual nombrada MVT, pero se refiere a lo mismo.

### Arquitectura MTV (Model-Template-View) o MVT:

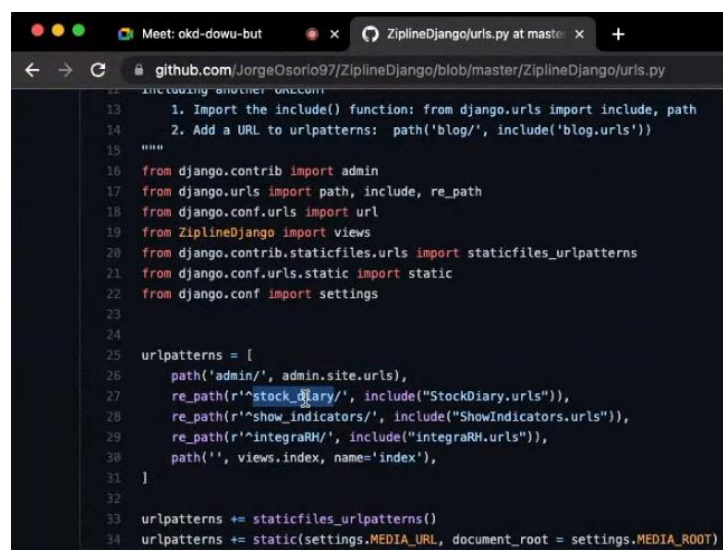
- **Model:** Crea y maneja las tablas de la base de datos, jalando información de un lado a otro.
- **Templates:** Es la parte visual que recopila datos, osea la interfaz ya sea de una página web o aplicación móvil.
- **View:** Parte de la arquitectura que conecta lo que venga del template y se lo pasa al model.

### Python y Django (Framework de manejo de datos):

- **Python:** Es el lenguaje de programación base del manejo de datos.
- **Django:** Framework que realiza la conexión con la base de datos, realizando así:
  - **Manejo de rutas:** Se refiere a la creación de URLs y modelos para la conexión a la base de datos (el modelo se refiere a una tabla de la base de datos).
  - **Creación de APIS por medio de Django REST Framework:** Es una herramienta de Django para la creación de API REST, Django es famoso por su manejo de formularios.

## API REST (Aplication Programming Interface) con Python:

Cuando se crea una API con Python literal lo que se hace es describir las reglas para comunicar APIS con cualquier software que la quiera utilizar, esto se realiza usando:



```
13 1. Import the include() function: from django.urls import include, path
14 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15
16 from django.contrib import admin
17 from django.urls import path, include, re_path
18 from django.conf.urls import url
19 from ZiplineDjango import views
20 from django.contrib.staticfiles.urls import staticfiles_urlpatterns
21 from django.conf.urls.static import static
22 from django.conf import settings
23
24
25 urlpatterns = [
26     path('admin/', admin.site.urls),
27     re_path(r'^stock_diary/', include("StockDiary.urls")),
28     re_path(r'^show_indicators/', include("ShowIndicators.urls")),
29     re_path(r'^integraRH/', include("integraRH.urls")),
30     path('', views.index, name='index'),
31 ]
32
33 urlpatterns += staticfiles_urlpatterns()
34 urlpatterns += static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)
```

- **JSON:** Estructura de datos basada en JavaScript para compartir una serie de datos que contienen un valor (value) y un identificador (key).
- **Verbos HTTP:** Get, Post, Put, Patch, etc.

## Jinja 2 – Aditivo a Django para la creación de Interfaces Web con Python:

El framework de Jinja 2 sirve para crear interfaces que se mezclan con el código de HTML para que se conecten entre ellas y así se creen las rutas URL a donde se manden los datos de una página web, creando así las plantillas del Framework Django.



```

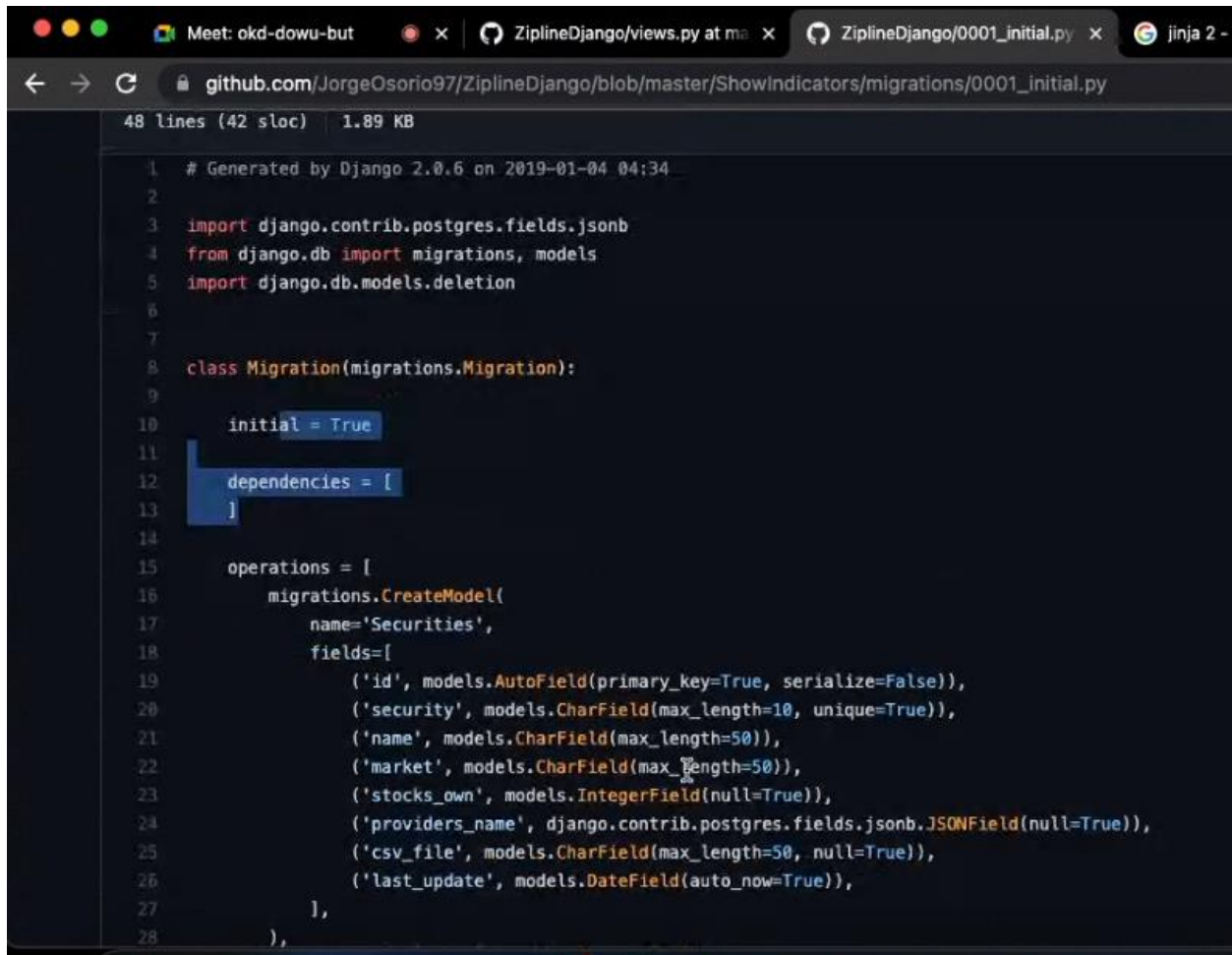
93     strategies_utils.createStrategy(pd.read_csv('static/historicos/'+ secl+'csv_file')).
94     return JsonResponse({'succes': 'true'})
95     return render(request, 'show_indicators/strategy_creator.html')
96
97 def addSecurity(request):
98     if request.method == "POST":
99         print("add_security_file")
100         print(request.FILES)
101         data = pd.read_csv(request.FILES['file'])
102         #print(data.head())
103         # TODO: agregar try para evita cargar errores
104         # TODO: agregar revision de securities repetidos
105         # TODO: agregar descarga de plantilla
106         for index, row in data.iterrows():
107             temp = Securities()
108             temp.security = row['security']
109             temp.name = row['name']
110             temp.market = row['market']
111             temp.stock_own = row['stocks_own']
112             temp.providers_name = json.dumps({row['provider']:row['ticker']})
113             temp.csv_file = row['csv_file']
114             temp.save()
115
116         form = UploadFileForm()
117         #print(form)
118         return render(request, 'show_indicators/add_security.html', {'form': form})
119
120 @csrf_exempt
121 def newSecurity(request):
122     print("new_security")

```

```
Meet: okd-dowu-but x ZiplineDjango/views.py at mas x ZiplineDjango/add_security.ht x jinja 2 - Buscar con
github.com/JorgeOsorio97/ZiplineDjango/blob/master/templates/show_indicators/add_security.html
10 <div class="container">
11 <div class="jumbotron">
12 <h1 class="display-4 text-white">Agregar Acciones</h1>
13 <form>
14     {% csrf_token %}
15     <label for="security_name">Nombre de la Empresa</label>
16     <input type="text" id="security_name" class="form-control" placeholder="ex: Apple Inc."/>
17     <label for="security_code">Codigo</label>
18     <input type="text" id="security_code" class="form-control" placeholder="ex: AAPL"/>
19     <select id="market">
20         <option value="BMV">BMV</option>
21     </select><br>
22     <label for="csv_file">Archivo CSV</label>
23     <input type="text" id="csv_file" class="form-control" placeholder="ex: aapl.csv"/>
24
25
26
27     <button type="button" class="btn btn-success " id="add_security"> Estrategia Recomendada</button>
28
29 </form>
30
31 <h2>Subir con archivo</h2>
32
33 <form method="POST" action="{% url 'ShowIndicators:addSecurity' %}" enctype="multipart/form-data">
34     {% csrf_token %}
35     {{ form }}
36     <input type="submit" value="Enviar"/>
37 </form>
38
39 </div>
```

```
Meet: okd-dowu-but x ZiplineDjango/views.py at ma x ZiplineDjango/models.py at n x
github.com/JorgeOsorio97/ZiplineDjango/blob/master/ShowIndicators/models.py
1 from django.db import models
2 from django.contrib.postgres.fields import JSONField
3
4 class Securities(models.Model):
5     id = models.AutoField(primary_key=True)
6     security = models.CharField(max_length=10, unique=True)
7     name = models.CharField(max_length=50)
8     market = models.CharField(max_length=50)
9     stocks_own = models.IntegerField(null=True)
10    providers_name = JSONField(null=True)
11    csv_file = models.CharField(max_length=50, null=True)
12    last_update = models.DateField(auto_now=True)
13    best_strategy = models.ForeignKey('Strategies', on_delete=models.CASCADE, null=True)
14
15
16    def __str__(self):
17        return self.name + " - " + self.security
18
19 class Strategies(models.Model):
20     id = models.AutoField(primary_key=True)
21     security = models.CharField(max_length=50)
22     securities_tested = models.PositiveIntegerField(null=True)
23     strategy = JSONField()
24     percentage_up = models.FloatField(default=None)
25     last_modified = models.DateTimeField(auto_now=True)
26     max_point = models.FloatField(null=True)
27     min_point = models.FloatField(null=True)
28     trades = models.IntegerField(null=True)
```

Así se crean las colecciones de datos desde Django, no se debe hacer nada desde la base de datos, todo se hace desde Django.

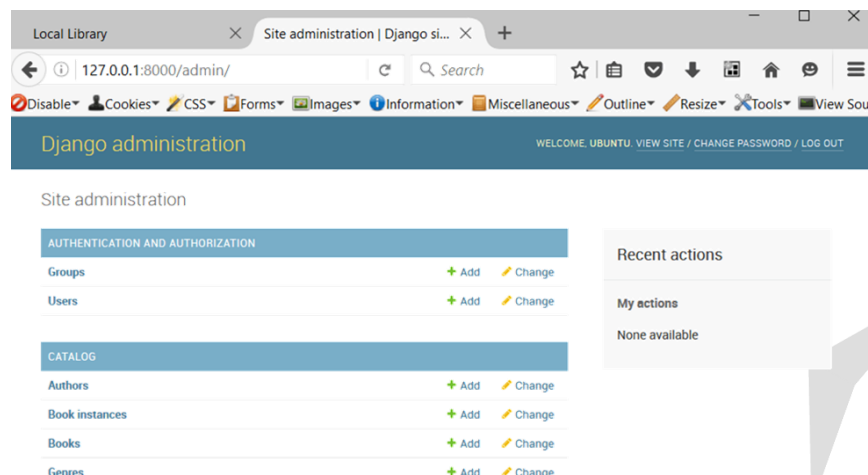


The screenshot shows a web browser displaying a GitHub repository page for 'ZiplineDjango'. The file '0001\_initial.py' is open, showing a Django migration script. The script is 48 lines long and 1.89 KB in size. It was generated by Django 2.0.6 on 2019-01-04 04:34. The code defines a 'Migration' class with 'initial = True', 'dependencies = []', and a list of 'operations' including 'CreateModel' for a 'Securities' model. The model has fields: 'id' (AutoField, primary key), 'security' (CharField, max\_length=10, unique), 'name' (CharField, max\_length=50), 'market' (CharField, max\_length=50), 'stocks\_own' (IntegerField, null=True), 'providers\_name' (JSONField, null=True), 'csv\_file' (CharField, max\_length=50, null=True), and 'last\_update' (DateField, auto\_now=True).

```
1 # Generated by Django 2.0.6 on 2019-01-04 04:34
2
3 import django.contrib.postgres.fields.jsonb
4 from django.db import migrations, models
5 import django.db.models.deletion
6
7
8 class Migration(migrations.Migration):
9
10     initial = True
11
12     dependencies = [
13     ]
14
15     operations = [
16         migrations.CreateModel(
17             name='Securities',
18             fields=[
19                 ('id', models.AutoField(primary_key=True, serialize=False)),
20                 ('security', models.CharField(max_length=10, unique=True)),
21                 ('name', models.CharField(max_length=50)),
22                 ('market', models.CharField(max_length=50)),
23                 ('stocks_own', models.IntegerField(null=True)),
24                 ('providers_name', django.contrib.postgres.fields.jsonb.JSONField(null=True)),
25                 ('csv_file', models.CharField(max_length=50, null=True)),
26                 ('last_update', models.DateField(auto_now=True)),
27             ],
28         ),
```

## Django Admin

Django admin es un administrador como el de wordpress, donde se pueden ver los modelos de la base de datos de una manera sencilla, a esto se accede en la url, después de poner **/admin**



# Postman

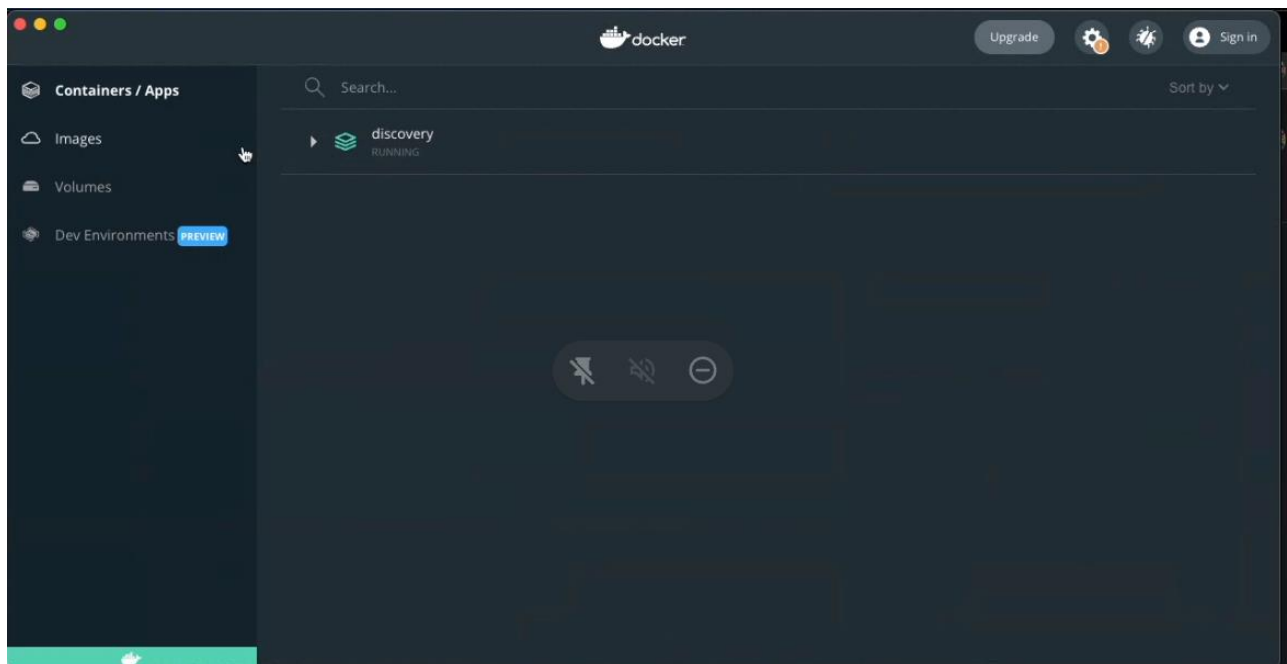
Postman es una herramienta utilizada para probar las APIs, cada software necesita una API para interactuar con los demás programas sin dar a conocer su código fuente.



## Docker: Contenedores de distintos OS

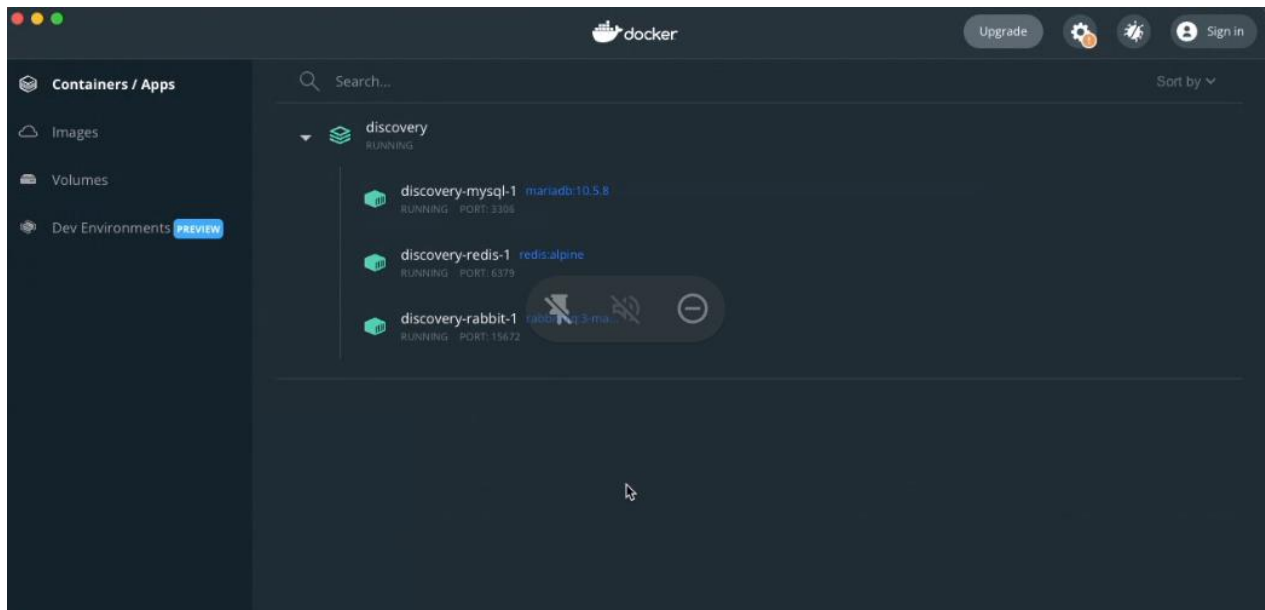
Docker es una interfaz que se controla desde consola y sirve para correr un mismo programa en varios sistemas operativos, se le llama comúnmente como contenedor y es porque en cada sistema operativo la ejecución de distintos programas se debe correr de distinta forma.

Esta herramienta es una evolución a las máquinas virtuales, ya que pesa mucho menos. Al crear un contenedor se elige un sistema operativo específico, se elige cada configuración con un lenguaje o librerías en específico y a eso se le llama imagen, a cada imagen se le pueden ir agregando cosas, como librerías.





Se puede manejar cada servicio en un servidor distinto, para esto también se usa Docker, para simular esto, solo en AWS se puede correr en la vida real las imágenes de Docker, sino solo se sube el código al servidor.

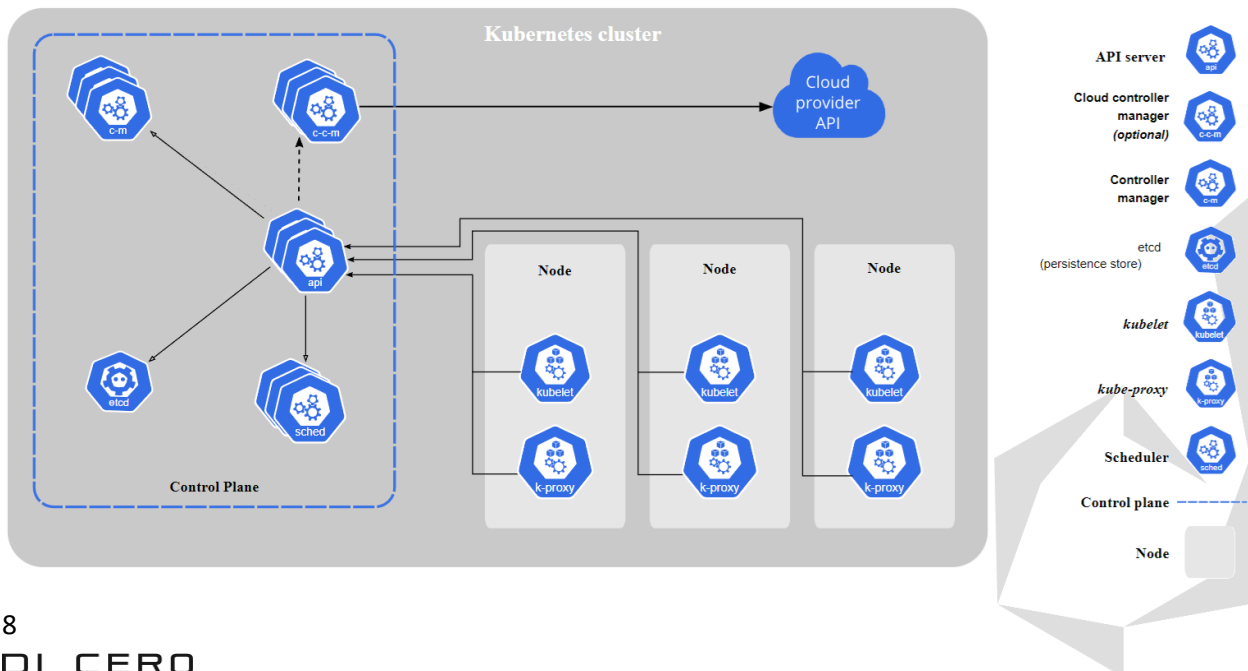
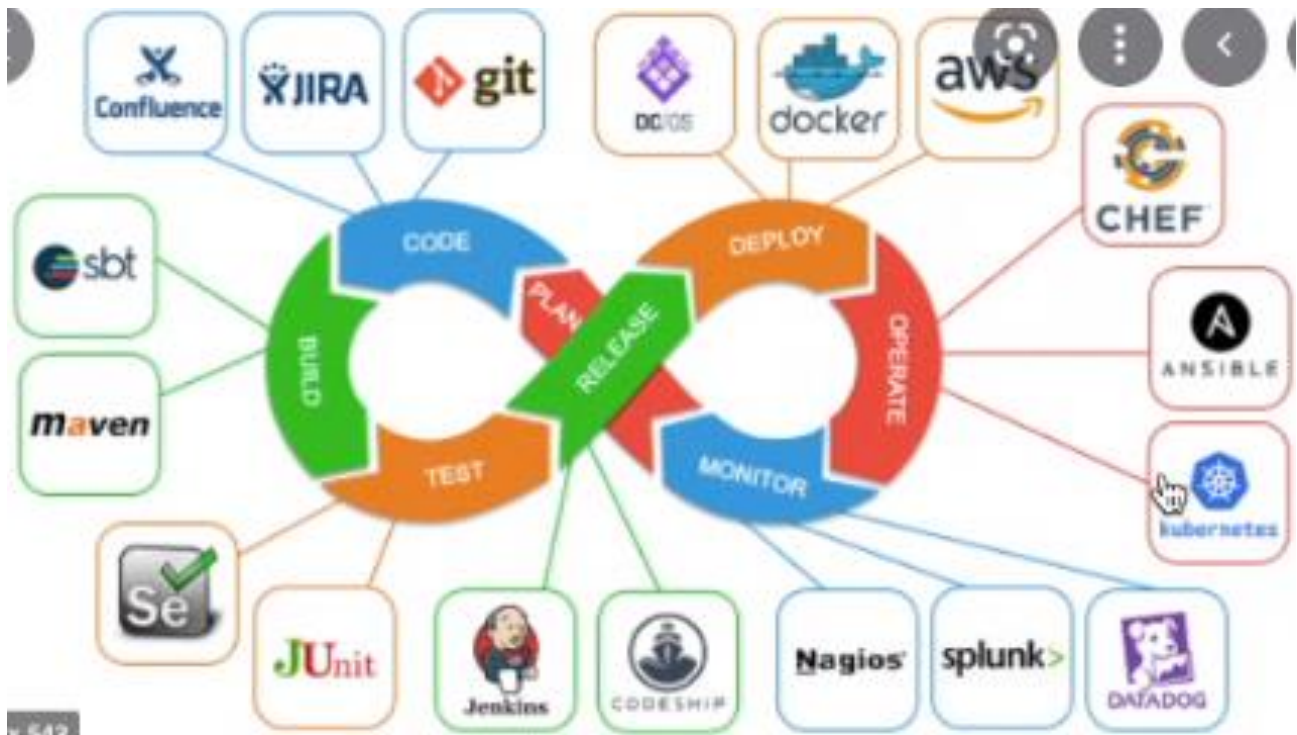




## Docker - Kubernetes

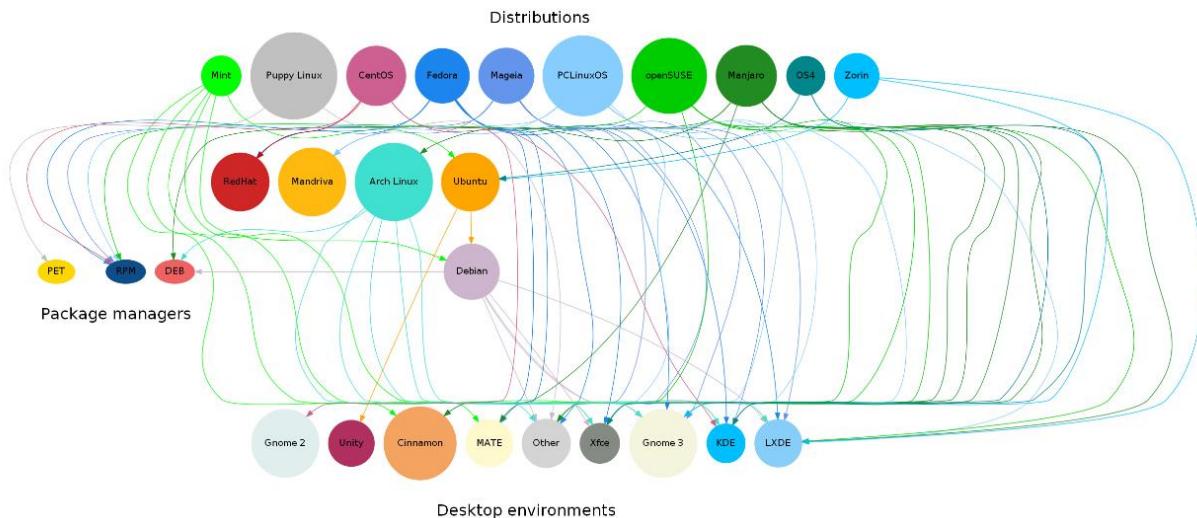
Kubernetes es un administrador de contenedores (dockers), lo que este hace es abrir varios dockers y poner reglas para cuando un contenedor se sobesatura, se cree uno nuevo o demás reglas de administración de recursos.

Desde la perspectiva de DevOps, lo que hace Kubernetes es la automatización la ejecución del código de operación desde un software cliente hacia el servidor, automatizando de esta forma las pruebas unitarias.



# Python - Distribuciones de Linux

**Kernel de Linux:** Es la forma en la que se conecta el CPU con los drivers que controlan los periféricos, de ahí se crearon ciertas variantes y se crearon las distintas distribuciones de Linux que son: Debian, Ubuntu, Fedora, CentOS, AWS, Arch.



## Python - TDD (Test Driven Development) con frameworks

Es una forma en la que se puede generar código con la intención de que se generen menos bugs y se pueda automatizar de mejor manera. Este consiste en que primero se programa una prueba del código y luego se pasa al código.

The screenshot shows an IDE with a project named 'solar'. The left sidebar shows a file tree with 'services' and 'InteractionServiceSpec.scala'. The main editor displays Scala code for 'InteractionServiceSpec'. The code includes tests for 'addAcceptedRecommendationInteraction' and 'addAcceptedVerificationInteraction'. The bottom panel shows a terminal with the output of a Scala test run, including compilation errors and test results.

```
//scalastyle:on
21 "addAcceptedRecommendationInteraction" should {
22   "return the interactionId of the accepted recommendation interaction created" in {
23     val relationshipOpt = Some("led")
24     val connectionId = 1
25     val recommendationId = 1
26     val interactionId = 1
27
28     when(instanceOf[InteractionModel].add(any())) thenReturn(Future(interactionId))
29
30     val result = await {
31       instanceOf[InteractionService].addAcceptedRecommendationInteraction(relationshipOpt, connectionId, recommendationId)
32     }
33
34     result mustBe 1
35   }
36 }
37
38 "addAcceptedVerificationInteraction" should {
39   "return the interactionId of the accepted verification interaction created" in {
```

```
Terminal: solar: heda discovery vader tiamat
at play.sbt.PlayExceptions$CompilationException$.apply(PlayExceptions.scala:36)
at scala.Option.map(Option.scala:236)
at play.sbt.run.PlayReLoad$.anonfun$taskFailureHandler$1(PlayReLoad.scala:48)
at scala.Option.map(Option.scala:236)
at play.sbt.run.PlayReLoad$.taskFailureHandler(PlayReLoad.scala:35)
at play.sbt.run.PlayReLoad$.compileFailure(PlayReLoad.scala:28)
at play.sbt.run.PlayReLoad$.anonfun$compile$3(PlayReLoad.scala:63)
at scala.util.Either$LeftProjection.map(Either.scala:573)
at play.sbt.run.PlayReLoad$.compile(PlayReLoad.scala:63)
```

El test se programa en una clase, para que cuando ya haya programado mi test, la función hecha se someta al test para ver si esta lo pasa o tiene errores. Al proceso de TDD (Test Driven Development) de le llama Red – Green – Refactor:

- **Red:** Error fatal.
- **Green:** Función que ejecutó la clase test de manera exitosa.
- **Refactor:** Mejora de código.

## Pruebas Unitarias y Pruebas de Integración

**Pruebas unitarias:** Cuando se prueba que una función de código única haga su función correctamente.

**Pruebas de integración:** Cuando se prueba que todas las funciones de un código más complejo se ejecuten a la vez y que el programa en general ejecute de forma correcta.

