

# Evaluación Parcial 2

## “Interacción con API, Dockers, Jenkins y Gestión de Claves”

Sigla	Nombre Asignatura	Tiempo Asignado	% Ponderación
DRY7122	Programación y Redes Virtualizadas (SDN-NFV)	160 minutos	33

### 01. Agente evaluativo

<input checked="" type="checkbox"/>	Heteroevaluación	<input type="checkbox"/>	Coevaluación	<input type="checkbox"/>	Autoevaluación
-------------------------------------	------------------	--------------------------	--------------	--------------------------	----------------

### 02. Tabla de Especificaciones

Resultado de Aprendizaje	Indicador de Logro (IL)	Indicador de Evaluación (IE)*	Ponderación Indicador Logro	Ponderación Indicador de Evaluación
Emplea soluciones de conectividad programable para mantener actualizadas las capacidades de la red según las necesidades de la organización.	2.1 Identifica las diferentes interfaces de programación de aplicaciones (API), según los requerimientos de la organización.	1. Obtiene token de red programable, logrando que se almace en ambiente en Postman.	15%	5%
		2. Utiliza API que permita consultar por red física de red programable.		5%
		3. Utiliza API que permita crear usuario con contraseña requerida, permitiendo ingresar a dashboard de controlador.		5%
	2.2 Verifica las soluciones de conectividad programables según el tipo de protocolo para la infraestructura de la organización.	4. Demuestra con script desarrollado la distancia en kilómetros entre Santiago y Ovalle, además de solicitar Ciudad de Origen y Ciudad de Destino.	25%	5%

		5. Demuestra con script desarrollado la duración el viaje en horas, minutos y segundos, además del combustible requerido en litros.		5%
		6. Demuestra con script desarrollado que todos los valores utilizan dos decimales, además de imprimir la narrativa del viaje.		5%
		7. Demuestra con script desarrollado la salida del programa con la letra q, y subir el script respectivo a repositorio público en GitHub.		10%
	2.3 Emplea técnicas para la implementación integral y continua de procesos automatizados según los requerimientos de la empresa.	8. Ingresar al directorio solicitado y con comando respectivo poder almacenar todos los archivos en dicha ubicación.	20%	10%
		9. Logra cambiar el puerto de la aplicación de muestra al puerto 9999 y la instalación del Docker Jenkins.		10%
	2.4 Reconoce el concepto de alta disponibilidad en la implementación de aplicaciones de red en función de una arquitectura TI, con la finalidad de entregar seguridad a la organización.	10. Crea un trabajo en Jenkins con nombre requerido, además de asociarlo al repositorio GitHub respectivo, además que el trabajo permita ejecutar y compilar script para permitir funcionamiento de página web de muestra.	20%	10%
		11. Valida que la página web se encuentre disponible con el puerto requerido, y que los archivos se encuentren respaldados en GitHub.		10%
	2.5 Diseña arquitectura automatizadas con equipos físicos y/o virtuales para la infraestructura de la organización.	12. Crea script con nombre requerido y código importa los paquetes requeridos para la gestión de claves.	20%	5%
		13. Agrega código de Flask que permite al archivo crear la primera fase del contenido utilizando el puerto 5000, además de realizar la validación del sitio con el comando curl.		5%
		14. Agrega líneas de códigos dentro del script que permita almacenar nombres y contraseñas en texto plano, verificar nuevas contraseñas y en cada intento de		5%

		sesión leer los parámetros de una solicitud HTTP y verificar la cuenta.		
		15. Valida a través del comando curl dos usuarios con sus contraseñas y luego a través de aplicación DB Browser SQLite, revisa que dicha información se encuentre almacenada.		5%
Total			100%	100%

### 03. Instrucciones para el/la estudiante

Esta es una evaluación que corresponde a una evaluación práctica sin presentación y tiene un 33% de ponderación sobre la nota final de la asignatura.

El **tiempo** para desarrollar esta evaluación es de **160 minutos** y se realiza de manera **individual** en **laboratorio respectivo**. Para el desarrollo de esta evaluación requiere el uso de máquina virtual DEVASC.

#### **Contexto:**

La empresa “DRY7122 S.A” se encuentra en proceso de evaluación tecnológica, donde buscará migrar equipos de conectividad antiguos por versiones más actualizadas. Por otro lado, se han enterado de que muchos de estos cambios involucran la automatización y la integración de la programación, por lo cual antes de hacer este cambio necesitan nivelar el conocimiento de los administradores de red en esta temática para luego proceder con el cambio respectivo.

#### **Instrucciones Generales:**

1. Esta evaluación puede realizarse de manera individual.
2. Utilizará máquina virtual DEVASC, los cuales deberán estar importados en VirtualBox. En caso contrario lo puede obtener del siguiente link: [bit.ly/3KqL89E](https://bit.ly/3KqL89E)
3. En un documento Word deben hacer capturas de pantallas, el cual debe aparecer fecha y hora, donde debe abordar el requerimiento planteado, además de una breve descripción de lo realizado ahí.
4. Es importante que guarden sus avances cada 5 minutos.
5. Una vez finalizada la evaluación deberán adjuntar el documento realizado en formato .PDF el cual debe ser adjuntado a través de la actividad AVA respectiva.

**Requerimientos:****A. Consumo de API en Red Programable**

Utilizando archivo Cisco Packet Tracer en este link: <http://bit.ly/3AgInnH> deberá realizar consumo de API a una controladora utilizando Postman, donde deberá lograr lo siguiente:

- Obtener token de la red programable, el cual deberá quedar almacenado en un ambiente llamado **Prueba2**
- Utilizar API que permita consultar por la red física.
- Utilizar API que permita crear usuario **Prueba2** y clave **SDN.2023**. Deberá demostrar el acceso al dashboard de la controladora.z

**B. Consumo de API Pública**

Utilizando el sitio de MapQuest y el token generado en laboratorio respectivo, deberá crear un código en VisualStudio Code de la máquina virtual DEVASC, donde el programa realice lo siguiente:

- Medir la distancia en kilómetros entre Santiago y Ovalle.
- Solicitar “Ciudad de Origen” y “Ciudad de Destino”
- Mostrar la duración del viaje en horas, minutos y segundos
- Mostrar el combustible requerido para el viaje representado en litros
- Todos los valores deben utilizar dos decimales.
- Debe imprimir la narrativa del viaje.
- Agregar una salida del programa con la letra “q”
- Subir el script creado en un repositorio público en GitHub con el nombre de “**Evaluación N°2 DRY7122**” y en la descripción los nombres de los integrantes. Debe incluir un commit con el nombre de “**Consumo de API Pública**”

DEVASC-LABVM [Corriendo] - Oracle VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

graphhopper\_parse-json\_1.py - Evaluacion2 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

```

graphhopper_parse-json_1.py X
graphhopper_parse-json_1.py > geocodificar > datos
1 import requests
2 import urllib.parse
3
4
5 # Configuración de la API
6 url_ruta = "https://graphhopper.com/api/1/route?"
7 clave_api = "ab404e69-2ec4-4ab9-9f0d-d32565d8553b" # Reemplaza con tu clave real
8
9
10 def geocodificar(ubicacion, clave):
11     """Convierte nombres de lugares a coordenadas geográficas"""
12     while ubicacion == "":
13         ubicacion = input("Por favor ingrese la ubicación nuevamente: ")
14
15     url_geocodificacion = "https://graphhopper.com/api/1/geocode?"
16     parametros = {"q": ubicacion, "limit": "1", "key": clave}
17     url = url_geocodificacion + urllib.parse.urlencode(parametros)
18
19     try:
20
21         respuesta = requests.get(url)
22         datos = respuesta.json()
23
24         if respuesta.status_code == 200 and datos.get("hits"):
25             latitud = datos["hits"][0]["point"]["lat"]
26             longitud = datos["hits"][0]["point"]["lng"]
27             nombre = datos["hits"][0]["name"]
28
29             pais = datos["hits"][0].get("country", "")
30             region = datos["hits"][0].get("state", "")
31
32             if region and pais:
33                 ubicacion_formateada = f"{nombre}, {region}, {pais}"
34             elif pais:
35                 ubicacion_formateada = f"{nombre}, {pais}"
36             else:
37                 ubicacion_formateada = nombre
38
39             return respuesta.status_code, latitud, longitud, ubicacion_formateada
40         else:
41             print(f"\nError: No se encontró la ubicación '{ubicacion}'")
42             return respuesta.status_code, None, None, ubicacion
43     except Exception as e:
44         print(f"\nError de conexión: {str(e)}")
45         return 500, None, None, ubicacion
46
47
48 def calcular_combustible(distancia km):
49     """Calcula el combustible necesario (12 km por litro)"""
50     return distancia_km / 12.0
51

```

Ln 22, Col 9 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit

Menu graphhopper\_parse-js... [Evaluacion2]

CTRL DERECHA

DEVASC-LABVM [Corriendo] - Oracle VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

graphhopper\_parse-json\_1.py - Evaluacion2 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

graphhopper\_parse-json\_1.py x

graphhopper\_parse-json\_1.py > geocodificar > datos

```

53 def formatear_duracion(segundos):
54     """Convierte segundos a horas, minutos y segundos"""
55     horas = int(segundos // 3600)
56     minutos = int((segundos % 3600) // 60)
57     segundos = int(segundos % 60)
58     return horas, minutos, segundos
59
60
61 def mostrar_resultados(origen, destino, distancia, duracion, combustible):
62     """Muestra los resultados del cálculo de ruta"""
63     print("\n" + "="*60)
64     print(" RESUMEN DEL VIAJE".center(60))
65     print("="*60)
66     print(f" • Origen: {origen}")
67     print(f" • Destino: {destino}")
68     print(f" ✓ Distancia total: {distancia:.2f} km")
69
70     horas, minutos, segundos = formatear_duracion(duracion)
71     print(f" ⌚ Duración estimada: {horas:02d}h {minutos:02d}m {segundos:02d}s")
72
73     print(f" 🛢 Combustible necesario: {combustible:.2f} litros (a 12 km/l)")
74     print("="*60)
75
76
77 def mostrar_instrucciones(ruta):
78     """Muestra las instrucciones de navegación paso a paso"""
79     print("\n🔥 INSTRUCCIONES DE RUTA:")
80     for paso in ruta["paths"][0]["instructions"]:
81         distancia_km = paso["distance"] / 1000
82         print(f"- {paso['text']} ({distancia_km:.2f} km)")
83     print("="*60)
84
85
86 def main():
87     print("\n" + "="*60)
88     print(" 🗺 CALCULADOR DE VIAJES".center(60))
89     print("="*60)
90     print("Este programa calcula:")
91     print("- Distancia entre ciudades")
92     print("- Duración del viaje")
93     print("- Combustible requerido")
94     print("\nInstrucciones:")
95     print("1. Ingrese ciudad de origen y destino")
96     print("2. Presione 'q' para salir en cualquier momento")
97     print("="*60)
98
99
100     while True:
101         # Entrada de origen
102         origen_input = input("\n🏠 Ciudad de Origen (o 'q' para salir): ").strip()
103         if origen_input.lower() == 'q':

```

Ln 22, Col 9 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit

Menu graphhopper\_parse-js... [Evaluacion2]

CTRL DERECHA

DEVASC-LABVM [Corriendo] - Oracle VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

graphhopper\_parse-json\_1.py - Evaluacion2 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

graphhopper\_parse-json\_1.py x

graphhopper\_parse-json\_1.py > geocodificar > [e] datos

```

100 while True:
101     # Entrada de origen
102     origen_input = input("\n📍 Ciudad de Origen (o 'q' para salir): ").strip()
103     if origen_input.lower() == 'q':
104         break
105
106     # Geocodificación de origen
107     estado_origen, lat_origen, lon_origen, origen_nombre = geocodificar(origen_input, clave_api)
108     if estado_origen != 200:
109         print("No se pudo procesar el origen. Intente nuevamente.")
110         continue
111
112
113     # Entrada de destino
114     destino_input = input("\n📍 Ciudad de Destino (o 'q' para salir): ").strip()
115     if destino_input.lower() == 'q':
116         break
117
118     # Geocodificación de destino
119     estado_destino, lat_destino, lon_destino, destino_nombre = geocodificar(destino_input, clave_api)
120     if estado_destino != 200:
121         print("No se pudo procesar el destino. Intente nuevamente.")
122         continue
123
124
125     # Construcción de URL para la API de rutas
126     parametros_ruta = {
127         "key": clave_api,
128         "vehicle": "car",
129         "point": [f"{lat_origen},{lon_origen}", f"{lat_destino},{lon_destino}"]
130     }
131     url = url_ruta + urllib.parse.urlencode(parametros_ruta, doseq=True)
132
133
134     try:
135         respuesta = requests.get(url)
136         datos_ruta = respuesta.json()
137
138         if respuesta.status_code == 200:
139             # Cálculos principales
140             distancia_km = datos_ruta["paths"][0]["distance"] / 1000
141             duracion_seg = datos_ruta["paths"][0]["time"] / 1000
142             combustible = calcular_combustible(distancia_km)
143
144             # Mostrar resultados
145             mostrar_resultados(origen_nombre, destino_nombre, distancia_km, duracion_seg, combustible)
146             mostrar_instrucciones(datos_ruta)
147         else:
148             print(f"\nError al calcular la ruta: {datos_ruta.get('message', 'Error desconocido')}")
149     except Exception as e:
150         print(f"\nError de conexión: {str(e)}")

```

Ln 22, Col 9 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit

Menu graphhopper\_parse-js... [Evaluacion2]

CTRL DERECHA



```

graphhopper_parse-json_1.py - Evaluacion2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help

graphhopper_parse-json_1.py x
graphhopper_parse-json_1.py > geocodificar > datos
132
133
134     try:
135         respuesta = requests.get(url)
136         datos_ruta = respuesta.json()
137
138         if respuesta.status_code == 200:
139             # Cálculos principales
140             distancia_km = datos_ruta["paths"][0]["distance"] / 1000
141             duracion_seg = datos_ruta["paths"][0]["time"] / 1000
142             combustible = calcular_combustible(distancia_km)
143
144             # Mostrar resultados
145             mostrar_resultados(origen_nombre, destino_nombre, distancia_km, duracion_seg, combustible)
146             mostrar_instrucciones(datos_ruta)
147         else:
148             print(f"\nError al calcular la ruta: {datos_ruta.get('message', 'Error desconocido')}")
149     except Exception as e:
150         print(f"\nError de conexión: {str(e)}")
151
152     # Consultar por otra ruta
153     otra_ruta = input("\n¿Desea calcular otra ruta? (s/n): ").lower()
154     if otra_ruta != 's':
155         break
156
157     print("\n¡Gracias por usar el calculador de viajes! 🚗")
158
159
160
161 if __name__ == "__main__":
162     try:
163         main()
164     except KeyboardInterrupt:
165         print("\nPrograma interrumpido por el usuario")
166     except Exception as e:
167         print(f"\nError inesperado: {str(e)}")
168
169
170

```

```

DEVASC-LABVM [Corriendo] - Oracle VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda

graphhopper_parse-json_1.py - Evaluacion2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help

graphhopper_parse-json_1.py x graphhopper_parse-json_2.py
graphhopper_parse-json_1.py > ...
132

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Debug Console

=====
CALCULADOR DE VIAJES
=====
Este programa calcula:
- Distancia entre ciudades
- Duración del viaje
- Combustible requerido

Instrucciones:
1. Ingrese ciudad de origen y destino
2. Presione 'q' para salir en cualquier momento
=====

🏠 Ciudad de Origen (o 'q' para salir): ovalle
🏠 Ciudad de Destino (o 'q' para salir): santiago

=====
📋 RESUMEN DEL VIAJE
=====
• Origen: Ovalle, Región de Coquimbo, Chile
• Destino: Santiago, Región Metropolitana de Santiago, Chile
📏 Distancia total: 404.38 km
🕒 Duración estimada: 03h 50m 56s
🛢️ Combustible necesario: 33.70 litros (a 12 km/l)
=====

📌 INSTRUCCIONES DE RUTA:
- Continue onto Libertad (0.47 km)
- Turn left onto Tamaya (0.27 km)
- Turn left onto Independencia (0.01 km)
- Turn right onto Avenida La Chimba (0.78 km)
- Turn right onto Costanera del Río Limarí (2.12 km)
- Turn left onto Gobernadora Laura Pizarro (0.84 km)
- Turn right onto Acceso a La Chimba (0.06 km)
- Turn left onto Camino Antiguo de La Chimba (2.79 km)
- Turn right onto Ruta 45 (26.86 km)
- Continue and take 5 toward Los Vilos (241.89 km)
- Keep left onto Ruta 5 Norte (101.82 km)
- Keep left onto Ruta 5 Norte (23.39 km)
- Keep right (1.71 km)
- Keep right and drive toward Mercado Central (0.50 km)
- Keep left (0.07 km)
- Turn left onto Ismael Valdés Vergara (0.25 km)
- Turn right onto San Antonio (0.36 km)
- Turn right onto Monjitas (0.19 km)
- Arrive at destination (0.00 km)
=====

¿Desea calcular otra ruta? (s/n): n

¡Gracias por usar el calculador de viajes! 🎉
devasc@labvm:~/Labs/devnet-src/Evaluacion2$
  
```



### Codigo

```

import requests
import urllib.parse

# Configuración de la API
url_ruta = "https://graphhopper.com/api/1/route?"
clave_api = "API" # Reemplaza con tu clave real

def geocodificar(ubicacion, clave):
    """Convierte nombres de lugares a coordenadas geográficas"""
    while ubicacion == "":
        ubicacion = input("Por favor ingrese la ubicación nuevamente: ")

    url_geocodificacion = "https://graphhopper.com/api/1/geocode?"
    parametros = {"q": ubicacion, "limit": "1", "key": clave}
    url = url_geocodificacion + urllib.parse.urlencode(parametros)

    try:
        respuesta = requests.get(url)
        datos = respuesta.json()

        if respuesta.status_code == 200 and datos.get("hits"):
            latitud = datos["hits"][0]["point"]["lat"]
            longitud = datos["hits"][0]["point"]["lng"]
            nombre = datos["hits"][0]["name"]

```

```

    pais = datos["hits"][0].get("country", "")
    region = datos["hits"][0].get("state", "")

    if region and pais:
        ubicacion_formateada = f"{nombre}, {region}, {pais}"
    elif pais:
        ubicacion_formateada = f"{nombre}, {pais}"
    else:
        ubicacion_formateada = nombre

    return respuesta.status_code, longitud, ubicacion_formateada
else:
    print(f"\nError: No se encontró la ubicación '{ubicacion}'")
    return respuesta.status_code, None, None, ubicacion
except Exception as e:
    print(f"\nError de conexión: {str(e)}")
    return 500, None, None, ubicacion


def calcular_combustible(distancia_km):
    """Calcula el combustible necesario (12 km por litro)"""
    return distancia_km / 12.0


def formatear_duracion(segundos):
    """Convierte segundos a horas, minutos y segundos"""
    horas = int(segundos // 3600)
    minutos = int((segundos % 3600) // 60)
    segundos = int(segundos % 60)
    return horas, minutos, segundos


def mostrar_resultados(origen, destino, distancia, duracion, combustible):
    """Muestra los resultados del cálculo de ruta"""
    print("\n" + "="*60)
    print("📋 RESUMEN DEL VIAJE".center(60))
    print("="*60)
    print(f"♦ Origen: {origen}")
    print(f"♦ Destino: {destino}")


```


```


print(f" Distancia total: {distancia:.2f} km")

horas, minutos, segundos = formatear_duracion(duracion)
print(f" Duración estimada: {horas:02d}h {minutos:02d}m {segundos:02d}s")

print(f" Combustible necesario: {combustible:.2f} litros (a 12 km/l)")
print("="*60)

def mostrar_instrucciones(ruta):
    """Muestra las instrucciones de navegación paso a paso"""
    print("\n  INSTRUCCIONES DE RUTA:")
    for paso in ruta["paths"][0]["instructions"]:
        distancia_km = paso["distance"] / 1000
        print(f"- {paso['text']} ({distancia_km:.2f} km)")
    print("="*60)

def main():
    print("\n" + "="*60)
    print("  CALCULADOR DE VIAJES".center(60))
    print("="*60)
    print("Este programa calcula:")
    print("- Distancia entre ciudades")
    print("- Duración del viaje")
    print("- Combustible requerido")
    print("\nInstrucciones:")
    print("1. Ingrese ciudad de origen y destino")
    print("2. Presione 'q' para salir en cualquier momento")
    print("="*60)

    while True:
        # Entrada de origen
        origen_input = input("\n  Ciudad de Origen (o 'q' para salir): ").strip()
        if origen_input.lower() == 'q':
            break

        # Geocodificación de origen
        estado_origen, lat_origen, lon_origen, origen_nombre = geocodificar(origen_input, clave_api)

```

```

if estado_origen != 200:
    print("No se pudo procesar el origen. Intente nuevamente.")
    continue

# Entrada de destino
destino_input = input("📍 Ciudad de Destino (o 'q' para salir): ").strip()
if destino_input.lower() == 'q':
    break

# Geocodificación de destino
estado_destino, lat_destino, lon_destino, destino_nombre = geocodificar(destino_input, clave_api)
if estado_destino != 200:
    print("No se pudo procesar el destino. Intente nuevamente.")
    continue

# Construcción de URL para la API de rutas
parametros_ruta = {
    "key": clave_api,
    "vehicle": "car",
    "point": [f"{lat_origen},{lon_origen}", f"{lat_destino},{lon_destino}"]
}
url = url_ruta + urllib.parse.urlencode(parametros_ruta, doseq=True)

try:
    respuesta = requests.get(url)
    datos_ruta = respuesta.json()

    if respuesta.status_code == 200:
        # Cálculos principales
        distancia_km = datos_ruta["paths"][0]["distance"] / 1000
        duracion_seg = datos_ruta["paths"][0]["time"] / 1000
        combustible = calcular_combustible(distancia_km)

        # Mostrar resultados
        mostrar_resultados(origen_nombre, destino_nombre, distancia_km, duracion_seg, combustible)
        mostrar_instrucciones(datos_ruta)
    else:

```

```

        print(f"\nError al calcular la ruta: {datos_ruta.get('message', 'Error desconocido')}")
    except Exception as e:
        print(f"\nError de conexión: {str(e)}")

    # Consultar por otra ruta
    otra_ruta = input("\n¿Desea calcular otra ruta? (s/n): ").lower()
    if otra_ruta != 's':
        break

    print("\n¡Gracias por usar el calculador de viajes! 💎")

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print("\nPrograma interrumpido por el usuario")
    except Exception as e:
        print(f"\nError inesperado: {str(e)}")

```

---