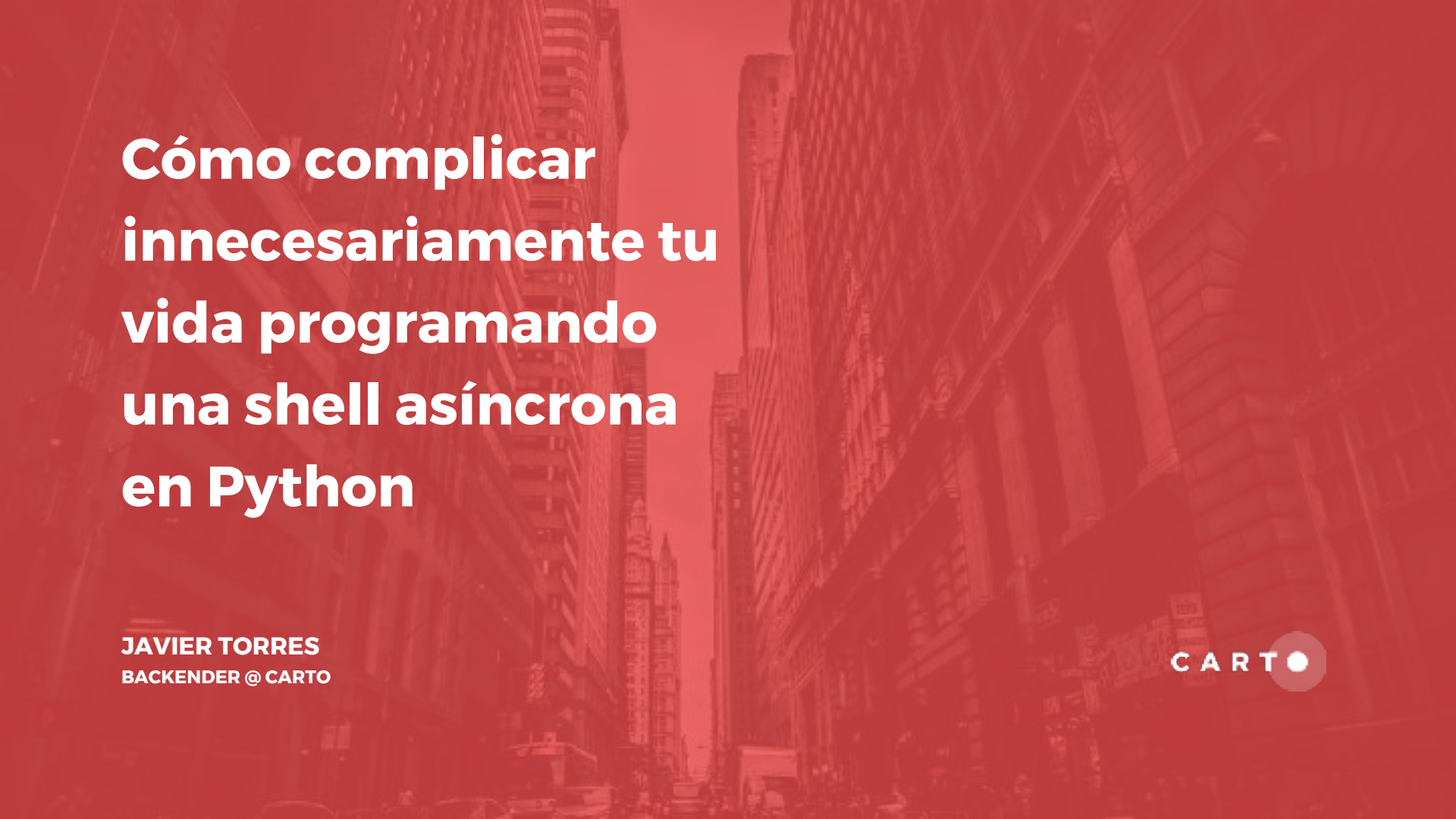


Cómo complicar innecesariamente tu vida programando una shell asíncrona en Python

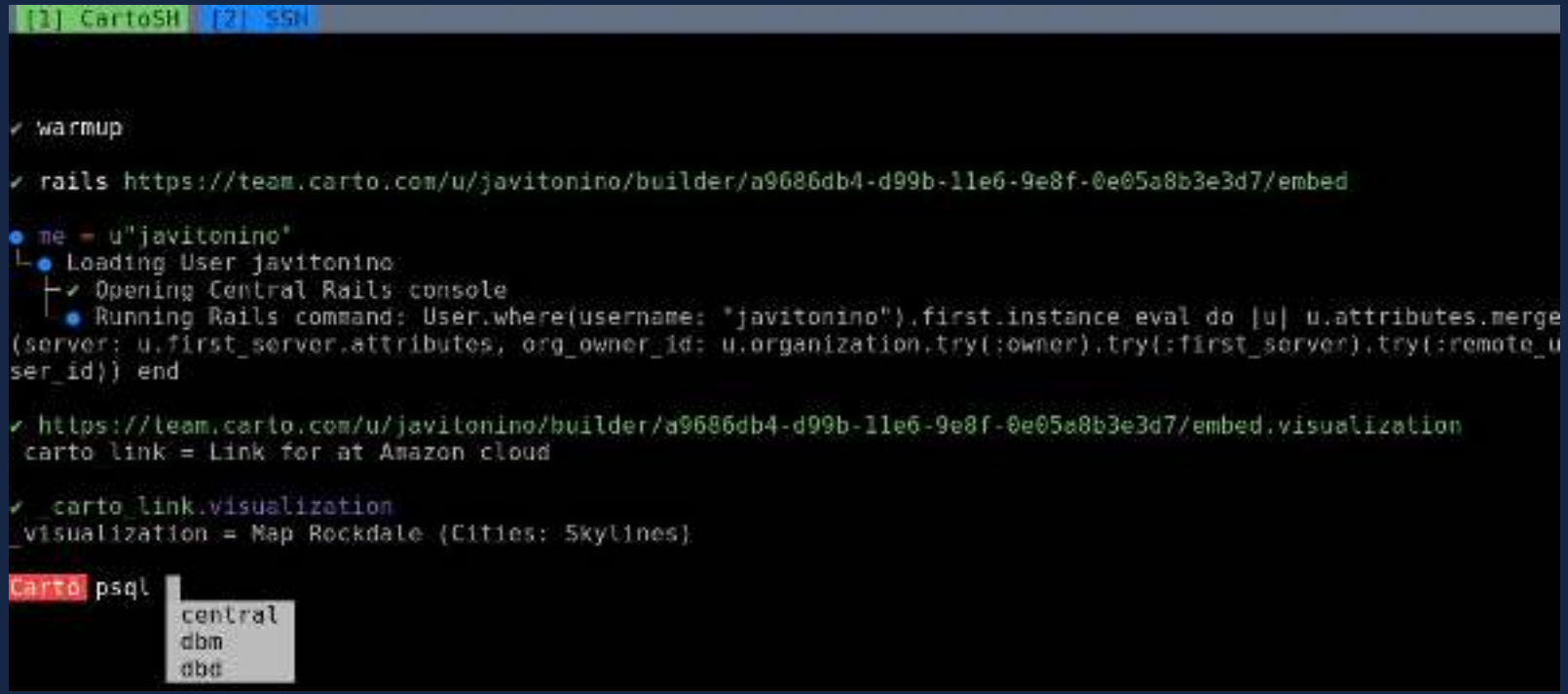
JAVIER TORRES
BACKENDER @ CARTO

CARTO

WHY?

WHY NOT?

DEMO TIME


[1] CartoSH [2] SSH
✓ warmup
✓ rails https://team.carto.com/u/javitonino/builder/a9686db4-d99b-11e6-9e8f-0e05a8b3e3d7/embed
● me = u"javitonino"
└● Loading User javitonino
├✓ Opening Central Rails console
└● Running Rails command: User.where(username: "javitonino").first.instance_eval do |u| u.attributes.merge
(server: u.first_server.attributes, org_owner_id: u.organization.try(:owner).try(:first_server).try(:remote_u
ser_id)) end
✓ https://team.carto.com/u/javitonino/builder/a9686db4-d99b-11e6-9e8f-0e05a8b3e3d7/embed.visualization
carto_link = Link for at Amazon cloud
✓ carto_link.visualization
_visualization = Map Rockdale (Cities: Skylines)
Carto psql
central
dbm
dbd

LAS PIEZAS DEL PUZZLE

PROMPT_TOOLKIT

ASYNCIO

PLY

META PROGRAMACIÓN

PROMPT TOOLKIT

“Press any key to continue or any other key to quit.” - Unknown

LAYOUT

Los contenedores dividen la pantalla en partes (horizontal/vertical) y ajustan el tamaño de su contenido.

Los componentes muestran distinto contenido, por lo general, tokens.

Hay un layout para prompts por defecto. Tosh lo utiliza, añadiendo la barra de tabs.

```
class MainWindow(HSplit):
    def __init__(self, tosh):
        self._tosh = tosh
        self._tabs = [ToshTab(tosh)]
        self._active_tab = 0
        layout = [
            Window(
                TokenListControl(
                    self._get_tabs_tokens,
                    default_char=Char(' ', Token.Tabs),
                ),
                height=D(max=1)
            ),
            self._tabs[0].layout
```

TOKENS

Un token se compone de un texto acompañado de una tag que indica el estilo.

Un estilo mapea tags a estilos, se adaptan automáticamente a las capacidades del terminal.

```
prompt = [
    (Token.Prompt.Text, 'tosh'),
    (Token.Prompt, '▌')
]

style = {
    Token.Prompt:                '#f24440',
    Token.Prompt.Text:           '#fff bg:#f24440'
}
```

ENTRADA

Se pueden definir key bindings con un decorador. También para ratón.

prompt_toolkit ejecutará la función asociada cuando se detecte la pulsación de teclas.

Pueden usarse filtros dependiendo del estado de la aplicación.

```
prompt =
 ConditionalRegistry(filter=Condition(_prompt_tab))
interactive =
 ConditionalRegistry(filter=Condition(_interactive_tab))

@prompt.add_binding(Keys.ControlW)
@interactive.add_binding(Keys.ControlB, 'w')
def _close_tab(event):
    tosh.window.active_tab().close()

@global.add_binding(Keys.ControlB, Keys.Left)
def _prev_tab(_):
    tosh.window.prev_tab()
```

ASYNCIO

“In the jungle
The mighty jungle
The lion sleeps tonight.

Async, await, async,
await, async, await...”

ASYNCIO

Permite ejecutar varias tareas en paralelo sin usar hilos.

Integración directa con prompt_toolkit.

```
self._cli = CommandLineInterface(
    application=application,
    eventloop=create_asyncio_eventloop(),
    output=create_output(true_color=True)
)

asyncio.get_event_loop().run_until_complete(
     self._cli.run_async()
)
```

ASYNCSSH

Implementa el protocolo SSH sobre asyncio.

Una conexión, varias sesiones, varios usuarios. Mutex para evitar conflictos.

Código de pymux para convertir de prompt_toolkit a vt100.

```
class _SSHSwitchableSession(asyncssh.SSHClientSession):
    async def __aenter__(self):
        await self._lock.acquire()
        return self._handler

    async def __aexit__(self, *_):
        self._lock.release()


async with session as handler:
    await self.sub(handler.connect_db, db_name)
    tab = create_interactive_tab(self._tosh, session)
```

PLY

“Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.
” - Jamie Zawinski

LEXER + PARSER

Usa docstrings para reglas de construcción.

El lexer identifica tokens, por ejemplo: una cadena de texto o un entero.

El parser identifica, en este caso, comandos. Indica que tokens pueden seguir a cuáles.

```
def t_INTEGER(self, t):
    r'\d+'
    t.value = Integer.load_task(self._tosh, t.value)
    return t


def p_assignment_statement(self, t):
    """
    statement : VARIABLE '=' expression
              | VARIABLE '=' command
              | BARE_WORD '=' expression
              | BARE_WORD '=' command
    """
    t[0] = AssignmentStatement(
                self._tosh, t[1].bare_word, t[3])
```

AUTO COMPLETE

Funcionalidad de prompt_toolkit, implementada a mano.

Es difícil usar ply puesto que se trata de autocompletar comandos parciales (que fallan el parseo).

```
if tokens[-1].type == '=':
    # After an equal, return expressions
    for i in self._add_space(self._expressions(), fix):
        yield i
elif tokens[-1].type == '(':
    # After a parens, return commands
    for i in self._add_space(self._commands(), fix):
        yield i
elif tokens[-1].type == '.':
    # After a dot, return attributes
    var = tokens[-2].value.return_type
    for c in self._add_space(var.attributes.keys()):
        yield c
```

META PROGRAMACIÓN

“It´s okay to figure out murder mysteries, but you should not need to figure out code. You should be able to read it.
” - Steve McConnell

VARIABLES

Usan metaclases para registrar automáticamente las subclases, con sólo importarlas.

`__subclasses__` sólo devuelve las subclases cargadas en un módulo.

```
class _VariableMeta(type):
    by_prefix = {}

    def __init__(self, name, bases, attrs):
        super().__init__(name, bases, attrs)

        # Register this variable
        self.by_class_name[name] = self
        if hasattr(self, 'prefix'):
            self.by_prefix[self.prefix] = self


class Variable(metaclass=_VariableMeta):
    pass

Variable.by_prefix[prefix]
```

TASKS

Decorador para marcar cualquier corutina como tarea, sin necesidad de crear una clase.

Útil para métodos parte de comandos más complejos.

Instanciación manual para crear referencia circular.


```
def task(title):
    def task_decorator(func):
        def task_method(*args, **kwargs):
            tosh = kwargs.pop('_tosh')
            _task = CoroutineTask.__new__(CoroutineTask)
            _task.__init__(tosh, func(*args, **kwargs,
                                      task=_task))
            return _task
        return task_method
    return task_decorator


@task("Loading {pos[0].class_name} {pos[1]}")
async def load(cls, argument, *, task):
    pass
```

IT COMPILES!
SHIP IT!

1. TOSH

El núcleo de la aplicación. Disponible en:
https://github.com/javitonino/tosh

2. TOSHQL

Un paquete de ejemplo con algunos comandos para conectarse a bases de datos remotas.
https://github.com/javitonino/toshql


¿PREGUNTAS?
We are hiring: https://carto.com/careers/