# Deep Clojure/Python Integration

### Version Info

build passing build passing

## New Versions Are HOT! Huge New Features! You Can't Afford To Miss Out!

- API Documentation
- Java API - you can use libpython-clj from java - no Clojure required. The class is included with the jar so just put the jar on the classpath and then `import libpython_clj2.java_api;` will work. Be sure to carefully read the namespace doc as, due to performance considerations, not all methods are protected via automatic GIL management. Note this integration includes support for extremely efficient data copies to numpy objects and callbacks from python to java.
- make-fastcallable so if you calling a small function repeatedly you can now call it about twice as fast. A better optimization is to call a function once with numpy array arguments but unfortunately not all use cases are amenable to this pathway. So we did what we can.
- JDK-17 and Mac M-1 support. To use libpython-clj2 with jdk-17 you need to enable the foreign module - see deps.edn for a working alias.
- You can now Embed Clojure in Python - you can launch a Clojure REPL from a Python host process.
- **32 bit support**!!
- 20-30% better performance.
- Please avoid deprecated versions such as `[cnuernber/libpython-clj "1.36"]` (***note name change***).
- This library, which has received the efforts of many excellent people, is built mainly upon cnuernber/dtype-next and the JNA library.
- Static code generation - generate clojure namespaces wrapping python modules that are safe to use with AOT and load much faster than analogous `require-python` calls. These namespace will not automatically initialize the python subsystem – initialize! must be called first (or a nice exception is throw).

### libpython-clj features

- Bridge between JVM objects and Python objects easily; use Python in your Java and use some Java in your Python.
- Python objects are linked to the JVM GC such that when they are no longer reachable from the JVM their references are released. Scope based resource contexts are also available.

- Finding the python libraries is done dynamically allowing one system to run on multiple versions of python.
- REPL oriented design means fast, smooth, iterative development.
- Carin Meier has written excellent posts on plotting and advanced text generation. She also has some great examples.

## Vision

We aim to integrate Python into Clojure at a deep level. This means that we want to be able to load/use python modules almost as if they were Clojure namespaces. We also want to be able to use Clojure to extend Python objects. I gave a talk at Clojure Conj 2019 that outlines more of what is going on.

This code is a concrete example that generates an embedding for faces:

```clojure
(ns facial-rec.face-feature
  (:require [libpython-clj2.require :refer [require-python]]
            [libpython-clj2.python :refer [py. py.. py.-] :as py]
            [tech.v3.datatype :as dtype]))



(require-python 'mxnet
                '(mxnet ndarray module io model))
(require-python 'cv2)
(require-python '[numpy :as np])


(defn load-model
  [& {:keys [model-path checkpoint]
      :or {model-path "models/recognition/model"
           checkpoint 0}}]
  (let [[sym arg-params aux-params] (mxnet.model/load_checkpoint model-
     path checkpoint)
        all-layers (py. sym get_internals)
        target-layer (py/get-item all-layers "fc1_output")
        model (mxnet.module/Module :symbol target-layer
                                   :context (mxnet/cpu)
                                   :label_names nil)]
    (py. model bind :data_shapes [["data" [1 3 112 112]]])
    (py. model set_params arg-params aux-params)
    model))

(defonce model (load-model))


(defn face->feature
  [img-path]
  (py/with-gil-stack-rc-context
    (if-let [new-img (cv2/imread img-path)]
      (let [new-img (cv2/cvtColor new-img cv2/COLOR_BGR2RGB)
            new-img (np/transpose new-img [2 0 1])
            input-blob (np/expand_dims new-img :axis 0)
            data (mxnet.ndarray/array input-blob)
            batch (mxnet.io/DataBatch :data [data])]
        (py. model forward batch :is_train false)
        (-> (py. model get_outputs)
            first
            (py. asnumpy)
            (#(dtype/make-container :java-array :float32 %))))
      (throw (Exception. (format "Failed to load img: %s" img-path)))))
   )
```

## Usage

### Config namespace

```clojure
(ns my-py-clj.config
  (:require [libpython-clj2.python :as py]))

;; When you use conda, it should look like this.
(py/initialize! :python-executable "/opt/anaconda3/envs/my_env/bin/python3.7"
                :library-path "/opt/anaconda3/envs/my_env/lib/libpython3.7m.dylib")
```

### Update project.clj

```clojure
{...
  ;; This namespace going to run when the REPL is up.
  :repl-options {:init-ns my-py-clj.config}
...}
```

```clojure
user> (require '[libpython-clj2.require :refer [require-python]])
...logging info....
nil
user> (require-python '[numpy :as np])
nil
user> (def test-ary (np/array [[1 2][3 4]]))
#'user/test-ary
user> test-ary
[[1 2]
 [3 4]]
```

We have a document on all the features but beginning usage is pretty simple. Import your modules, use the things from Clojure. We have put effort into making sure things like sequences and ranges transfer between the two languages.

**Environments** One very complimentary aspect of Python with respect to Clojure is its integration with cutting edge native libraries. Our support isn't perfect so some understanding of the mechanism is important to diagnose errors and issues.

Current, we launch the python3 executable and print out various different bits of configuration as json. We parse the json and use the output to attempt to find the `libpython3.Xm.so` shared library so for example if we are loading python 3.6 we look for `libpython3.6m.so` on Linux or `libpython3.6m.dylib` on the Mac.

If we are unable to find a dynamic library such as `libpythonx.y.so` or `libpythonx.z.dylib`, it may be because Python is statically linked and the library is not present at all. This is dependent on the operating system and installation, and it is not always possible to detect it. In this case, we will receive an error message saying "Failed to find a valid python library!". To fix this, you may need to install additional OS packages or manually set the precise library location during `py/initialize!`.

This pathway has allowed us support Conda albeit with some work. For examples using Conda, check out the facial rec repository a)bove or look into how we build our test docker containers.

## Community

We like to talk about libpython-clj on Zulip as the conversations are persistent and searchable.

## Further Information

- Clojure Conj 2019 video and slides.
- development discussion forum
- design documentation
- scope and garbage collection docs
- examples
- docker setup
- pandas bindings (!!)
- nextjournal notebooks
- scicloj video
- Clojure/Python interop technical blog post
- persistent datastructures in python

## New To Clojure

New to Clojure or the JVM? Try remixing the nextjournal entry and playing around there. For more resources on learning and getting more comfortable with Clojure, we have an introductory document.

## Resources

- libpython C api
- create numpy from C ptr
- create C ptr from numpy

To install jar to local .m2 :

```
$ lein install
```

## Deploy to clojars

```
$ lein deploy clojars
```

This command will sign jar before deploy, using your gpg key. (see dev/src/build.clj for signing options)

## License