

Introducing **PyService**, a new
open-source **.Services** for
enterprise-level microservice
creation

Eugenio Andrieu
PyComZA 2021
7–8 October, 2021

Agenda

1 - Introduction

(3 minutes)

A brief explanation of what PIP.Services is.

2 - Demo

(17 minutes)

This demo will show how to build a simple microservice ("Hello World").

3 - Principles behind and architecture

(10 minutes)

Principles behind PIP.Services

A brief explanation of its architecture.

A brief explanation of a component's lifecycle

Advantages of this toolkit

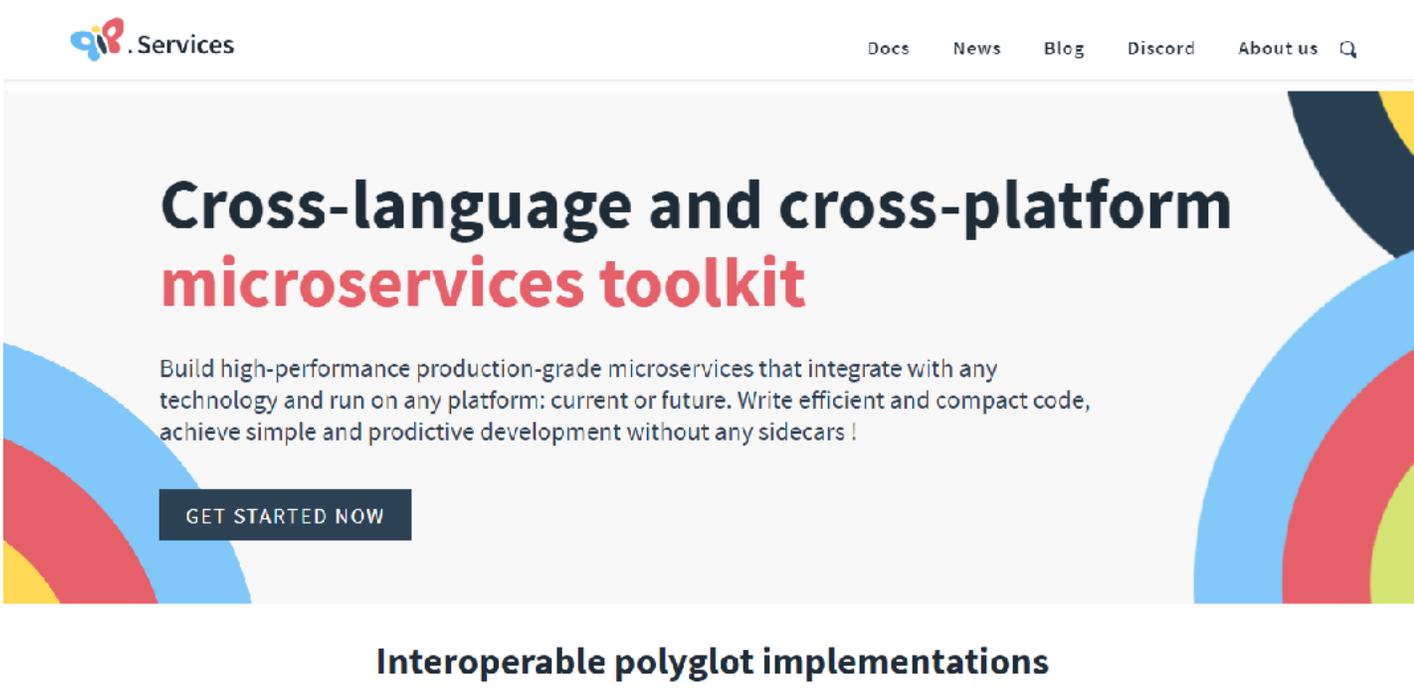
Part 1: Introduction

What is PIP.Services?

Pip.services is a **toolkit** that presents a collection of **ready-to-use components** used to build **microservices**.

- It has a large number of **reusable components** such as loggers, tracers, performance counters, distributed caches, distributed locks, credential stores, message queues, and configuration readers.
- At present, it supports **six programming languages**: .NET, Java, Node.js, Python, Golang, and Dart.

Where to find it

The image shows a screenshot of the PIP Services website. At the top left is the logo "pip.Services". To the right of the logo is a navigation menu with links for "Docs", "News", "Blog", "Discord", and "About us", followed by a search icon. The main content area features a large heading: "Cross-language and cross-platform microservices toolkit", where "microservices toolkit" is in red. Below the heading is a paragraph: "Build high-performance production-grade microservices that integrate with any technology and run on any platform: current or future. Write efficient and compact code, achieve simple and productive development without any sidecars!". A dark button with the text "GET STARTED NOW" is positioned below the paragraph. At the bottom of the screenshot, the text "Interoperable polyglot implementations" is displayed. The background of the main content area is decorated with colorful, overlapping circular shapes in shades of blue, red, yellow, and green.

pip.Services

Docs News Blog Discord About us Q

Cross-language and cross-platform microservices toolkit

Build high-performance production-grade microservices that integrate with any technology and run on any platform: current or future. Write efficient and compact code, achieve simple and productive development without any sidecars !

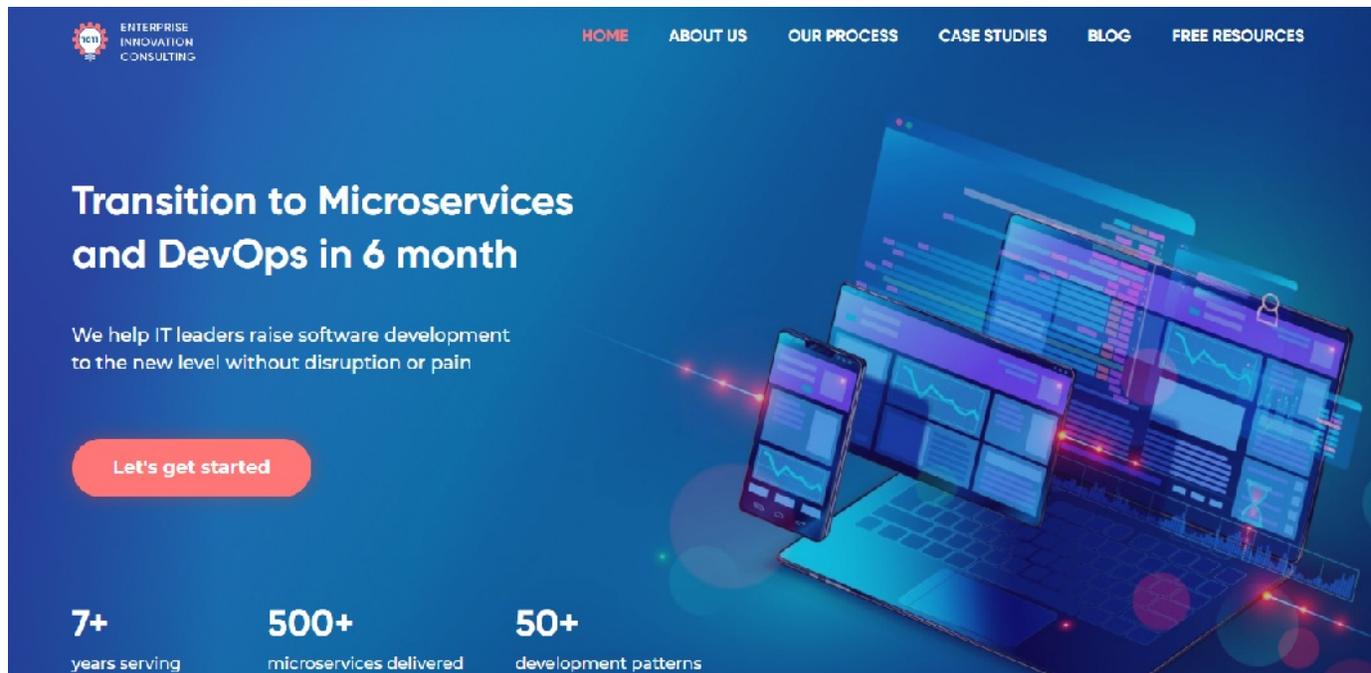
GET STARTED NOW

Interoperable polyglot implementations

<https://www.pipservices.org/>

Brief history

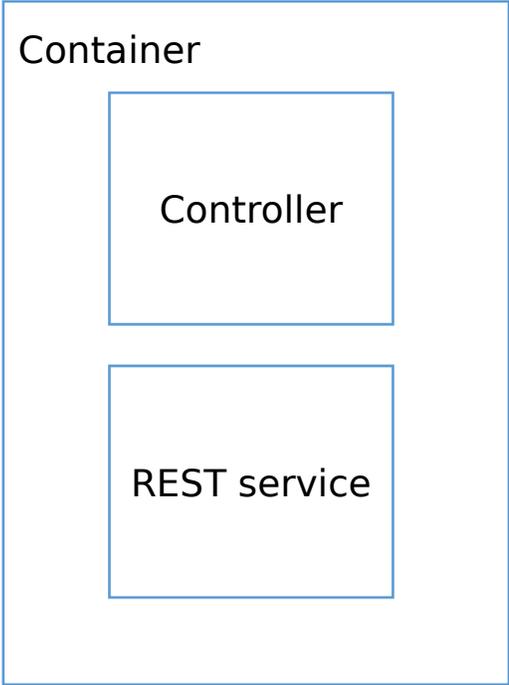
Initially developed by Enterprise Innovation Consulting and launched as open-source with an MIT license in June 2021.



The screenshot shows the homepage of Enterprise Innovation Consulting. The header includes the company logo and navigation links: HOME, ABOUT US, OUR PROCESS, CASE STUDIES, BLOG, and FREE RESOURCES. The main content area features a large heading: "Transition to Microservices and DevOps in 6 month". Below this is a sub-heading: "We help IT leaders raise software development to the new level without disruption or pain". A prominent orange button says "Let's get started". At the bottom, three statistics are displayed: "7+ years serving", "500+ microservices delivered", and "50+ development patterns". The background is a dark blue with glowing digital elements and a laptop displaying code and charts.

<https://www.entinco.com/>

Part 2: Demo



Where to find the code

Python:

[https://](https://github.com/pip-services-samples/service-quickstart-python)

github.com/pip-services-samples/service-quickstart-python

Node.js:

[https://](https://github.com/pip-services-samples/service-quickstart-nodejs)

[github.com/pip-services-samples/service-quickstart-node](https://github.com/pip-services-samples/service-quickstart-nodejs)
[x](https://github.com/pip-services-samples/service-quickstart-nodejs)

Golang:

[https://](https://github.com/pip-services-samples/service-quickstart-go)

github.com/pip-services-samples/service-quickstart-go

Part3: Principles behind and architecture

Principles behind

Symmetric implementation

It means that for every programming language it is implemented in, there are a common set of classes, methods, and method signatures.

/HelloWorldProcess.js

```
"use strict";

const rpc = require("pip-services3-rpc-nodex");
const container = require('pip-services3-container-nodex');
const factory = require("./HelloWorldServiceFactory");

class HelloWorldProcess extends container.ProcessContainer {
  constructor() {
    super('hello-world', 'HelloWorld microservice');
    this._configPath = './config.yml';
    this._factories.add(new factory.HelloWorldServiceFactory());
    this._factories.add(new rpc.DefaultRpcFactory());
  }
}

exports.HelloWorldProcess = HelloWorldProcess;
```

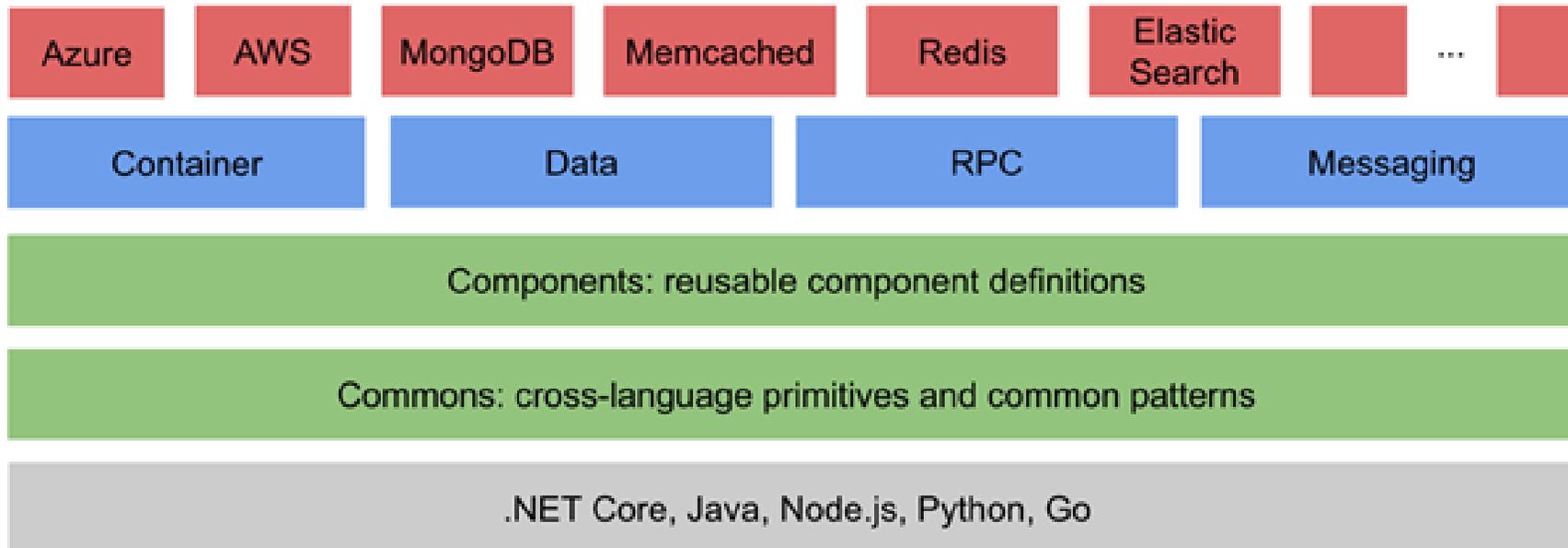
/HelloWorldProcess.py

```
# -*- coding: utf-8 -*-
from HelloWorldServiceFactory import HelloWorldServiceFactory
from pip_services3_container.ProcessContainer import ProcessContainer
from pip_services3_rpc.build import DefaultRpcFactory

class HelloWorldProcess(ProcessContainer):
    def __init__(self):

        super(HelloWorldProcess, self).__init__('hello-world', 'HelloWorld microservice')
        self._config_path = './config.yml'
        self._factories.add(HelloWorldServiceFactory())
        self._factories.add(DefaultRpcFactory())
```

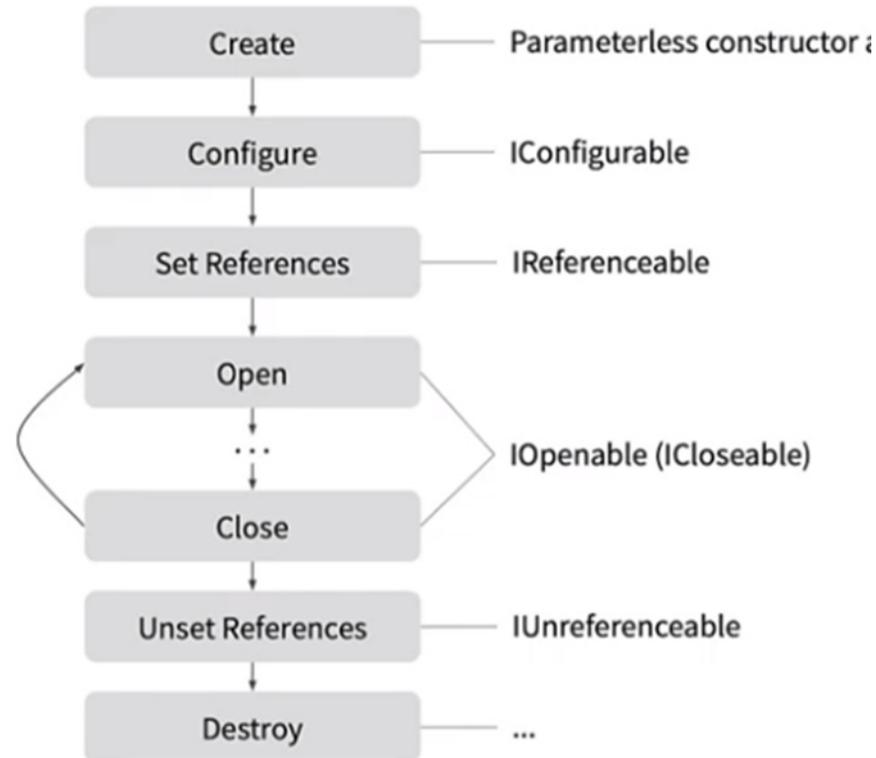
Architecture



A component's lifecycle

The life of a component

Component lifecycle



Advantages of PIP.Services

Why PIP.Services?

1. Both cross-language and cross-platform
2. Comprehensive
3. Incremental approach
5. Production-grade code
6. High-development productivity
7. Architectural flexibility



It allows businesses to create **long-living, adaptable** systems in a **short time**.

The end