Microservices

References

- https://en.wikipedia.org/wiki/Monolithic_application
- https://dzone.com/articles/scalable-cloud-computing-with-microservices
- http://www.dotnet-stuff.com/tutorials/c-sharp/understanding-loose-coupling-and-tight-coupling
- https://en.wikipedia.org/wiki/Loose_coupling
- https://en.wikipedia.org/wiki/Service-oriented_architecture
- https://en.wikipedia.org/wiki/Separation_of_concerns
- https://dzone.com/articles/microservices-vs-soa-2
- Microservices from Day One: Build robust and scalable software from the start, by Tim Schmelmer; Cloves Carneiro Jr., Published by Apress, 2016
- https://nameko.readthedocs.io/en/stable/
- https://en.wikipedia.org/wiki/RabbitMQ
- http://lucumr.pocoo.org/2015/4/8/microservices-with-nameko/

References

- http://werkzeug.pocoo.org
- https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface
- http://wsgi.readthedocs.io/en/latest/what.html
- http://werkzeug.pocoo.org/docs/0.12/serving/

Software Architectures

- To understand the significance of Microservices it is useful to take a brief look a tight and loose coupling and software architectures.

- First will also briefly discuss tightly-coupled and loosely-coupled software

- Next we will briefly examine:
  – Monolithic Software Architectures
  – Service Orientated Architectures

Tight Coupling

- **tightly coupled** objects/code need to know about each other.
  - a class that calls methods in another class − either directly via accessing the class or via an interface implemented by the class

    needs to know about the other object's methods/ properties/parameters

Tight Coupling

– The objects are highly dependent on each other – even if object calls are via interfaces

– Tightly coupled objects must be deployed (installed) together for the system to function

Tight Coupling

- Tightly coupled objects, especially those deployed on the same computer/machine tend to require use of the same technologies
  - for example − a C++ data access module (barring the use of inter-technology integration mechanisms) has to be called from C++ code

Loose Coupling

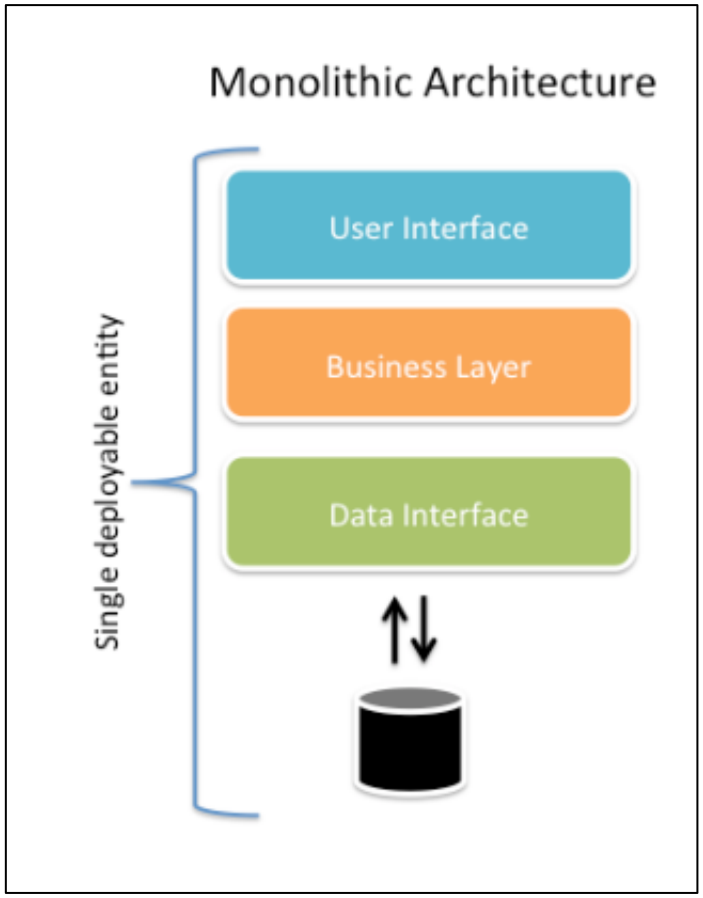
- From https://en.wikipedia.org/wiki/Loose_coupling:

  a **loosely coupled** system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components.

Loose Coupling

- Loosely coupled system can use different technologies
- For example – the interaction between a web browser and web server is loosely coupled via use of protocols:
  - communications is done via http(s)
  - ui is specified via HTML/CSS and client-side Javasscript

Monolithic Software Architecture



From:

https://dzone.com/articles/scalable-cloud-computing-with-microservices

A monolithic application is a single deployable entity – i.e. it all is deployed together. The deployment may be on a single machine (e.g. a desktop application) or as a multi-tier over the network/internet.

Monolithic applications are usually "**tightly coupled**". Communications between layers can use a variety of API mechanisms including: method/function calls, web services calls, protocol-based exchanges using sockets, etc.

However, monolithic applications may use **"loosely coupled**" communications mechanism like message-passing, commands, etc.

Monolithic Software Architecture

- Advantages of Monolithic approach:
  - faster initial deployment
  - easier to understand and comprehend
  - we all learned how to develop software this way in school
  - typically use the same software technology stack throughout all tiers of the application
    - e.g. C# Desktop app talks to an ASP.NET web service

Monolithic Software Architecture

- Disadvantages
  - does not scale out as easily − especially if the tiers are tightly coupled
  - pre-dates BigData, Cloud Computing − based on "legacy thinking"

Service Oriented Architecture (SOA)

- From https://en.wikipedia.org/wiki/Service-oriented_architecture

  A service-oriented architecture (SOA) is a style of software design where services are provided to the other components by **application components**, through a communication protocol over a network.

- An **application component** is a portion/sub-system of an application/system where the application is divided into "parts"/ components where each components handles a specific set of tasks related to an area.

Service Oriented Architecture (SOA)

- Application components , within our discussion context, are based on a software engineering concept called "**Separation of Concerns**" (SoC)
- The general idea is each component (within an SoC architecture) has handles get/managing/setting a particular set of data the application needs

Service Oriented Architecture (SOA)

- For example an online store will have application components to handle distinct areas like: a user's shopping cart, checkout. credit card processing

- In a monolithic software architecture these would be modules that commutate with each other , where each modules know the specific interfaces/methods properties of the other modules

Service Oriented Architecture (SOA)

- In a Service Oriented Architecture these areas (shopping cart, checkout, credit card processing), credit card processing) do not have direct knowledge of each other.

- Instead communications between the components will use (more) loosely coupled communications mechanism like REST and Messaging)

Service Oriented Architecture (SOA) Side Track: **Web Services**, REST

- A **Web Service** is an API that, in practice, communicates over HTTP/HTTP(s)

- The caller and the called objects do NOT have to use the same technology stacks.

Service Oriented Architecture (SOA) Side Track: Web Services, **REST**

- **REST** – briefly –
  - REST is a replacement for SOAP/WSDL-based web service communications
  - All communications is over HTTP/HTTP(s) expressed as HTTP operations (e.g. get, post, put).
  - REST based server do NOT keep state in-between calls.

Service Oriented Architecture (SOA) Side Track: Web Services, **REST**

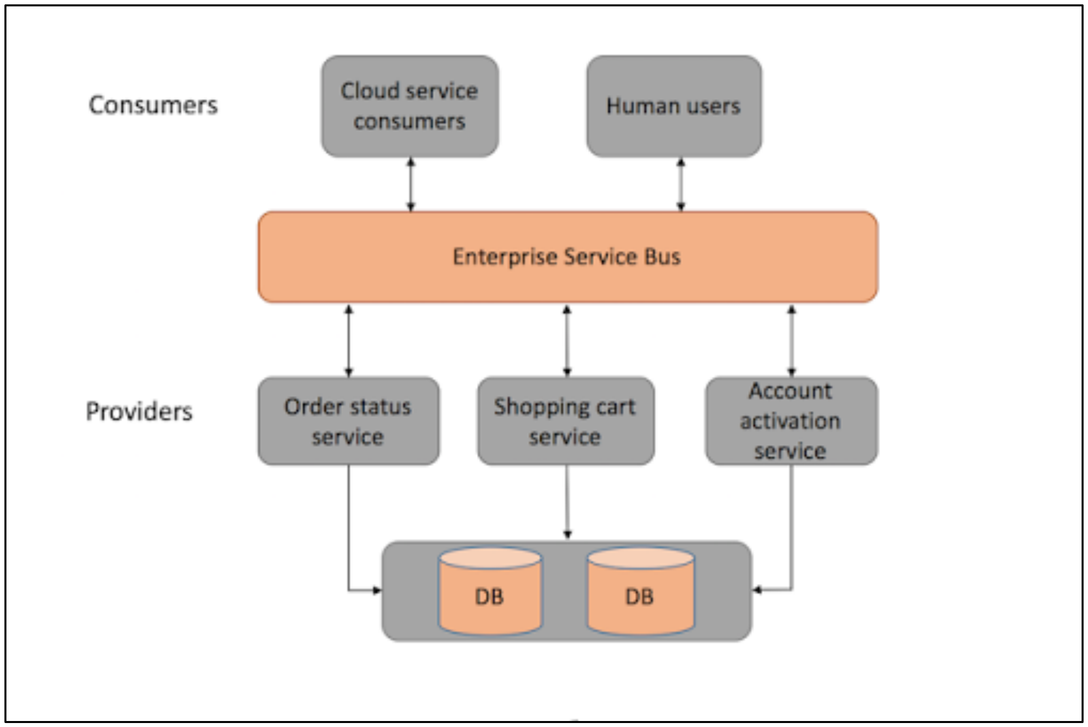
- **REST** – briefly –
  - Virtually all modern web services use REST
  - REST is based on Roy Fielding's PhD dissertation (from UC Irvine):

    https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- OK – back to SOA

Service Oriented Architecture (SOA)

- In summary − SOA is a design practice where different area (concerns) are separated into different independent applications called components.
- Each application/component can run without knowledge of the others.
- Components communicate via loosely-coupled protocols like REST

Service Oriented Architecture (SOA)



An "**enterprise service bus**" is a communication mechanism that supports point-to-point communications between components

SOA architectural diagram from:

https://dzone.com/articles/microservices-vs-soa-2

Microservices

- There is no exact definition of a Microservice.
- Instead think of Microservices as a "practice", a way of designing, implementing, and deploying software modules:
  - that are small (as little code as possible)
  - do not depend on other microservices
  - are independently deployed

Microservices

- The following slides provide some descriptions and definitions of Microservices

Microservices

- From: https://dzone.com/articles/microservices-vs-soa-2

  Microservices is a software architecture pattern in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs.

Microservices

- From reference: Microservices from Day One:

  It is hard to determine exactly who first used the term microservice and when. While it seems most likely that although the term first appeared on the scene around 2011, some companies adopted this style much earlier

Microservices

- From reference: Microservices from Day One, attributed to Randy Shoup − Micrsoservices have 3 main features:
- They have a **<u>very focused scope that concentrates on a very few functionalities</u>** (at times a single one), expressed by a small and also well-defined API.
- They are all **<u>very modular, and independently deployable</u>**, essentially elevating the rules of software encapsulation and modularity from the program to the deployment unit level.
- Finally, microservices have **<u>isolated persistence, meaning that they do not share persistent storage, such as a common database</u>**

Microservices

- The main difference between SOA components are Microservices are:
  - microservices are smaller than SOA components
  - microservices do NOT share centralized storage like SOA components do

Microservices

– SOA components normally use enterprise service bus architectures to communicate, each microservice uses provides a small "contract" (e.g. loosely-coupled interface

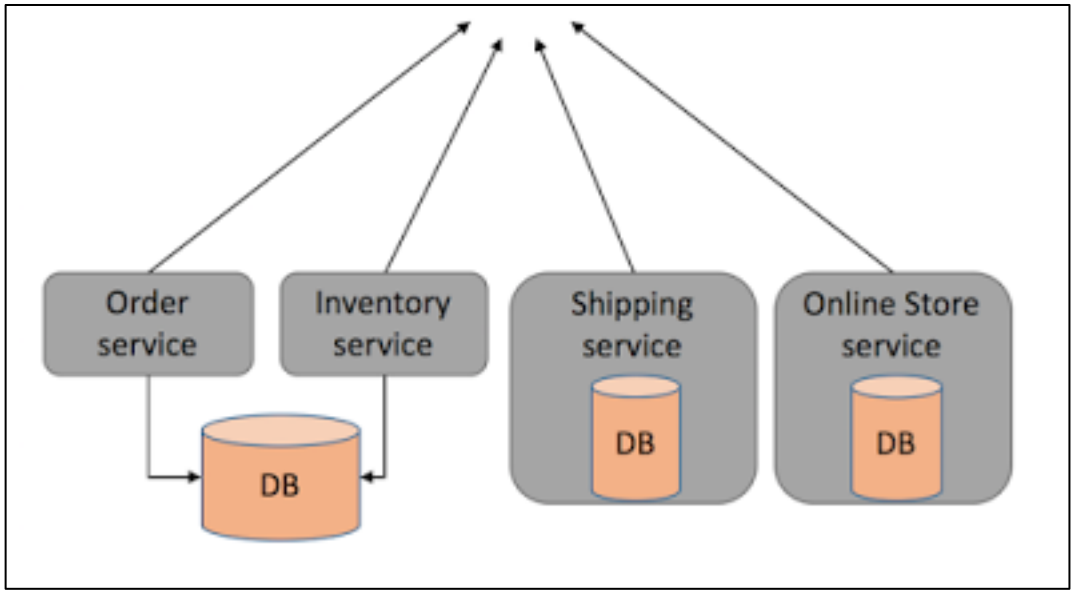
– At a company that uses microservices the software engineers/developers that built a micorservce are normally responsible for operationally supporting it − i.e. there are no barriers between software and operations

Microservices

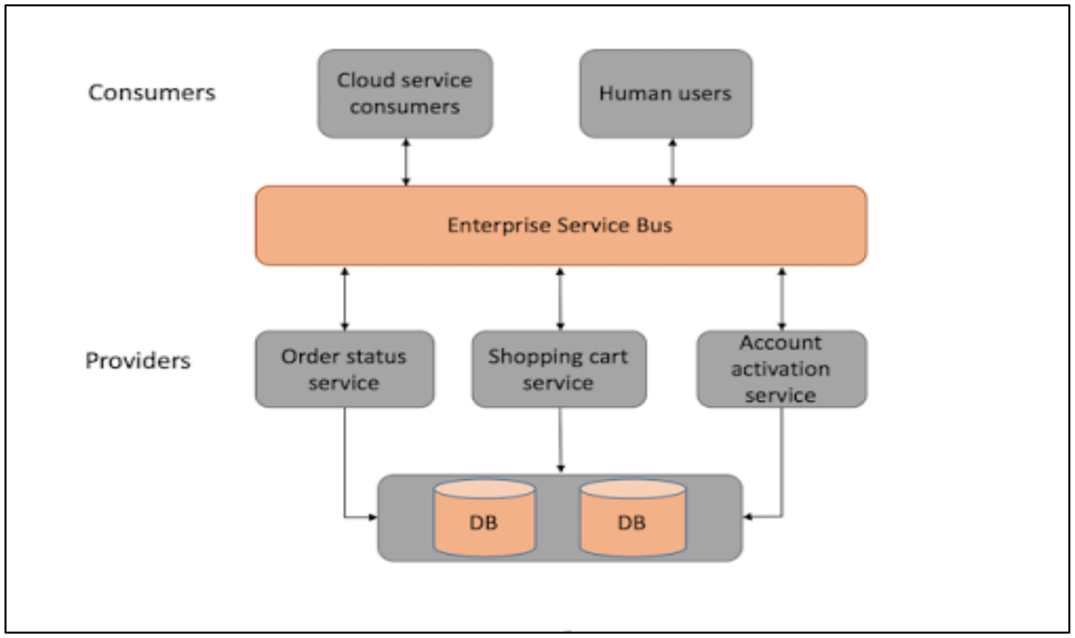
– As long as a Microservice is being used, it is supported

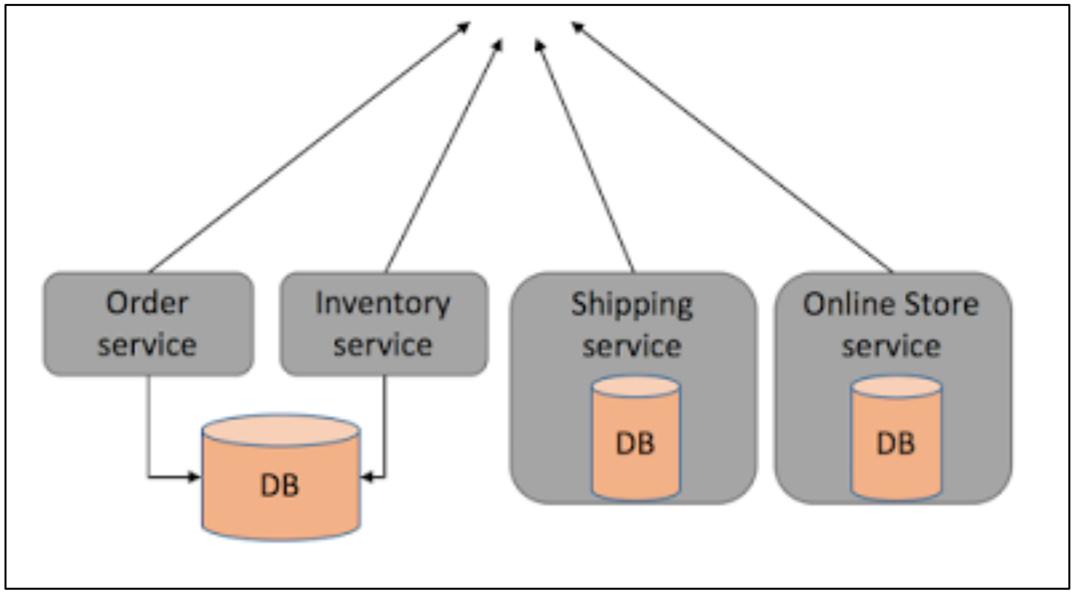
Microservices

from https://dzone.com/articles/microservices-vs-soa-2



SOA (top) vs. Microservices (bottom)





Microservices vs. SOA

- SOA: services share the data storage
- Microservice: each service can have an independent data storage

- Microservices are significantly smaller units/modules in an SOA system,

Microservices

- Also from reference: From reference: Microservices from Day One − is a "rules of engagement" chapter.
- This chapter provides a high-level operational view and guidance for designing and developing microservices
- The next few slides discuss some top level concepts from this chapter

Microservices

- Customer Facing Applications (services that provide APIs to application that customer use), cannot directly access data stores.
- No Microservice Accesses Another Microservice Data Store

Microservices

- Take the time to setup an infrastructure that makes bring up new microservices easy
  - called "Invest in Making Spinning Up New Services Trivial" in the book
  - basically these means − create reusable ways to automate tasks that will be performed many times (called templates in the book)

Microservices

– have all teams involved agree on how things are done while also being open to changing how things on done

– create/use repositories like git

– define data stores (possibly per microservice)

– use CI (continuous integration)

Microservices

– define a deployment methodology/methodologies

– deploy to:
    - development (first)
    - staging (testing)
    - production

– make sure you build in monition and logging

Microservices

– define microservice APIs in a consistent way between groups

  for example − all APIs return true/false where false means an error occurred

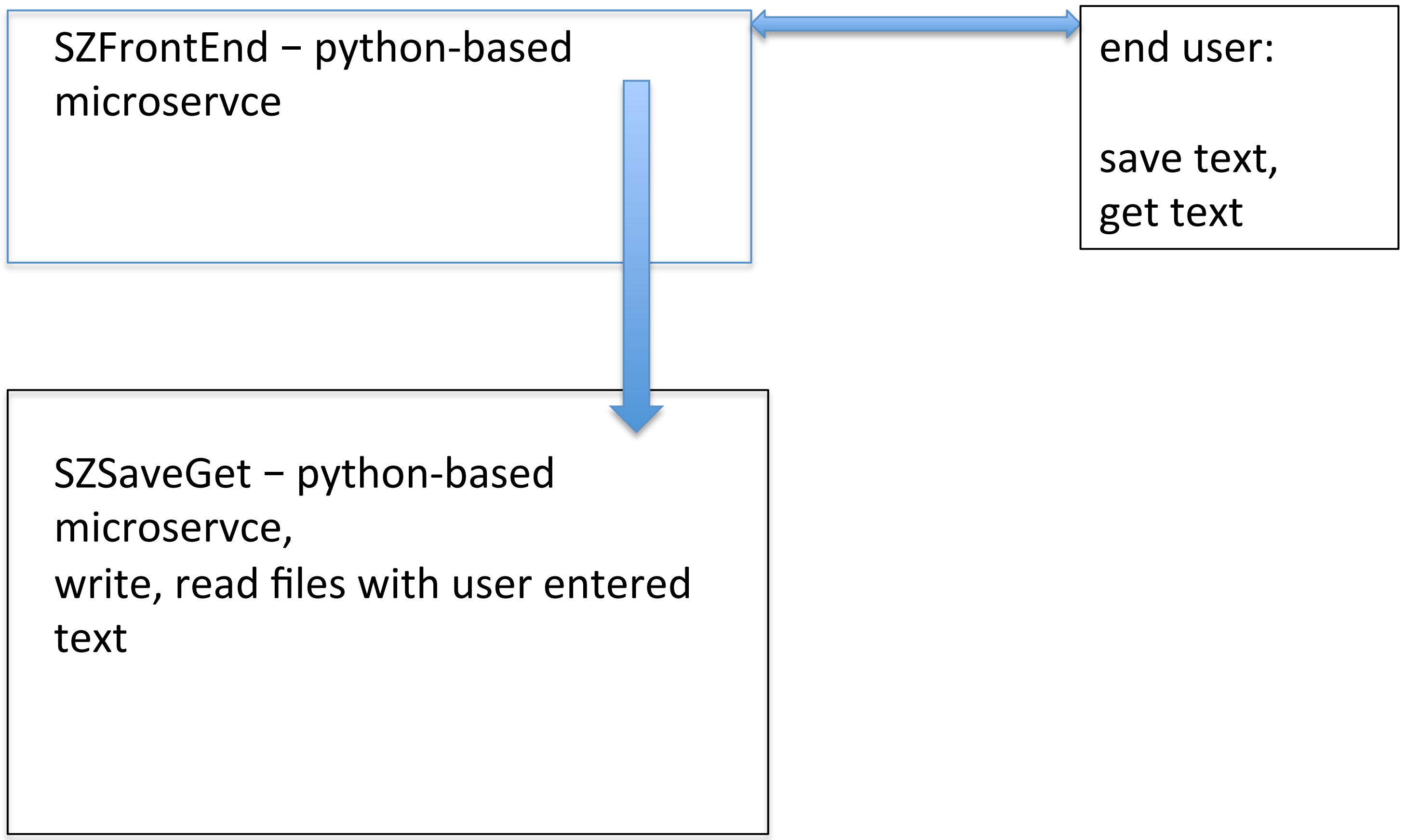
Microservices

- define APIs using a language independent coding mechanism
  - the term IDL in general means interface description language
  - for example: Swagger − provides a way to specify RESTFul APIs

Microservice Example

- The following slides will setup a small Microservces example using:
  - **Nameko** – a microservice framework for Python
  - A small Python microservice called **SZSaveGet** that takes a filename and some text and writes and reads the text to/from a file
  - Another small Python-based microservice, called **SZFrontEnd** that provide a simple web-based front-end that allows users to access SZSaveGet

Microservice Example


SZFrontEnd – python-based microservce
end user:
save text, get text
SZSaveGet – python-based microservce,
write, read files with user entered text

Microservice Example

- Install Nameko using pip:

```
pip install nameko
```

Microservice Example

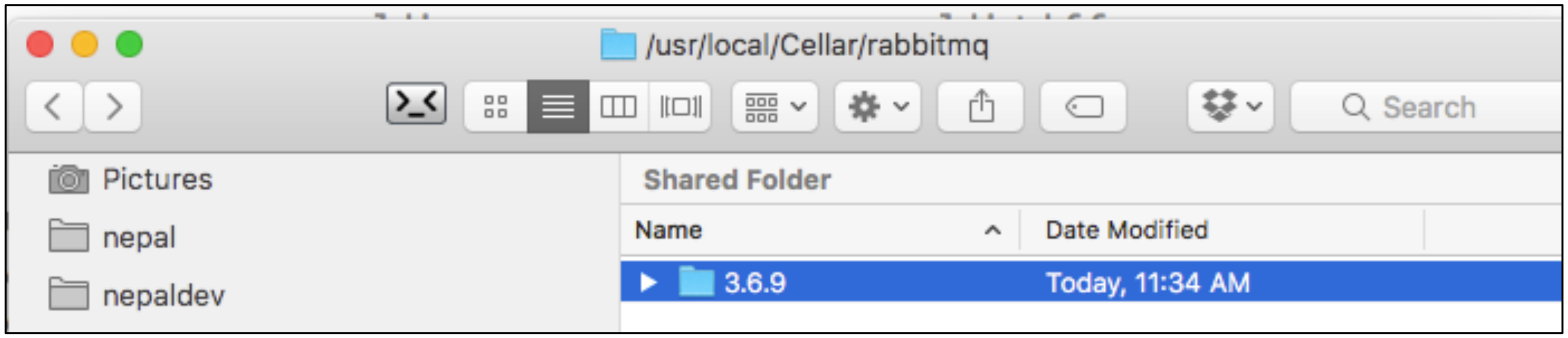
- Nameko makes use of RabbitMQ
- RabbitMQ is an open source message bus

- To install RabbitMQ on a Mac:

  ```
  brew install rabbitmq
  ```

- Homebrew will install all needed dependencies (RabbitMQ)

Microservice Example



/usr/local/Cellar/rabbitmq

Pictures
nepal
nepaldev

Shared Folder

Name	Date Modified
3.6.9	Today, 11:34 AM

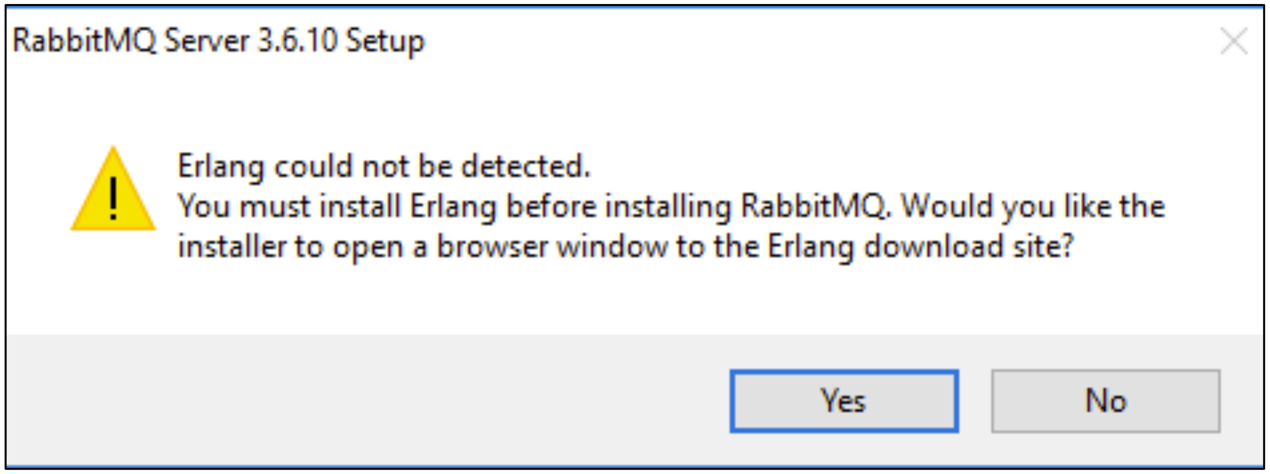
Microservice Example

- To install RabbitMQ on Windows get and install:

  https://www.rabbitmq.com/releases/rabbitmq-server/v3.6.10/rabbitmq-server-3.6.10.exe

Microservice Example

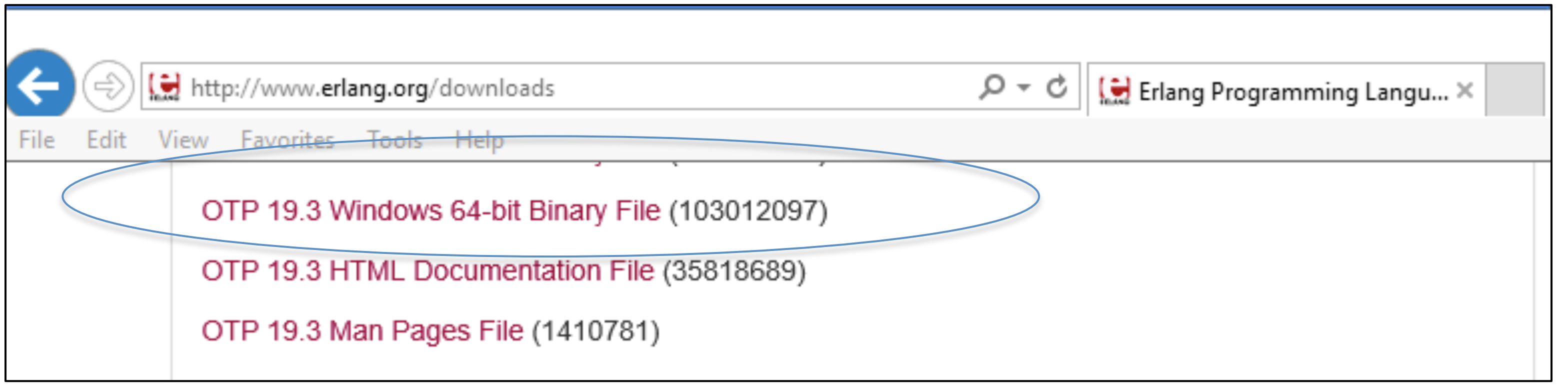
- RabbitMQ requires Erlang – if you see this, get and install Erlang:



RabbitMQ Server 3.6.10 Setup

Erlang could not be detected.
You must install Erlang before installing RabbitMQ. Would you like the installer to open a browser window to the Erlang download site?

Microservice Example

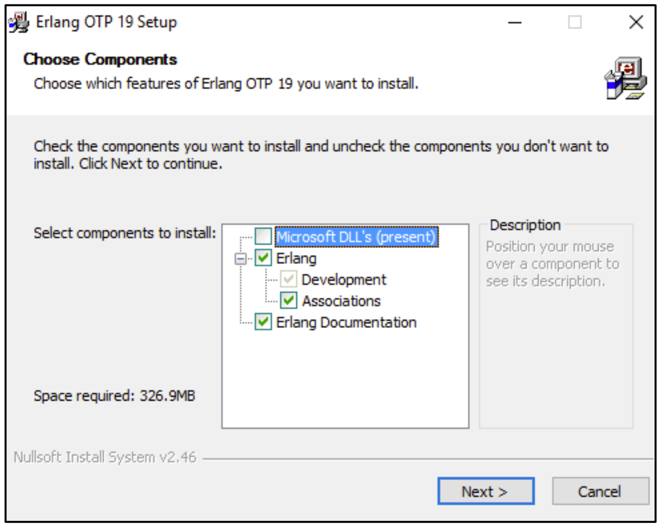


http://www.erlang.org/downloads
Erlang Programming Langu...
File Edit View Favorites Tools Help

OTP 19.3 Windows 64-bit Binary File (103012097)

OTP 19.3 HTML Documentation File (35818689)

OTP 19.3 Man Pages File (1410781)



Erlang OTP 19 Setup

Choose Components

Choose which features of Erlang OTP 19 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [ ] Microsoft DLL's (present)
- [✓] Erlang
  - [✓] Development
  - [✓] Associations
- [✓] Erlang Documentation

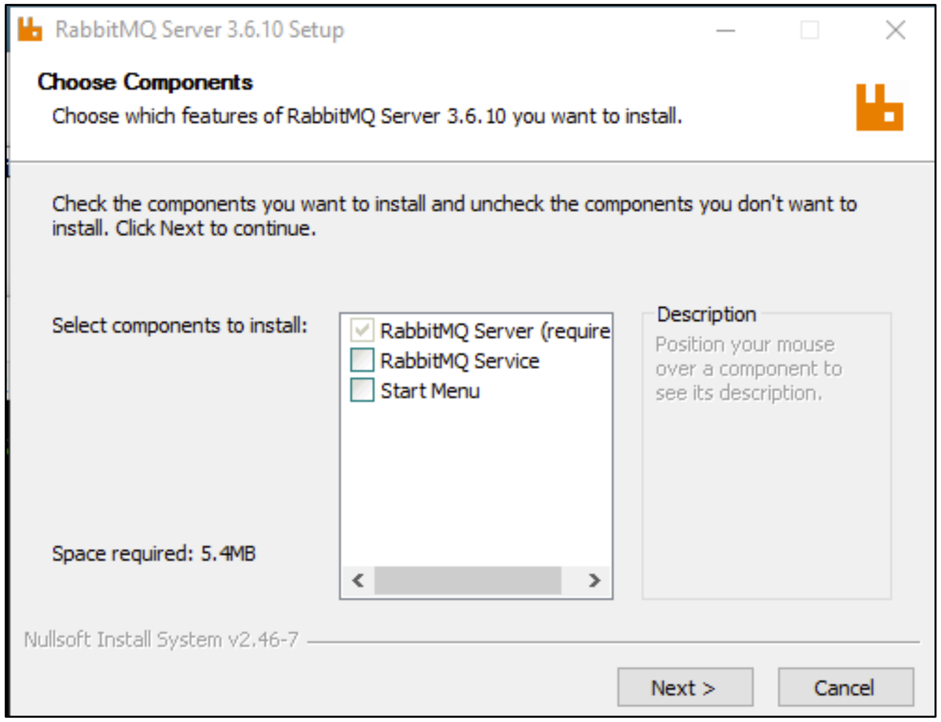
Description

Position your mouse over a component to see its description.

Space required: 326.9MB

Nullsoft Install System v2.46

Microservice Example


RabbitMQ Server 3.6.10 Setup

Choose Components

Choose which features of RabbitMQ Server 3.6.10 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] RabbitMQ Server (require
- [ ] RabbitMQ Service
- [ ] Start Menu

Description

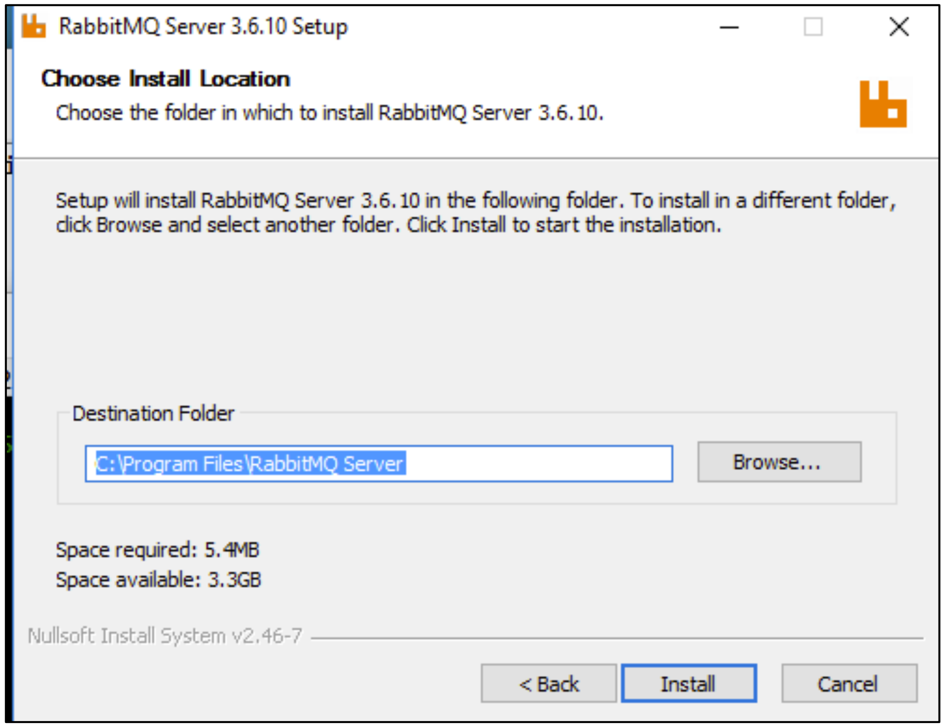
Position your mouse over a component to see its description.

Space required: 5.4MB

Nullsoft Install System v2.46-7

On Windows , after Erlang is installed, install RabbitMQ. Your instructor only installed the server and unchecked the service

Microservice Example



RabbitMQ Server 3.6.10 Setup

Choose Install Location

Choose the folder in which to install RabbitMQ Server 3.6.10.

Setup will install RabbitMQ Server 3.6.10 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

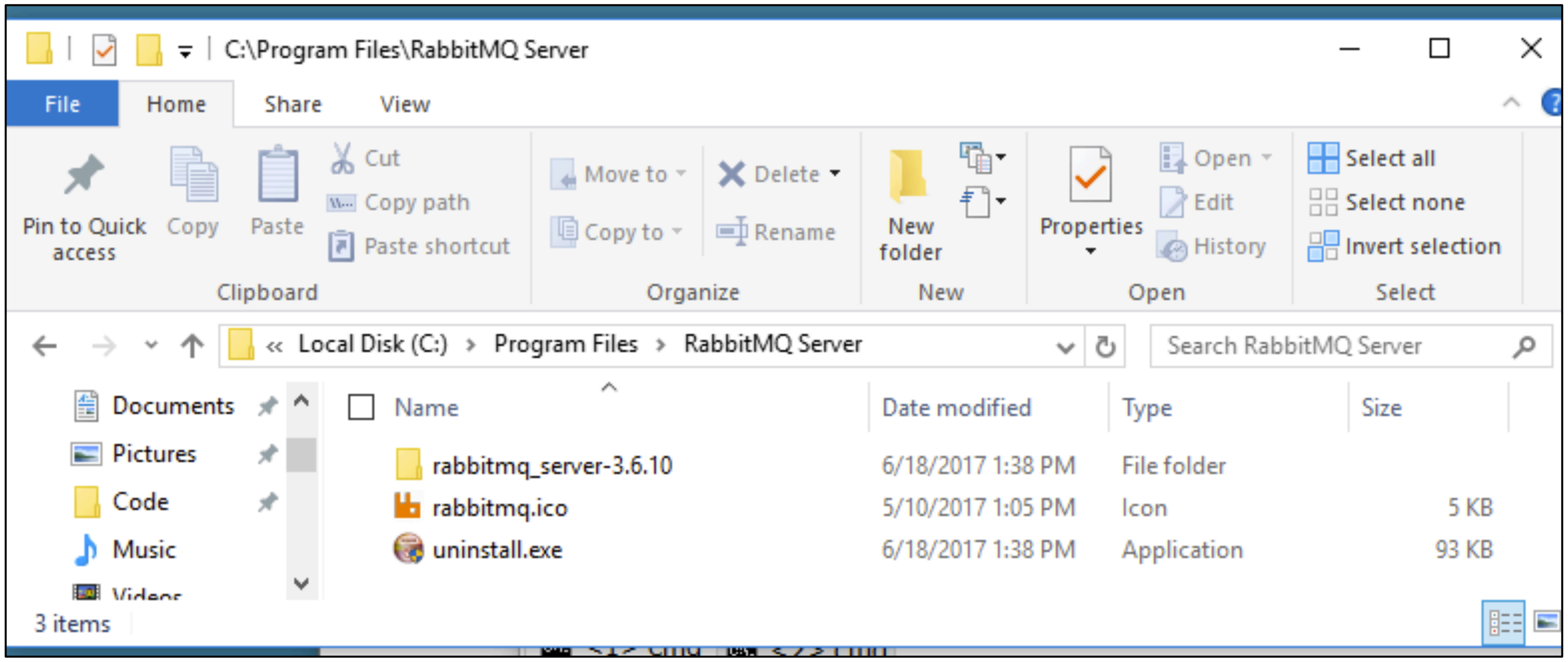
Destination Folder

C:\Program Files\RabbitMQ Server

Space required: 5.4MB
Space available: 3.3GB

Nullsoft Install System v2.46-7

Microservice Example



Name	Date modified	Type	Size
rabbitmq_server-3.6.10	6/18/2017 1:38 PM	File folder	
rabbitmq.ico	5/10/2017 1:05 PM	Icon	5 KB
uninstall.exe	6/18/2017 1:38 PM	Application	93 KB

Microservice Example

- As we discussed, Nameko used the rabbitmq server.
- Before running any Nameko services run rabbittmq

Microservice Example

- On a Mac, the script to run the rabbitmq server is the rabbitmq-server bash shell script located in:

  /usr/local/Cellar/rabbitmq/3.6.9/sbin

Microservice Example

- On a Windows, the script to run the rabbitmq server is the rabbitmq-server bash shell script located in:

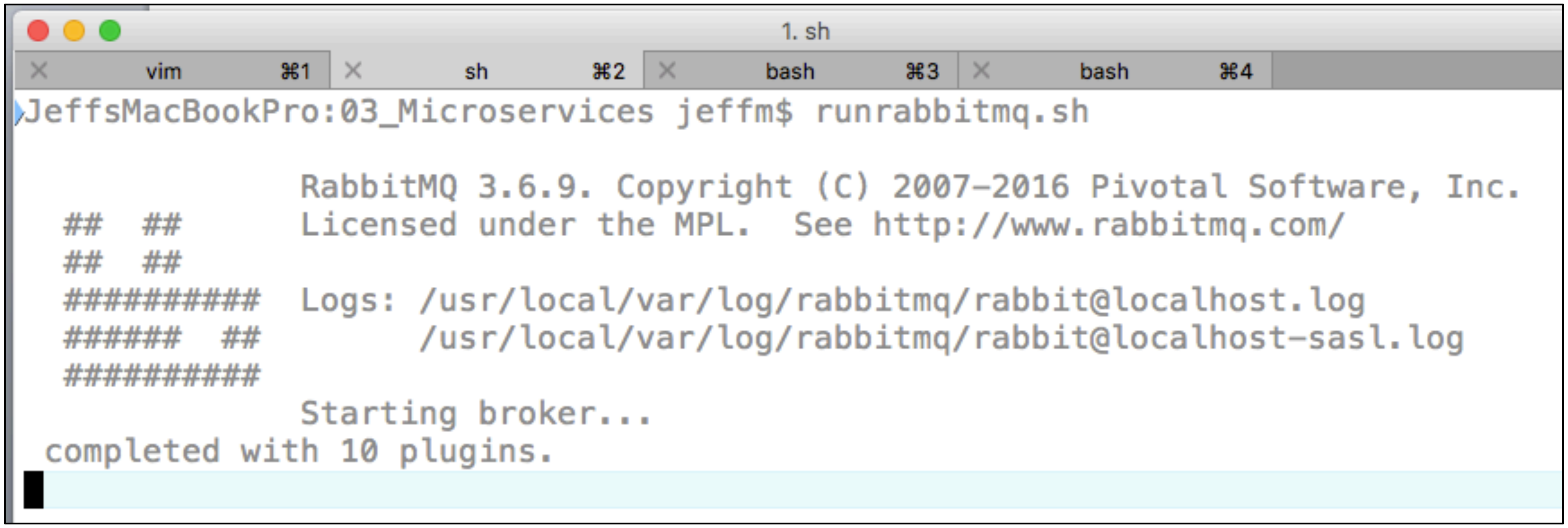
  C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.10\sbin

Microservice Example

- On a Mac, the following simple Bash shell script was created to run rabbitmq:

```
#!/bin/bash
/usr/local/Cellar/rabbitmq/3.6.9/sbin/rabbitmq-server
```

Microservice Example



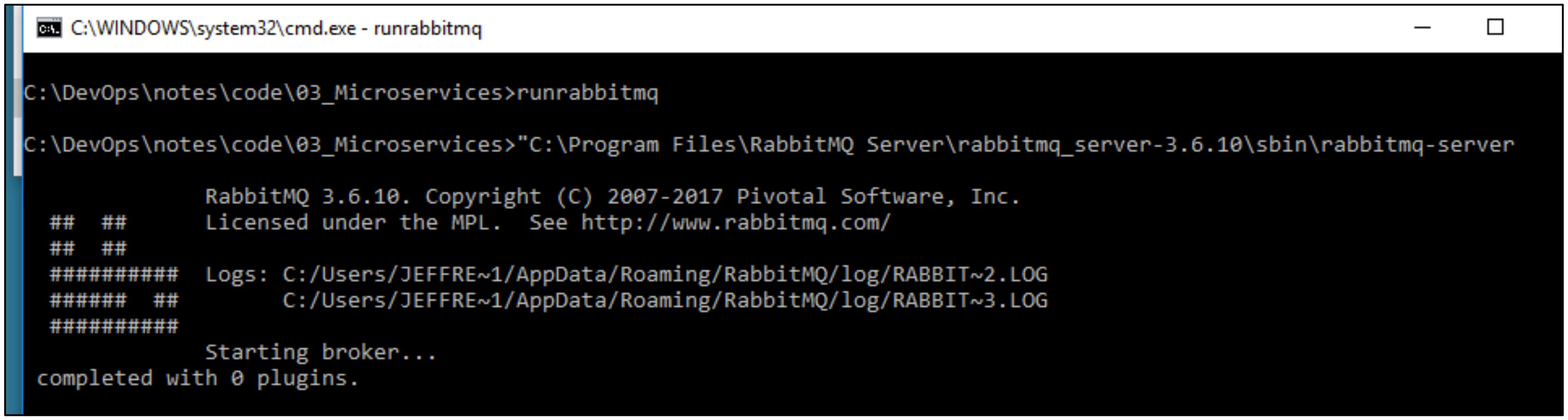
```
JeffsMacBookPro:03_Microservices jeffm$ runrabbitmq.sh

              RabbitMQ 3.6.9. Copyright (C) 2007-2016 Pivotal Software, Inc.
  ##  ##      Licensed under the MPL.  See http://www.rabbitmq.com/
  ##  ##
  ##########  Logs: /usr/local/var/log/rabbitmq/rabbit@localhost.log
  ######  ##        /usr/local/var/log/rabbitmq/rabbit@localhost-sasl.log
  ##########
              Starting broker...
 completed with 10 plugins.
```

Microservice Example

- To start rabitmq-server on Windows a simple batch file, runrabbitmq.bat, was created to start the server:

```
"C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.10\sbin\rabbitmq-server.bat"
```

Microservice Example



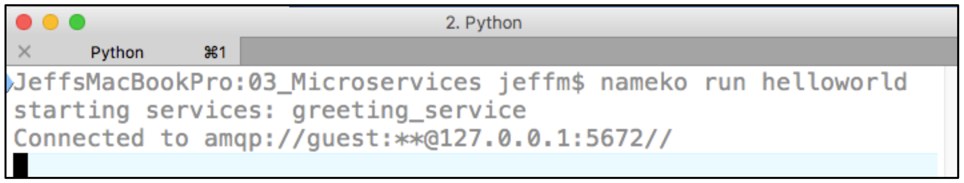
```
C:\DevOps\notes\code\03_Microservices>runrabbitmq

C:\DevOps\notes\code\03_Microservices>"C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.10\sbin\rabbitmq-server

              RabbitMQ 3.6.10. Copyright (C) 2007-2017 Pivotal Software, Inc.
  ##  ##      Licensed under the MPL.  See http://www.rabbitmq.com/
  ##  ##
  ##########  Logs: C:/Users/JEFFRE~1/AppData/Roaming/RabbitMQ/log/RABBIT~2.LOG
  ######  ##        C:/Users/JEFFRE~1/AppData/Roaming/RabbitMQ/log/RABBIT~3.LOG
  ##########
              Starting broker...
 completed with 0 plugins.
```

Microservice Example

```
# helloworld.py

from nameko.rpc import rpc

class GreetingService:
    name = "greeting_service"

    @rpc
    def hello(self, name):
        return "Hello, {}!".format(name)
```

Simple nameko hello world example from Nameko doc

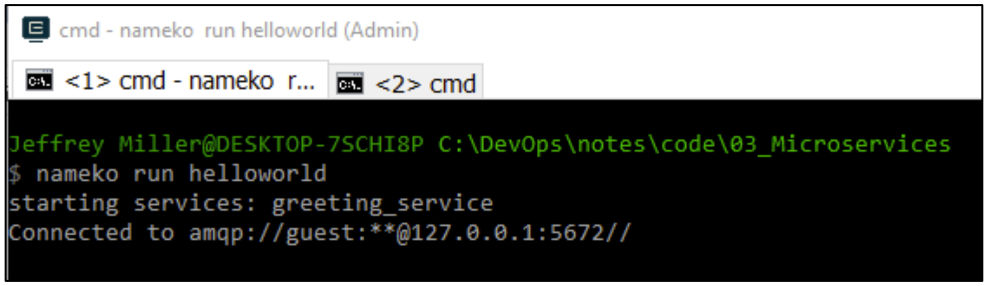
Microservice Example



```
JeffsMacBookPro:03_Microservices jeffm$ nameko run helloworld
starting services: greeting_service
Connected to amqp://guest:**@127.0.0.1:5672//
```

start the Nameko service, open a terminal in a Mac:

$ nameko run helloworld



```
Jeffrey Miller@DESKTOP-7SCHI8P C:\DevOps\notes\code\03_Microservices
$ nameko run helloworld
starting services: greeting_service
Connected to amqp://guest:**@127.0.0.1:5672//
```

in Windows, open a command prompt, start the Nameko service

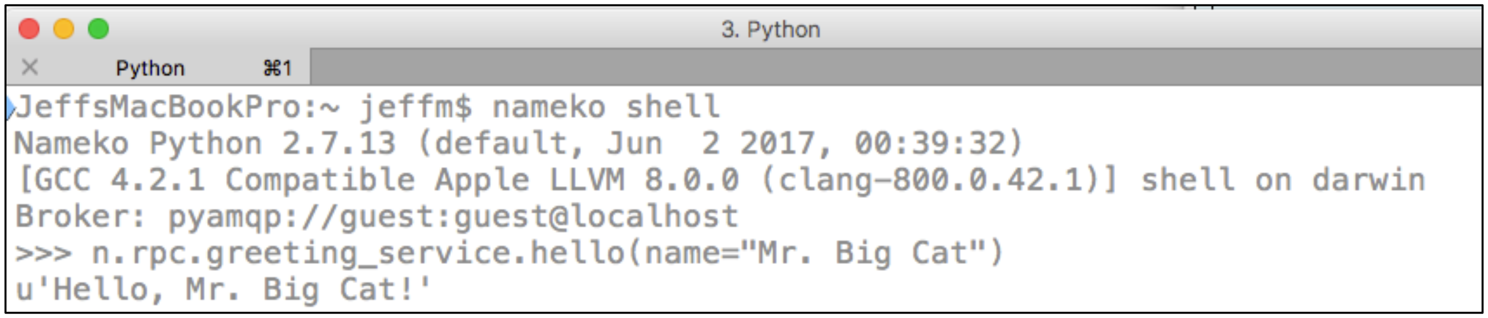
$ nameko run helloworld

remember to start rabbitmq-server before starting nameko – if you do not start rabbitmq-server first, you will see a connection refused error

Microservice Example

To call the Nameko-based Microservice:

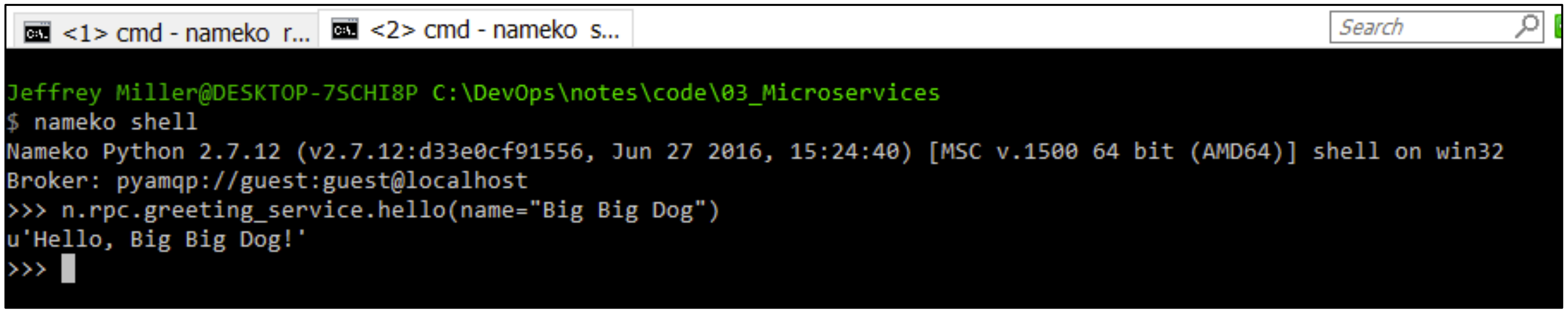
- start a Nameko shell
- write code to call hello:



3. Python

Python ⌘1

```
JeffsMacBookPro:~ jeffm$ nameko shell
Nameko Python 2.7.13 (default, Jun  2 2017, 00:39:32)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] shell on darwin
Broker: pyamqp://guest:guest@localhost
>>> n.rpc.greeting_service.hello(name="Mr. Big Cat")
u'Hello, Mr. Big Cat!'
```



<1> cmd - nameko r... <2> cmd - nameko s...

```
Jeffrey Miller@DESKTOP-7SCHI8P C:\DevOps\notes\code\03_Microservices
$ nameko shell
Nameko Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:24:40) [MSC v.1500 64 bit (AMD64)] shell on win32
Broker: pyamqp://guest:guest@localhost
>>> n.rpc.greeting_service.hello(name="Big Big Dog")
u'Hello, Big Big Dog!'
>>>
```

Microservice Simple Web Example

- The next example will place a Nameko-based microservice behind an http request.

- Nameko has a simple built-in extension that supports http get and post. We will use this mechanism as part of this example.

- Running Rabbitmq is not required when to use Nameko's http-based access

Microservice Simple Web Example

– In a real production deployment, you will want to integrated your microservices into a production-level web server like Apache.

– For Python-based Web code you can use WSGI - **Web Server Gateway Interface**

Microservice Simple Web Example

- WSGI specifies how:
  - a web server communicates with applications integrated into a web server
    - applications "plugged" into a web server are called **Web Applications**
  - how to change web application together to process the same request

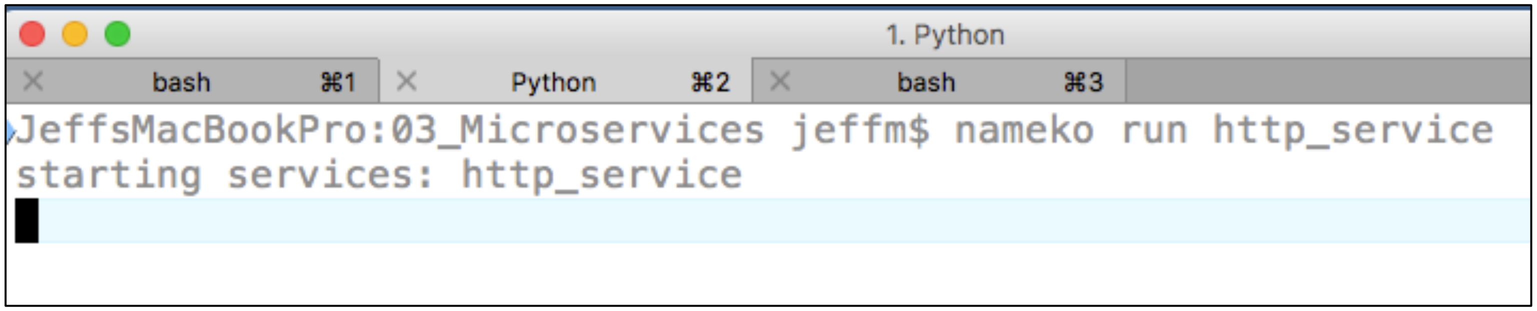
- There are several commonly used Python libraries that implement WSGI, including Werkzeug.

Microservice Simple Web Example

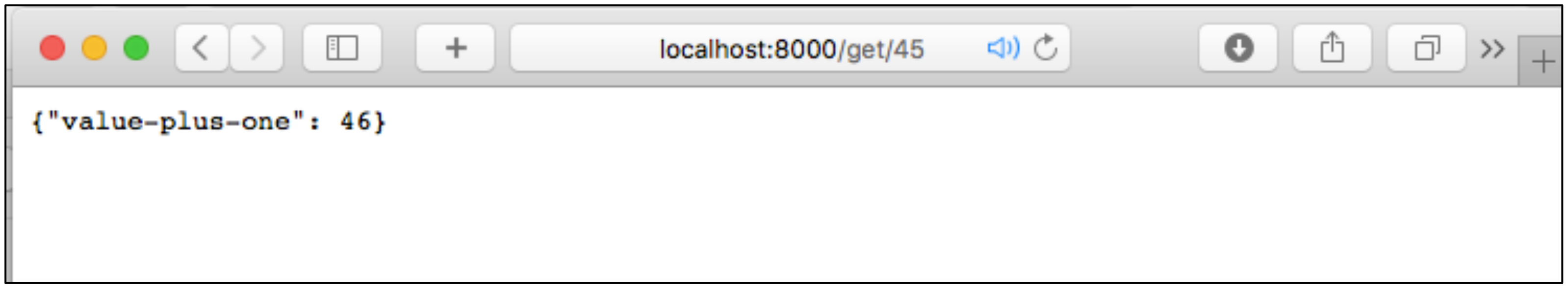
```
import json
from nameko.web.handlers import http

class HttpService:
    name = "http_service"

    @http('GET', '/get/<int:value>')
    def get_method(self, request, value):
        return json.dumps({'value-plus-one': value + 1})

    @http('POST', '/post')
    def do_post(self, request):
        return u"received: {}".format(request.get_data(as_text=True))
```

http_service.py

Microservice Simple Web Example



```
JeffsMacBookPro:03_Microservices jeffm$ nameko run http_service
starting services: http_service
```



```
{"value-plus-one": 46}
```

Microservice Frameworks

- Many other Microservice frameworks and tools are widely used, including:
  - Hystix from Netflix
  - Spring Boot
  - Prometheus

- Site: https://jaxenter.com/microservices-trends-2017-survey-133265.html - has a list of these frameworks and tools

Microservice Frameworks

- You can develop a microservice "raw" without a framework.
- However, using a framework can increase your efficiency abd speed of deployment.
- We looked at Nameko , a Python-based Microservices Framework.

Microservice Summary

- In this section we discussed microservice basics
- A microservice can be written in any programming language
- A microservice is a deployable unit that, if it has storage, does not share that storage