The Example with Cloud Foundry

In this lesson, we'll look at an example with cloud foundry.

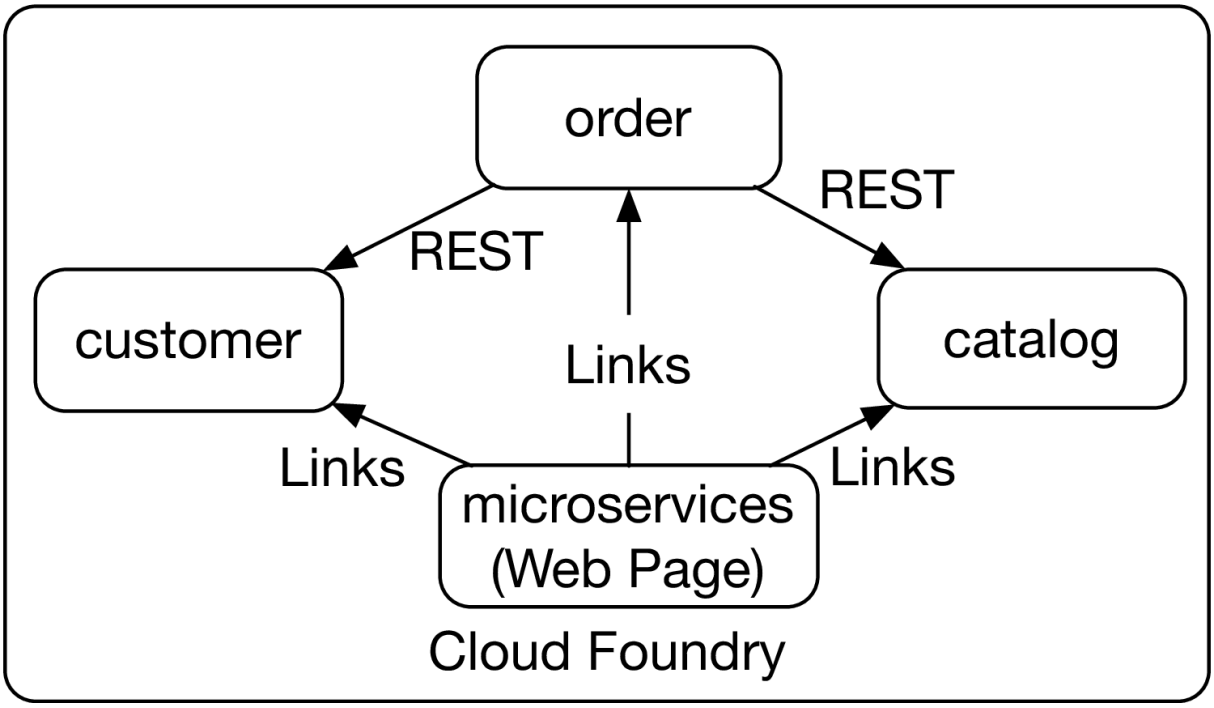
We'll cover the following

Introduction

The microservices in this example are identical to the examples from the previous chapters (see Example (https://www.educative.io/collection/page/10370001/5441945024331776/6627192651907072)).

- The **catalog** microservice controls the information concerning the goods.
- The **customer** microservice stores the customer data.
- The **order** microservice can receive new orders. It uses the catalog and customer microservice via REST.

- In addition, there is the **Hystrix dashboard**, a Java application for visualizing the monitoring of the Hystrix circuit breaker.
- Finally, there is a web page **microservices** that contains links to the microservices and thus facilitates the entry into the system.



The Microservices System in Cloud Foundry

Starting Cloud Foundry

A detailed description of how the example can be built and started is provided at https://github.com/ewolff/microservice-cloudfoundry/blob/master/HOW-TO-RUN.md (https://github.com/ewolff/microservice-cloudfoundry/blob/master/HOW-TO-RUN.md).

The Cloud Foundry example is available at https://github.com/ewolff/microservice-cloudfoundry (https://github.com/ewolff/microservice-cloudfoundry). Download the

code with `git clone https://github.com/ewolff/microservice-cloudfoundry.git`.

To start the system, the application is first compiled with Maven. To do so, you have to execute `./mvnw clean package` (macOS, Linux) or `mvnw.cmd clean package` (Windows) in the sub directory `microservice-cloudfoundry-demo`.

The example is supposed to be started on a local Cloud Foundry installation. The required installation is described at https://pivotal.io/pcf-dev (https://pivotal.io/pcf-dev). Upon the start of the Cloud Foundry environment, the PaaS should be assigned enough memory with `cf dev start -m 8086`. After the login with `cf login -a api.local.pcfdev.io --skip-ssl-validation` the environment should be usable.

Deploying the microservices

Now deploy the microservices. Execute `cf push` in the sub directory `microservice-cloudfoundry-demo`. This command evaluates the file `manifest.yml`, with which the microservices for Cloud Foundry are configured. `cf push catalog` deploys a single application such as `catalog`.

```
---
memory: 750M
env:
  JBP_CONFIG_OPEN_JDK_JRE: >
   [memory_calculator:
    {memory_heuristics:
     {metaspace: 128}}]
applications:
- name: catalog
  path: .../microservice-cloudfoundry-demo-catalog-0.0.1-SNAPSHOT.jar
- name: customer
  path: .../microservice-cloudfoundry-demo-customer-0.0.1-SNAPSHOT.jar
- name: hystrix-dashboard
  path: .../microservice-cloudfoundry-demo-hystrix-dashboard-0.0.1-SNAPSHOT.jar
- name: order
  path: .../microservice-cloudfoundry-demo-order-0.0.1-SNAPSHOT.jar
- name: microservices
  memory: 128M
  path: microservices
```

In detail, the following parts of the configuration can be distinguished.

- **Line 2** ensures that each application is provided with 750 MB RAM.
- **Lines 3** – 7 change the memory distribution so that enough memory for the Java bytecode is available in the meta space of the JVM.
- **Lines 8 – 16** configure individual microservices while simultaneously the JAR files to be deployed are specified. For each of the microservices, the common settings in lines 2-7 apply. The paths to the JARs are abbreviated to increase the clarity of the listing.
- Finally, **lines 17–19** deploy the application `microservices` that displays a static HTML page with links to the microservices. In the directory `microservices` an HTML file `index.html` is stored and an empty file `Staticfile` marks the content of the directory as static web application.

Cloud Foundry uses the Java buildpack to create Docker containers, which are then started. At http://microservices.local.pcfdev.io/ (http://microservices.local.pcfdev.io/) the static web page is provided

which allows the user to use the individual microservices.

As you can see, the configuration to run the microservices on Cloud Foundry is very simple.

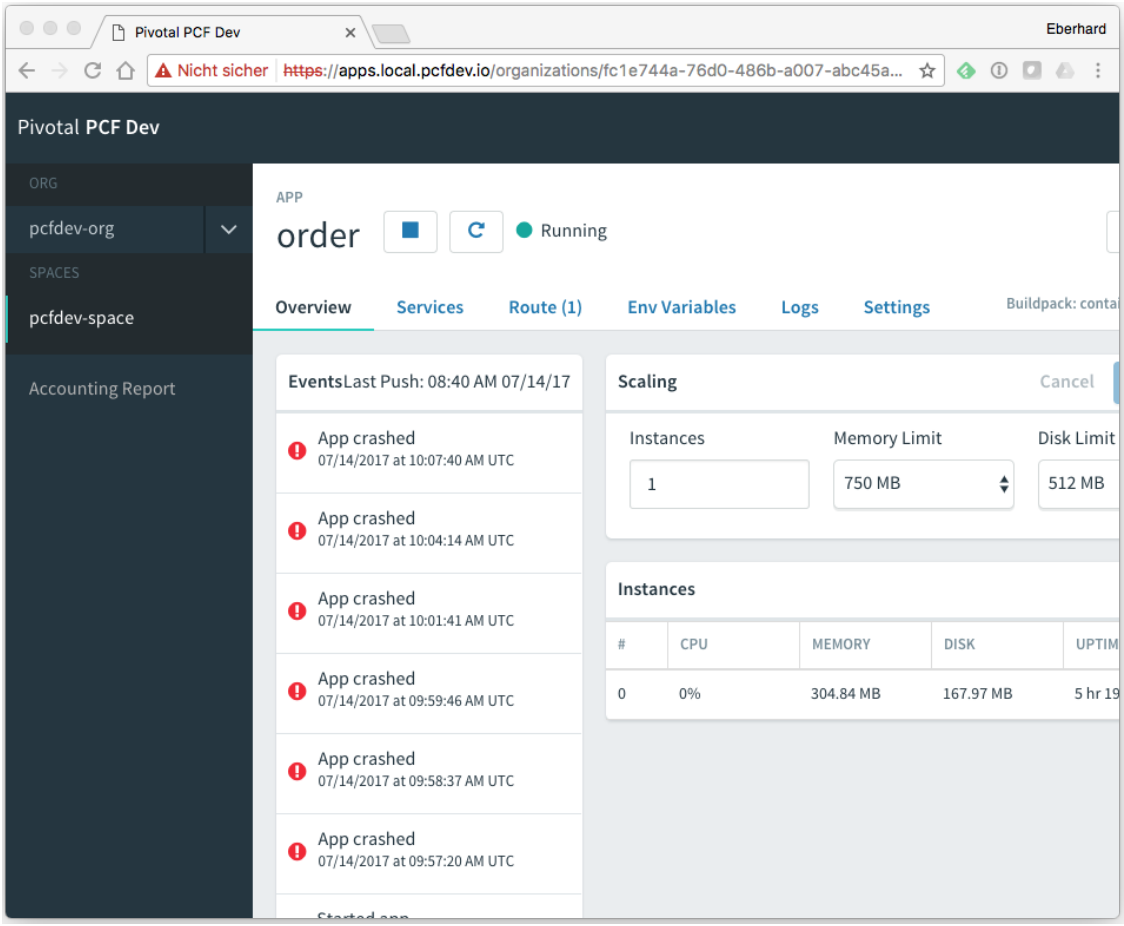
DNS for routing or service discovery

The microservices themselves have no code dependencies to Cloud Foundry. DNS is used for service discovery.

The order microservice calls the catalog and customer microservices. To do this, it uses the host names `catalog.local.pcfdev.io` and `customer.local.pcfdev.io`, which are derived from the names of the services. The `local.pcfdev.io` domain is the default but can be customized in the Cloud Foundry configuration.

`catalog.local.pcfdev.io`, `order.local.pcfdev.io`, and `customer.local.pcfdev.io` are also the hostnames used in the web browser for the links in the HTML UI of the microservices. Behind this is a routing concept in Cloud Foundry that makes the microservices accessible from outside and implements load balancing.

With `cf logs`, it is possible to have a look at the microservices logs. `cf events` returns the last entries. At https://local.pcfdev.io/ (https://local.pcfdev.io/) a dashboard with basic information about the microservices and with an overview of the logs is available, see the screenshot below.



Cloud Foundry Dashboard

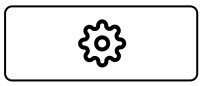
With `cf ssh catalog` the user can login to the Docker container in which the microservice `catalog` runs. This allows the user to examine the environment more closely.

Using databases and other services

It is possible to provide the microservices with additional **services** such as databases. `cf marketplace` shows the services in the marketplace. These are all services that are available in the Cloud Foundry installation.

From these services, instances can be created and made available to the

From these services, instances can be created and made available to the applications.



Example for a service from the marketplace

`p-mysql` is the name of the MySQL service provided by the local Cloud Foundry installation that can be used to run the examples. With `cf marketplace -s p-mysql` you can obtain an overview of the different offerings for the service `p-mysql`.

`cf cs p-mysql 512mb my-mysql` generates a service instance named `my-mysql` with the configuration `512mb`. The command `cf bind-service` can make the service available to an application.

With `cf ds my-mysql` the service can be deleted again.

Using services in applications

The application must be configured with the information for accessing the service. For this, Cloud Foundry uses environment variables that contain server addresses, user accounts, and passwords.

The application must read this information. There are different possibilities for this in the respective programming languages. The buildpack documentation (https://docs.run.pivotal.io/buildpacks/) contains more information.

A configuration using environment variables can also be used for settings provided during the installation of the microservice. In the `manifest.yml`, variables can be set that can be read by deployed applications.

Services for asynchronous communication

Some services add asynchronous communication to Cloud Foundry. For example, the local Cloud Foundry installation offers **RabbitMQ and Redis** as services.

These are both technologies that can send messages asynchronously between microservices. Other Cloud Foundry offerings can provide additional MOMs as services.

In the next lesson, we'll look at some recipe variations.

← Back

Cloud Foundry

Next →

Variations & Experiments

Mark as Completed

Report an Issue