Cloud

Définition ?

"… the market seems to have come to the conclusion that cloud computing has a lot in common with obscenity--you may not be able to define it, but you'll know it when you see it"

James Urquhart – The Wisdom of Clouds

Cloud origin

* Cloud computing, introduced by

* Amazon (2002), suite of cloud-based services including storage, computation and even human intelligence through the Amazon Mechanical Turk.
* 2006, Amazon launched its Elastic Compute cloud (EC2)
* was announced as "Azure" in October 2008 and was released on 1 February 2010 as Windows Azure, before being renamed to Microsoft Azure on 25 March 2014. **Google App Engine** (often referred to as **GAE** or simply **App Engine**)

Cloud origin

* Now: GAE, Azure EC2, IBM SmartCloud, Oracle Cloud, Heroku, etc.
  * Dockers, micro-services, kubernetes, qarkus, cloud foundry, Knative, etc.

Cloud features :

* new API,
* storage,
* compute,
* Scalability (long term),
* Elasticity (short term),
* etc.

Cloud origin

Les principaux fournisseurs

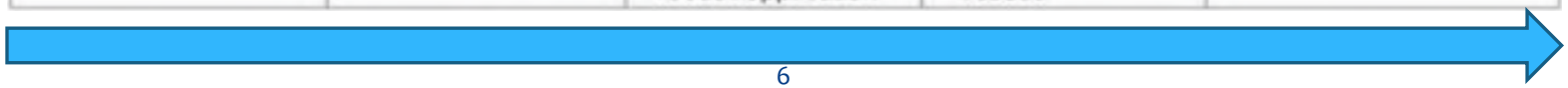
Plateforme	Fournisseur	CA en milliards $ pour 2018	Début 2019
Azure	Microsoft	+10(1)	+73% sur la croissance d'Azure par rapport à début 2018
AWS	Amazon	29-30	+ 41 % annoncés. AWS pèse 13 % du CA d'Amazon et 50 % des bénéfices !
IBM Cloud	IBM	12-13	Surveiller l'intégration de Red Hat
Cloud Platform	Google	+4	GCP doit continuer à grossir pour pouvoir prétendre concurrencer Azure et AWS
Alibaba Cloud	Alibaba	-4	Le défi est de croître en dehors de l'Asie
Oracle Cloud	Oracle	+26(2)	?

(1) Pas de montant compatible spécifique à Azure. Estimation

(2) Montant global incluant les services cloud, le support, les licences

Cloud origin

Serveurs physiques (bare metal)	VMs	Orchestration de containers Kubernetes	Plateformes cloud native CloudFoundry	Serverless KNative
Performance Maitrisé, idéal pour mainframes et architecture legacy Pas/très peu d'automatisation pour dev & ops Modèle de sécurité rigide et macro	Densité des workloads Fort bénéfice opérationnel (versions, snapshots, encapsulation, migration) Très peu de valeur pour dev Non homogène	Densité extrême Fort bénéfice opérationnel et par API / infra as code Solution à "it was working on my machine!" Idéal pour contrôler l'infrastructure sous l'application	Gains opérationnels DevOps dès les 4 facteurs apps Transformation forte vers DevOps & CI/CD avec forte productivité des Devs à l'échelle Gains sur sécurité et réseau	Productivité Dev & simplification du code maximale Idéal pour architecture orientée "event" (vs request/response) Peu adaptée à une modernization applicative massive

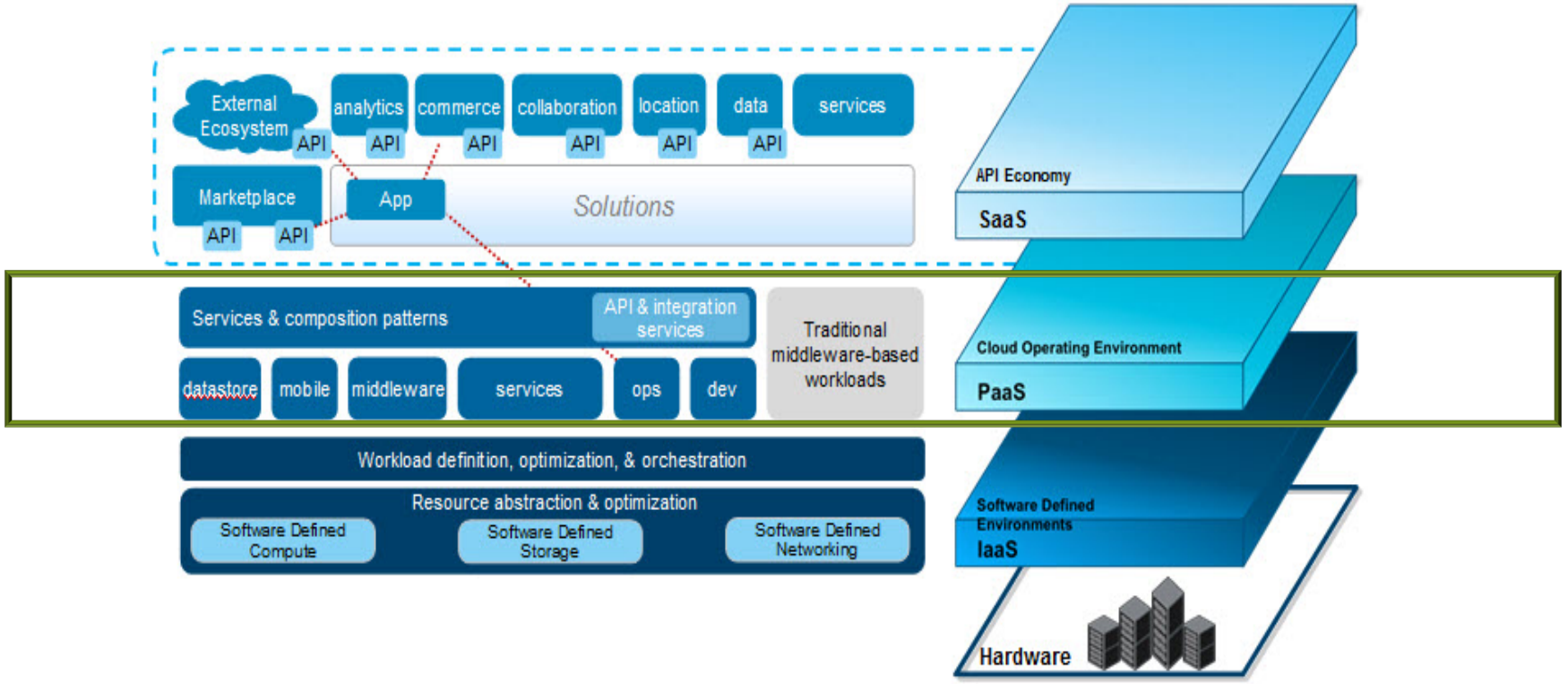


Cloud origin

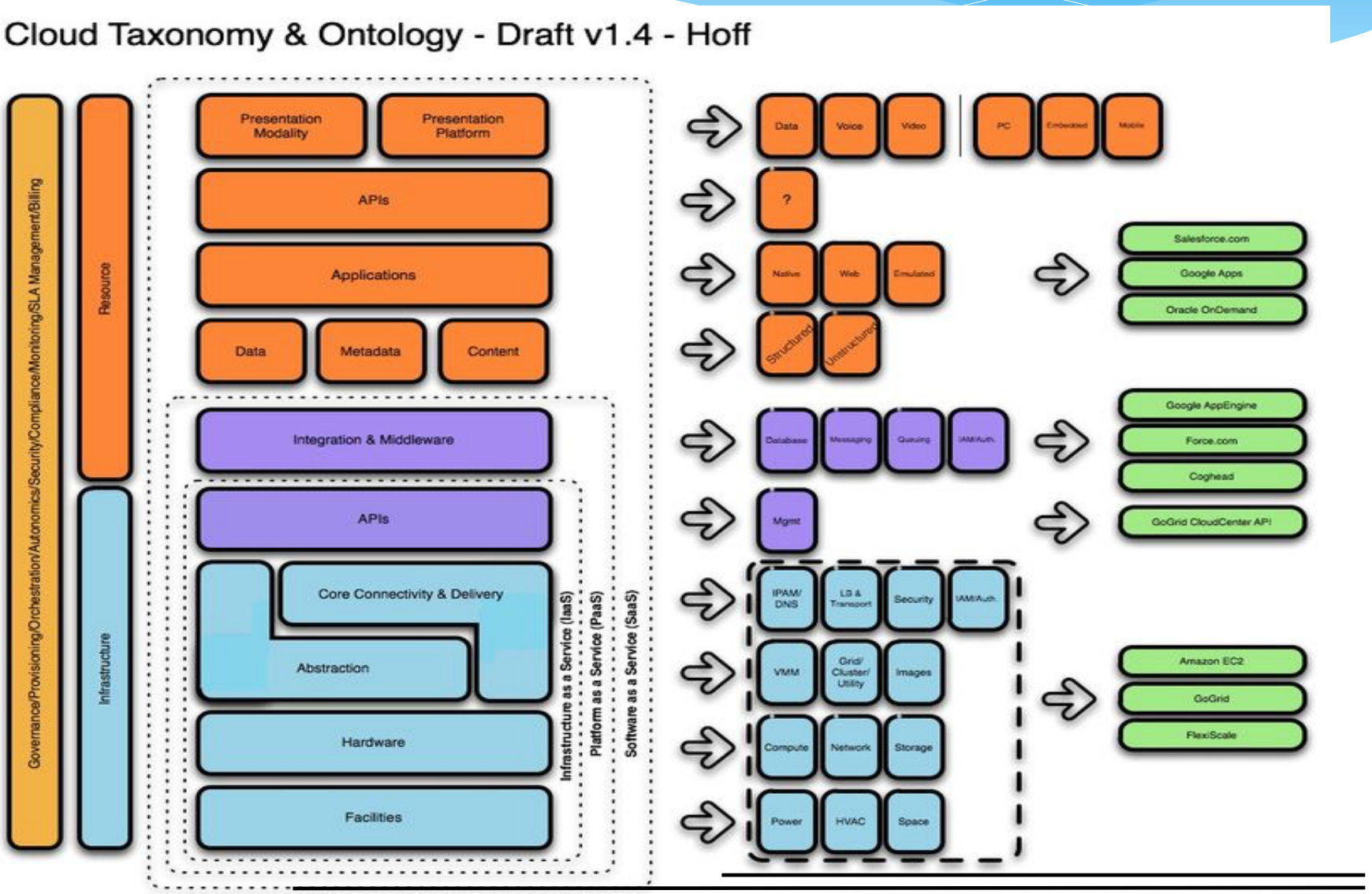
* Kubernetes: gestionnaire de conteneurs (nb instances, load balancing, deploiement, self-healing), différent de Paas

* Serverless: app. Qui dépendent fortement de services tiers (cloud ou autre, pour BD auth, etc.) différent de « pas de serveur »

Architecture



Architecture



Architecture

- IaaS: Infrastructure as a service
  - Virtualisation d'OS
  - Le hardware est extensible et non géré
  - Ex: amazon

- PaaS : platform as a service
  - Déploiement d'application dans env. extensible
  - OS+serveur d'application (glassfish, jboss, etc.) + couche persistance + API
  - Ex: GAE, Windows Azure, openshift,etc.

- SaaS : software as a service
  - Service proposés aux clients

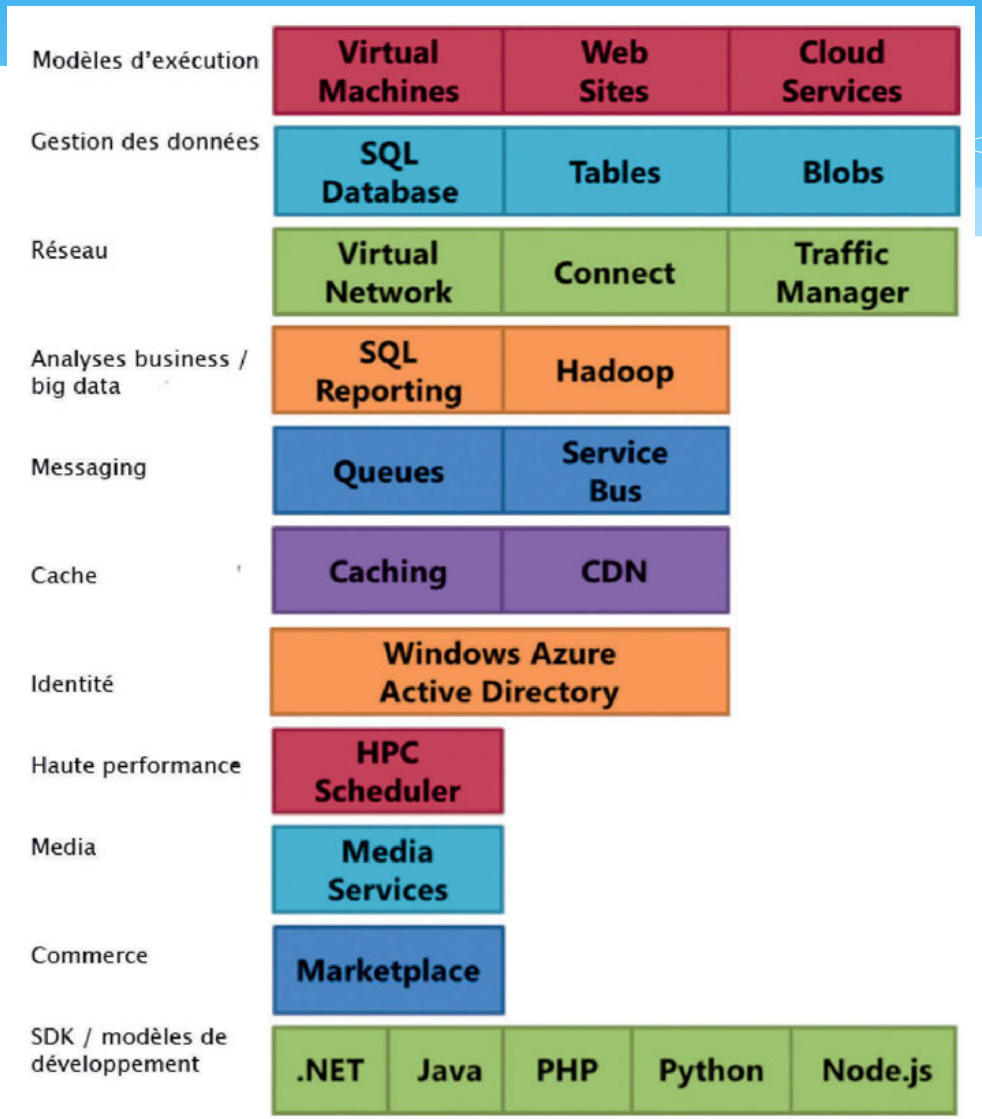
Types d'architectures

- Cloud public: solutions de stockage et applications offertes au public par accès via Internet (Amazon, Microsoft, Google)
- Cloud communautaire: infrastructure partagée entre organisation. Gestion du Cloud en interne ou par tierce partie. Travail collaboratif
- Cloud hybride: composé de >1 clouds privés, communautaires ou privés. Offre l'avantage de promouvoir plusieurs modèles de déploiements. Infrastructure interne+ externe => utilisation immédiate et locale et non dépendance à Internet. Evolutif en terme de taille via l'architecture externe,
- Cloud privé: infrastructure privée uniquement à une seule organisation. Nécessité de gérer la partie infrastructure: virtualiser environnement Business, réévaluer les ressources existantes, les problèmes de sécurité à chaque modification. Perte des avantages liés au Clouds; flexibilité, évolutivité

Aperçu de Windows Azure

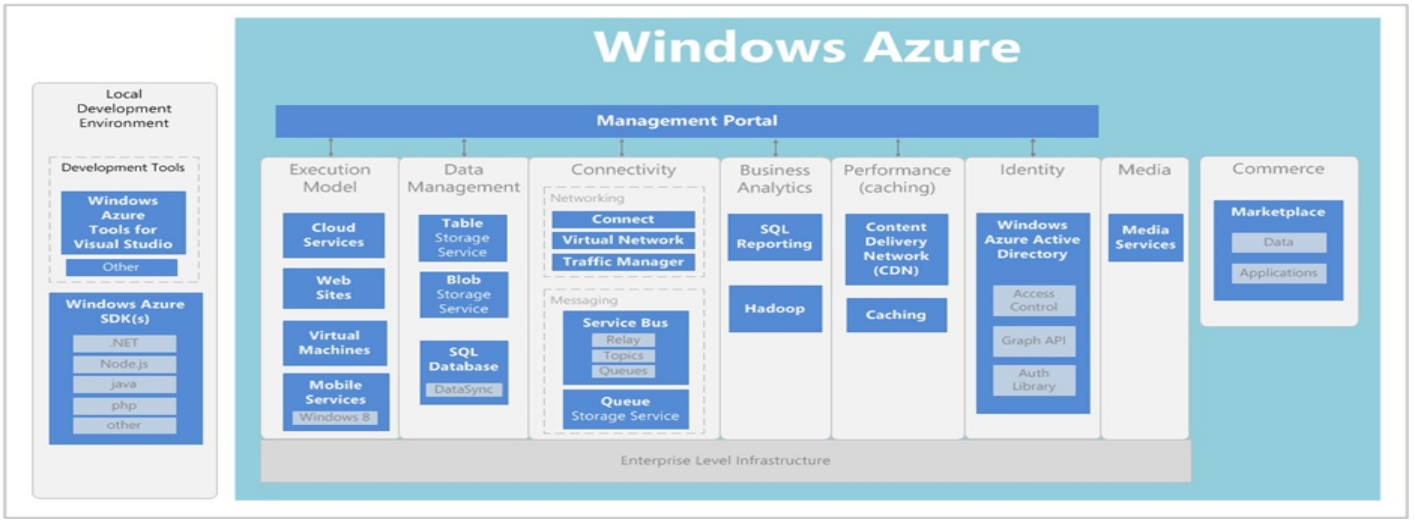
Mais vraiment un petit aperçu

Fonctionnalités



PaaS Windows Azure

* Partie IaaS et SaaS non traitée (mais vous pouvez louer des VMs)
* PaaS avec types de services proposés très large:
  * Langages:
    * C# VB bien sur
    * Python
    * Java avec un serveur d'application tel que tomcat ou autre
    * PHP (voir en fin)
    * Ruby,etc.
  * Types d'applications :
    * Services Web SOAP, REST, plain/text,
    * Sites web, applications en worker role
  * Plusieurs types de services à louer (service bus et autres)
  * Consoles d'administration, analyse de performance, etc.

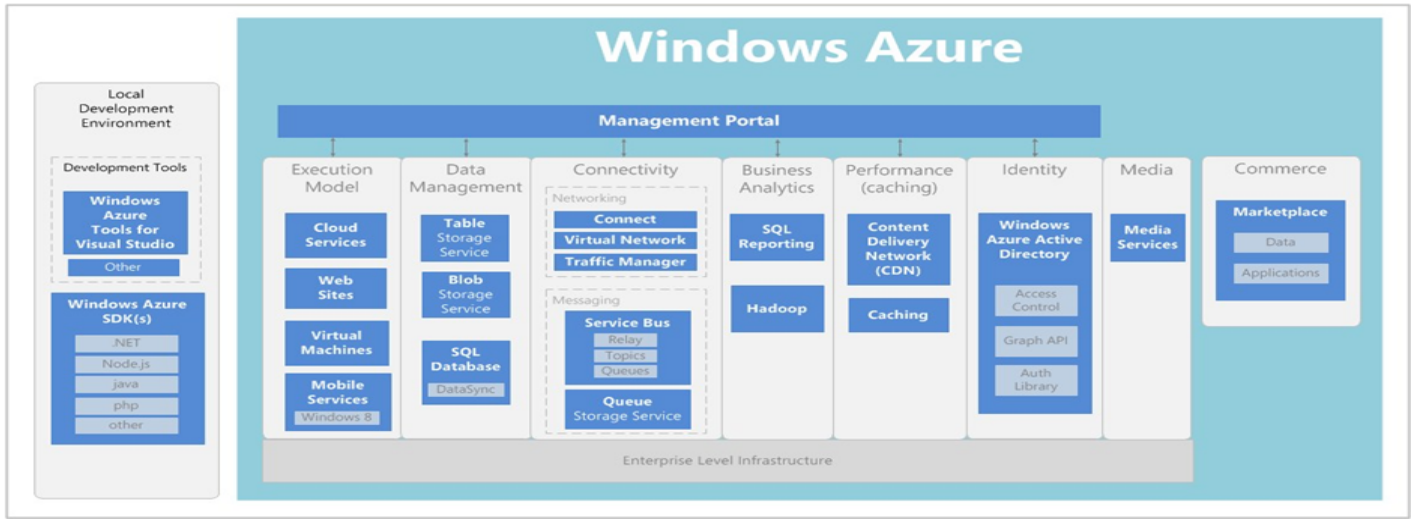


Service Bus permet d'exposer des services simplement, le bus se chargeant du routage des requêtes vers le service concerné.

Acces control: permet de gérer l'accès au Service Bus suivant des mécanismes standards tels que OAuth et les Simple Web Tokens (SWT) pour les services REST, ou encore des mécanismes à base de revendications de type SAML, WS-Federation et WS-Trust pour l'accès à des services SOAP

Composite App Service et **Composition Model** fournissent un environnement de développement pour faciliter la création, la gestion et le déploiement d'applications composites.

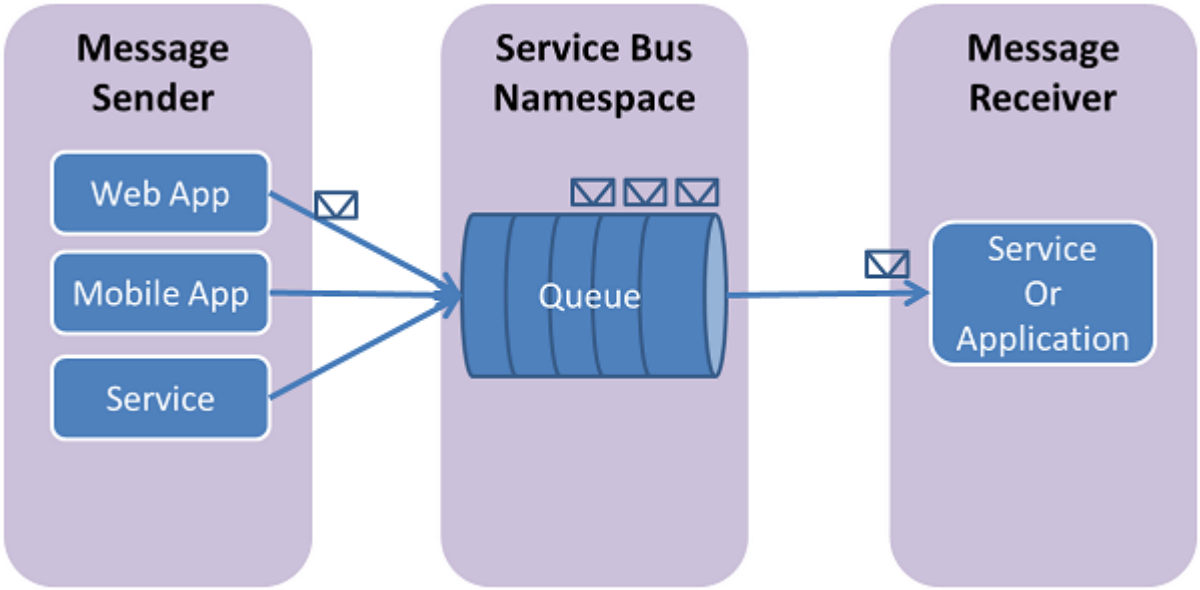
Cloud services : SOAP Rest web services, web role, worker roles



Blobs: blob files allowing to store files or meta-data
Table: non relational tables, fulfilled with entities,
Queue asynchronous FIFO between apps
Drive manage and configure vituel disks

PaaS Windows Azure

Les possibilités offertes par ServiceBus sont nombreuses:
Ex: Utiliser une FIFO pour la réception de messages

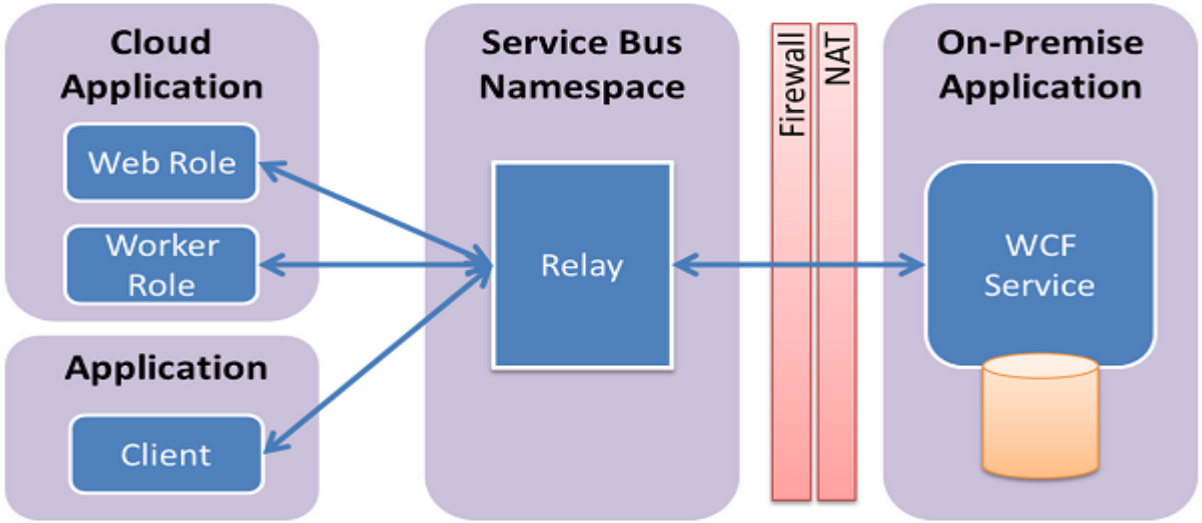


Permet un envoi de messages de façon asynchrone: lecture asynchrone, le sender n'a pas à attendre une réponse.

PaaS Windows Azure

ServiceBus, d'autres possibilités:

Relay messaging: Utilisation d'un relay entre entités
Possibilité de construire des applications hybrides déployées dans Windows Azure ou autre
Sécurisation de l'ensemble via le relay
https://docs.microsoft.com/fr-fr/azure/service-bus-relay/service-bus-relay-tutorial



PaaS Windows Azure

ServiceBus, d'autres possibilités:

Brokered messaging: stockage intermédiaire de haut capacité et durable des messages peuvent être stockés et traités

les deux extrémités peuvent être complétement hétérogène en terme de puissance

Elles peuvent être en ligne ou non

PaaS Windows Azure

* D'autres paradigmes (Windows Azure)
  * Rôles d'applications avec commutation de rôles:
    * *web role : service ou appli Web*
    * *worker role :* démons persistant qui peut recevoir des données d'une autre appli
  * Chaque application est un composant (couche appFabric assure la connectivité)

PaaS Windows Azure

* D'autres paradigmes (Web et worker roles)

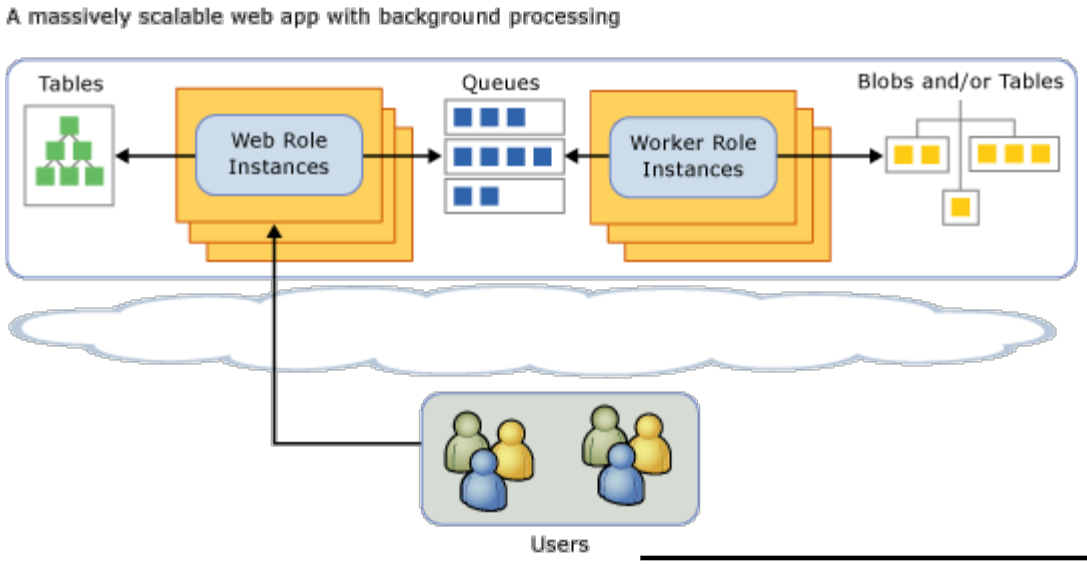
* Web Role:
  * Application pour Requètes / réponses sur HTTP (pages, WCF Web service configurés par basichttpbinding, etc.)

* Worker role:
  * Application de type service fonctionnant en tache de fond. N'accepte pas de requète de l'extérieur

* Web roles et worker roles peuvent dialoguer ensemble via des objets Queues: classiquement worker produit des données, le web role les lits à la demande de l'utilisateur et produit un affichage

PaaS Windows Azure

* D'autres paradigmes (Web et worker roles)
* Une application peut changer d'état!
* Un Web service a généralement un Web role mais peut également être implanté en worker role
* Web roles et worker roles peuvent être placés dans des VM roles (distribution manuelle sur des VM différentes)



PaaS Windows Azure

* Sécurité :
  * Par AppFabric Access control service => propose WS-Trust, HTTPS, token,…
* Envoi multicast
* Buffer partagé de type fifo : pour effectuer des partages de données rapidement

PaaS Windows Azure

- Implantation de services

- Utilisation de WCF services:

  - WCF: Windows Communication Foundation: framework pour création d'applications orientées service sur HTTP
  - Utilisation très simple
  - Envois asynchrones possible
  - Exposition en plain/text, Rest, SOAP, etc.

PaaS Windows Azure

Ex en c# (new projet / WCF / WCF library)
Définition de l'interface du service (Contrat et OperationContrat)
=> Fichier nom_de_la_classe

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;

namespace MathsLibrary
{
    [ServiceContract]
    public interface IMathsOperations
    {
        [OperationContract]
        int Add(int num1, int num2);
        [OperationContract]
        int Multiply(int num1, int num2);
    }
}
```

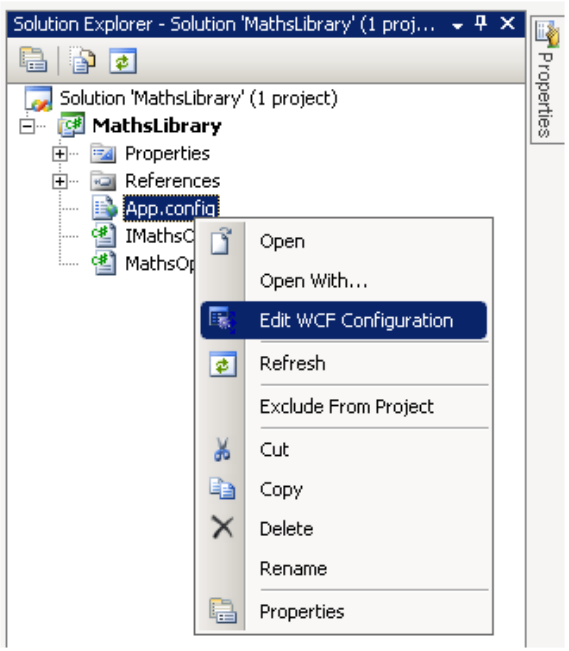
PaaS Windows Azure

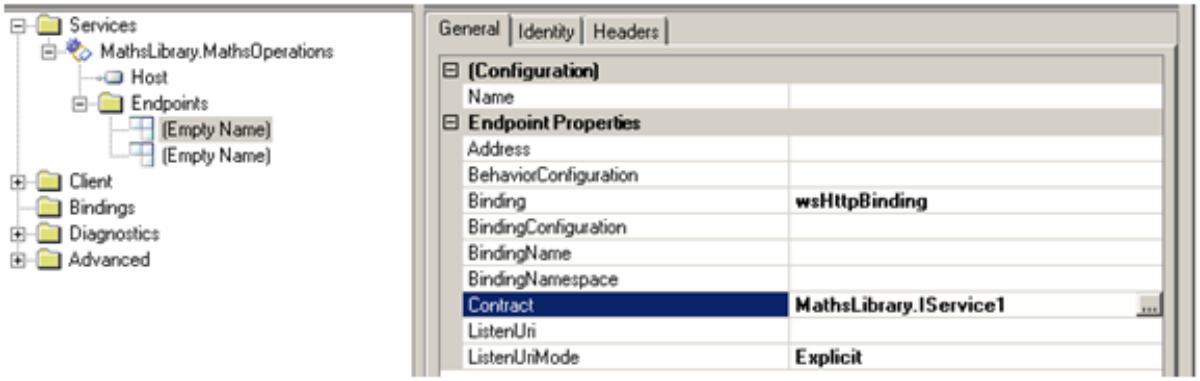
Ex en c# (new projet / WCF / WCF library)
Implantation du service
=> Fichier nom_de_la_classe

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MathsLibrary
{
    public class MathsOperations : IMathsOperations
    {
        #region IMathsOperations Members

        public int Add(int num1, int num2)
        {
            return num1 + num2;
        }

        public int Multiply(int num1, int num2)
        {
            return num1 * num2;
        }

        #endregion
    }
}
```

PaaS Windows Azure

Ex en c# (new projet / WCF / WCF library)
Configuration du contrat:





* Exemple: *https://docs.microsoft.com/fr-fr/visualstudio/data-tools/windows-communication-foundation-services-and-wcf-data-services-in-visual-studio?view=vs-2019*
* Création du service :
* Définition de l'interface et définition du code

```C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WCFServiceWebRole1
{

    [ServiceContract]
    public interface IService1
    {

        [OperationContract]
        string GetHello();
    }

}
```

```C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace WCFServiceWebRole1
{

    public class Service1 : IService1
    {
        public string GetHello()
        {
            return "Hello from my WCF service in Windows
Azure!";
        }
    }
}
```

* Exemple:

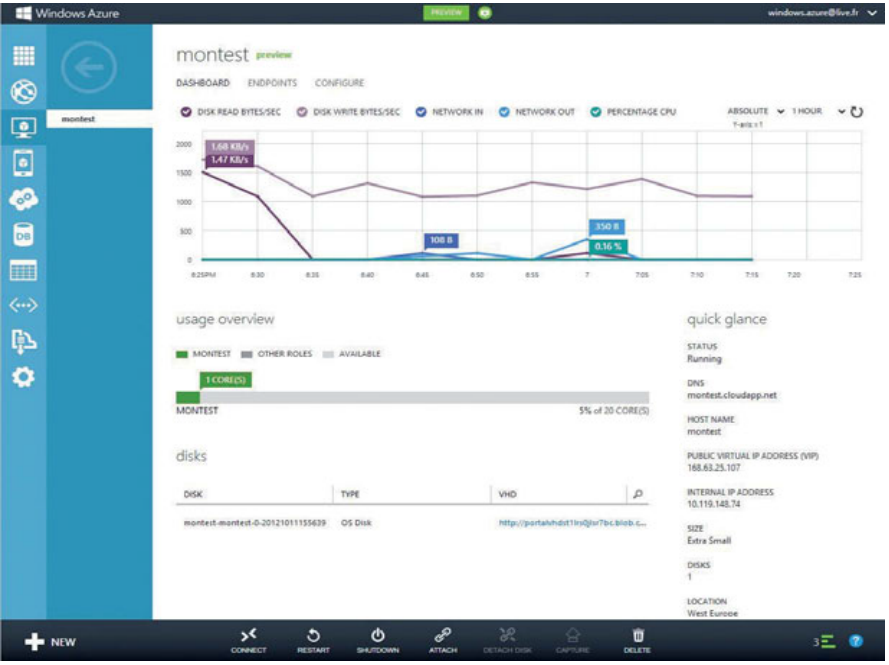
* Création Client : Ajout d'une référence (comme sur Netbeans), génération de squelettes et complétion du code généré dans Program.cs

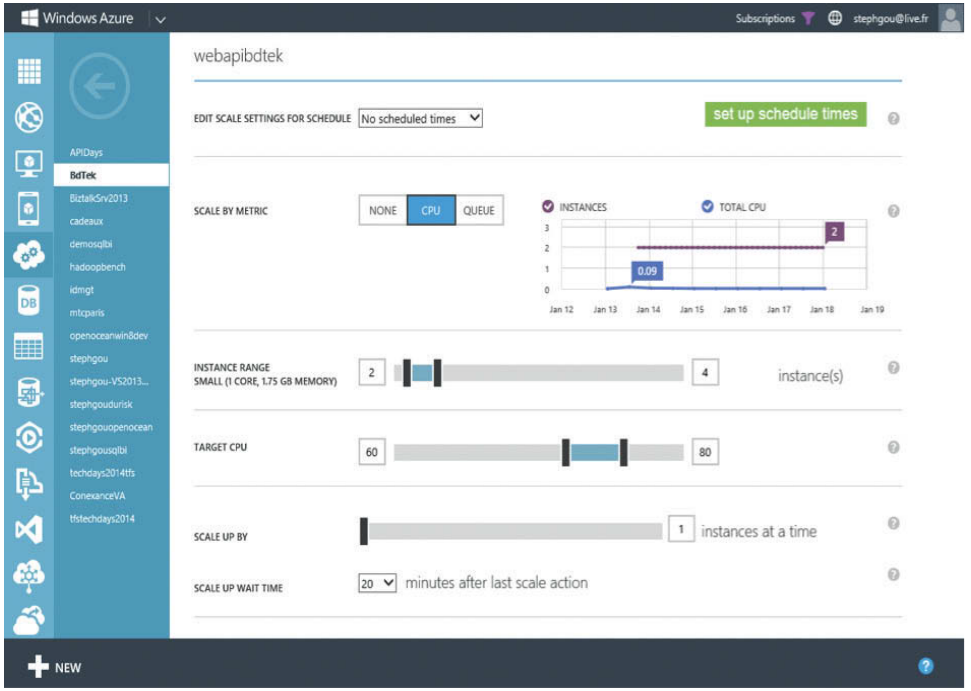
```C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

// Modify the following line to use your project name
// if your project is not named Console1Application1.
using ConsoleApplication1.ServiceReference1;

// Modify the following line to use your project name
// if your project is not named Console1Application1.
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Service1Client client = null;
```

```
            try
            {
                client = new Service1Client();

                GetHelloRequest request = new
GetHelloRequest();
                GetHelloResponse response;

                response = client.GetHello(request);

                Console.WriteLine("The WCF service called
returned: '{0}'",
                                  response.GetHelloResult);
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception encounter: {0}",
                                  e.Message);
            }
            finally
            {
                if (null != client)
                {
                    client.Close();
                }
            }
        }
    }
}
```

Appel

Interface de configuration

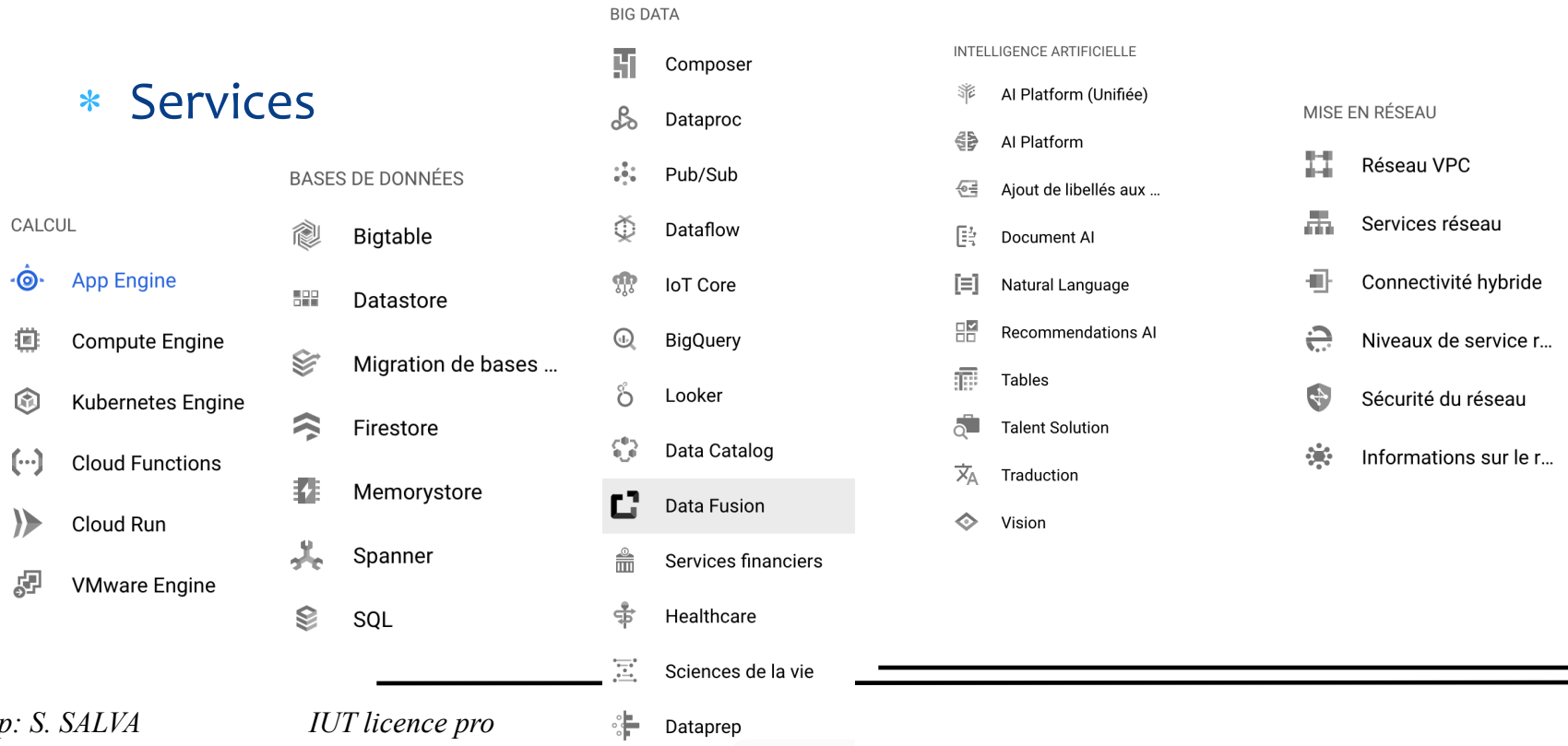




Introduction à GAE (google App Engine)

PaaS Google App engine

* Service d'exécution en Python, Java, Go
* Actuellement gratuit pour une appli avec accès < 5millions/mois
* Services



PaaS Google App engine

* Types d'applications Java :
  * servlet/JSP, services Web en Rest, Java 8 ou 11
  * Plusieurs librairies Java supportées mais pas toutes
  * Implantation Services Rest possible (Springboot, jersey , Micronaut, etc.)
  * Mais aussi python, ruby, php, node.js Go, etc.

PaaS Google App engine

* Des limitations:
* Pas de connexion TCP
  * Connexion URLConnect pour effectuer des appels entre pl. servlets
* Pas de processus
* Timeouts limités
* Env. standard limité (version gratuite)
* Env flexible: instances d'application s'exécutent dans des conteneurs Docker sur des (VM) Compute Engine.

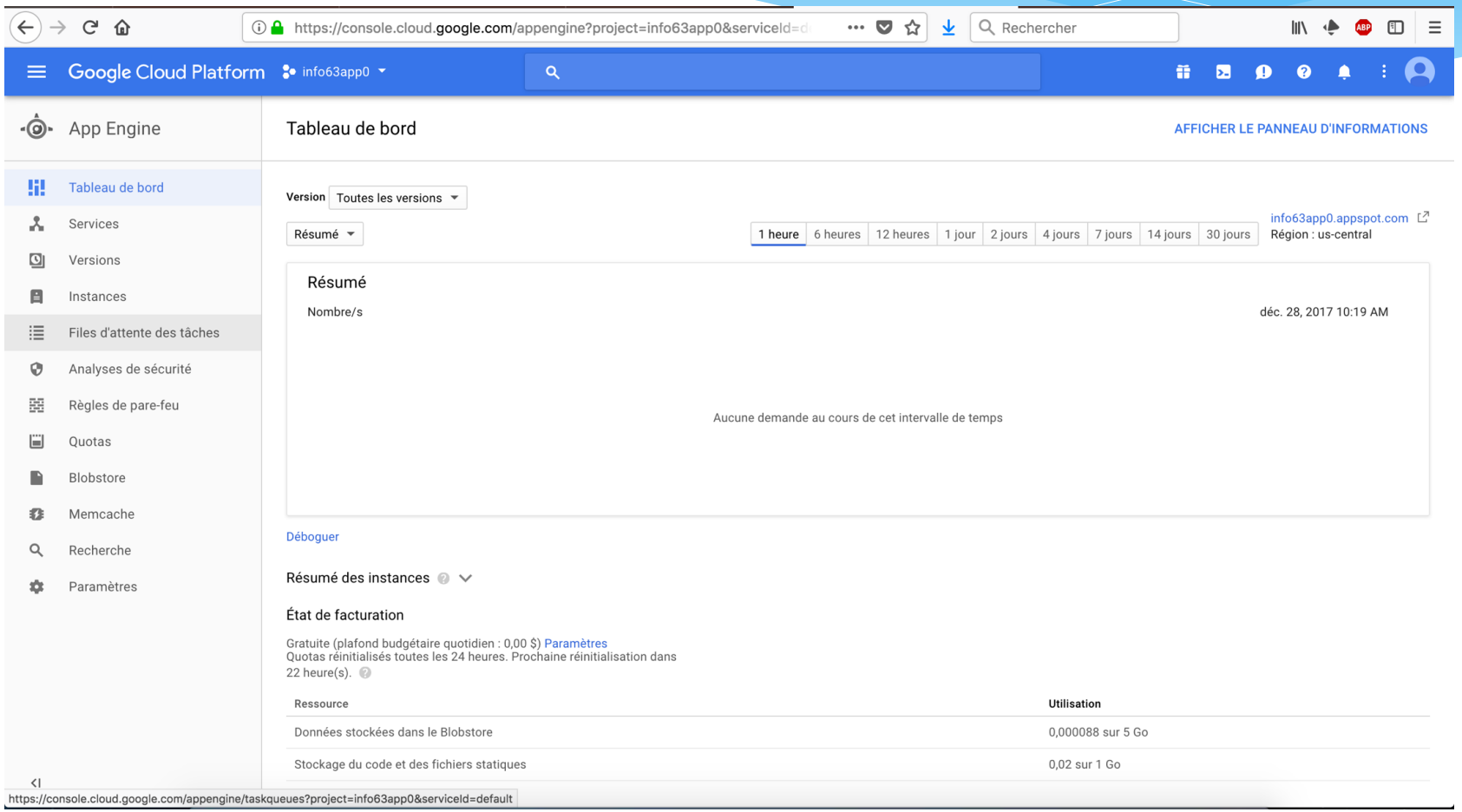
PaaS Google App engine

* Installation avec Java (Google Cloud Platform)
  * https://cloud.google.com/appengine/docs/standard/java11/setting-up-environment

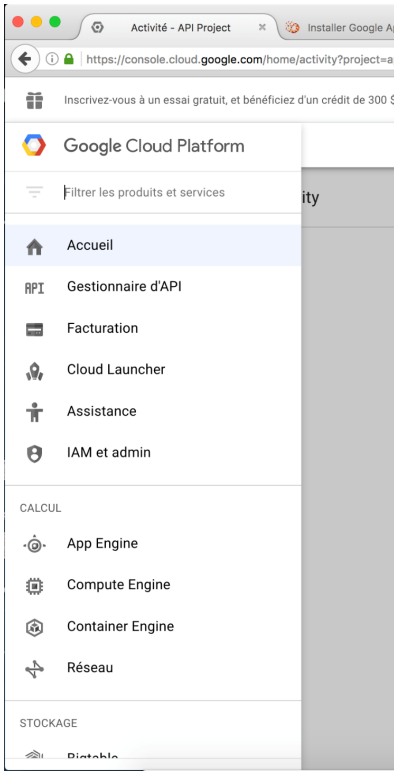
* Créer une application en console avec mvn et gcloud, et la déployer
  * https://cloud.google.com/appengine/docs/standard/java11/building-app/creating-project

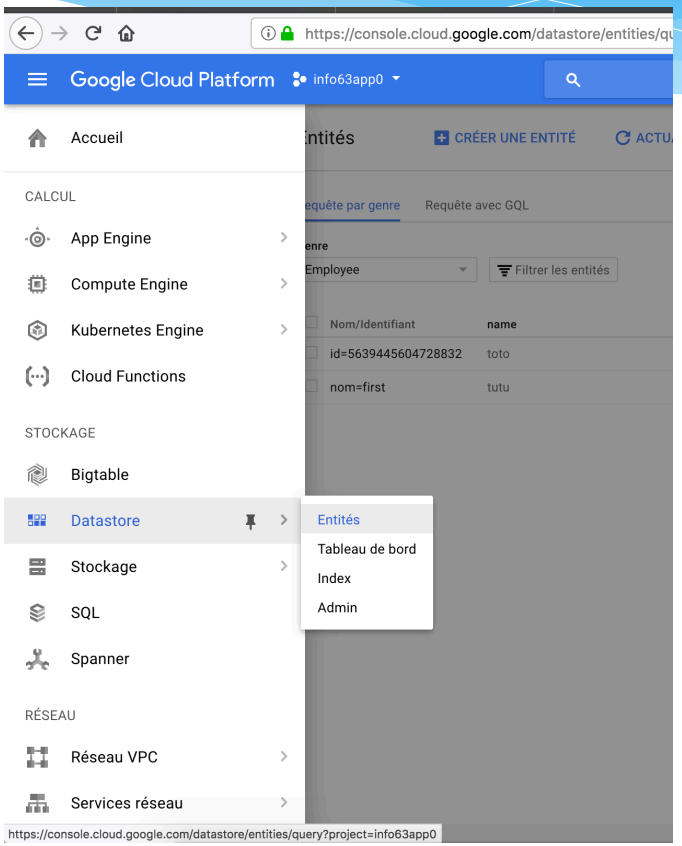
* Pour déployer une application web, il est nécessaire de créer une application dans la console google https://console.cloud.google.com/appengine => fournir un id

GAE, création d' une application



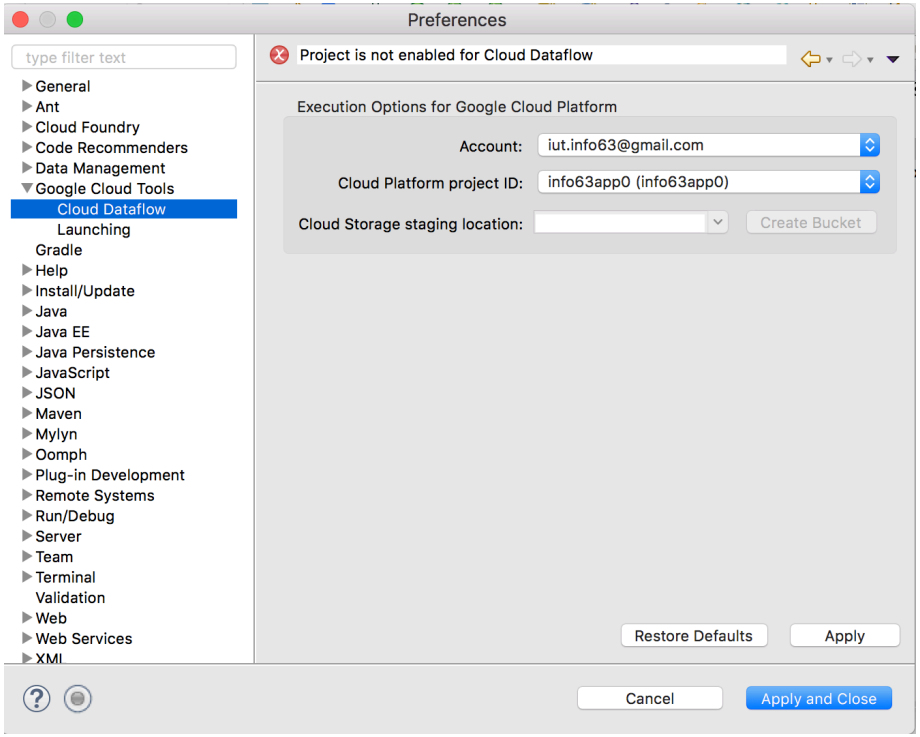
GAE, portail



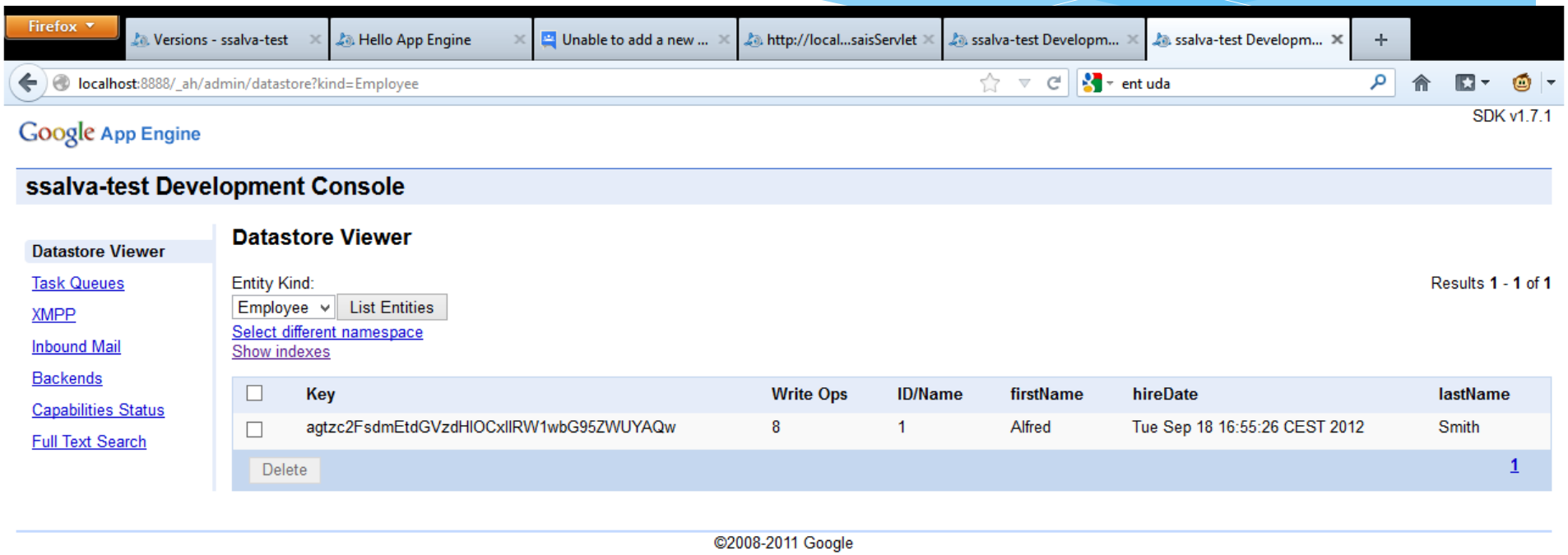


Appel d'une application web sur votre env:
http://version.id_app.appspot.com/nom_servlet ou ws
ex: http://1-dot-ssalva-test.appspot.com/Service/rest/gae_restws

GAE, plugin Eclipse



GAE, test de l'application en local



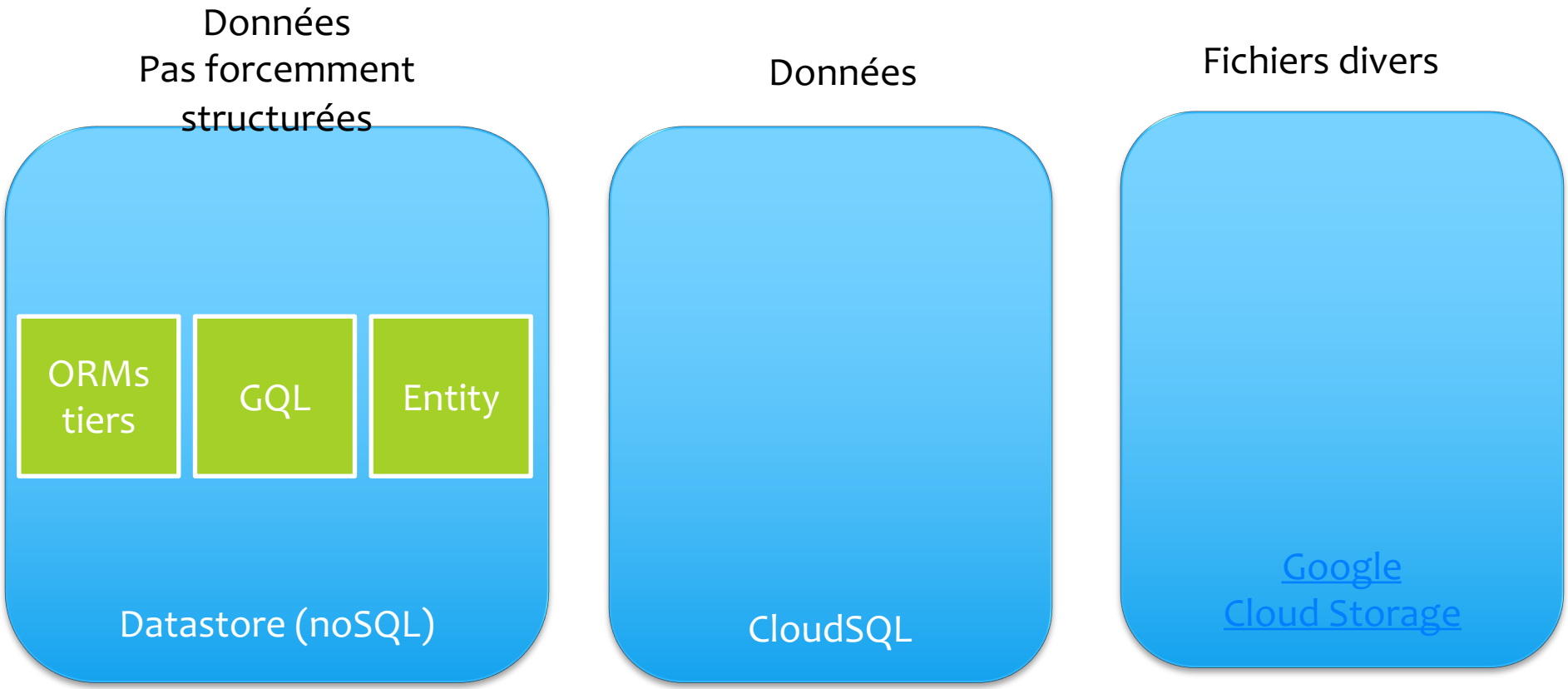
http://localhost:8888/_ah/admin => console d'administration locale (affichage du blob de données, des services, etc.)

http://localhost:8888/mon_servlet ou service

JAVA 8 ? (java 13 KO, passer par Jetty)

PaaS Google App engine

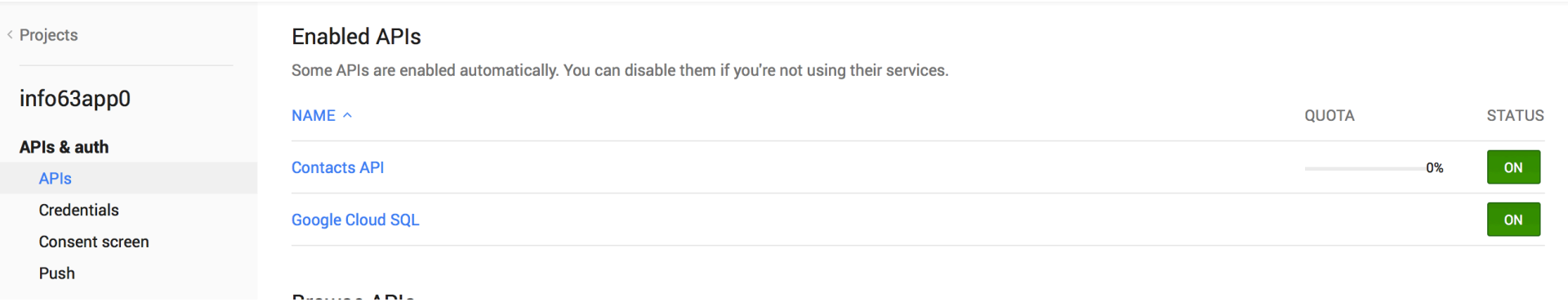
* Stockage



PaaS Google App engine

* Besoin d'activer des apis
* Ex: Stockage dans CloudSQL = base Mysql

* Besoin d'activer l'api pour une application dans



PaaS Google App engine

* Stockage dans CloudSQL

1. Création de tables

2. utilisation de jdbc:odbc classique

PaaS Google App engine

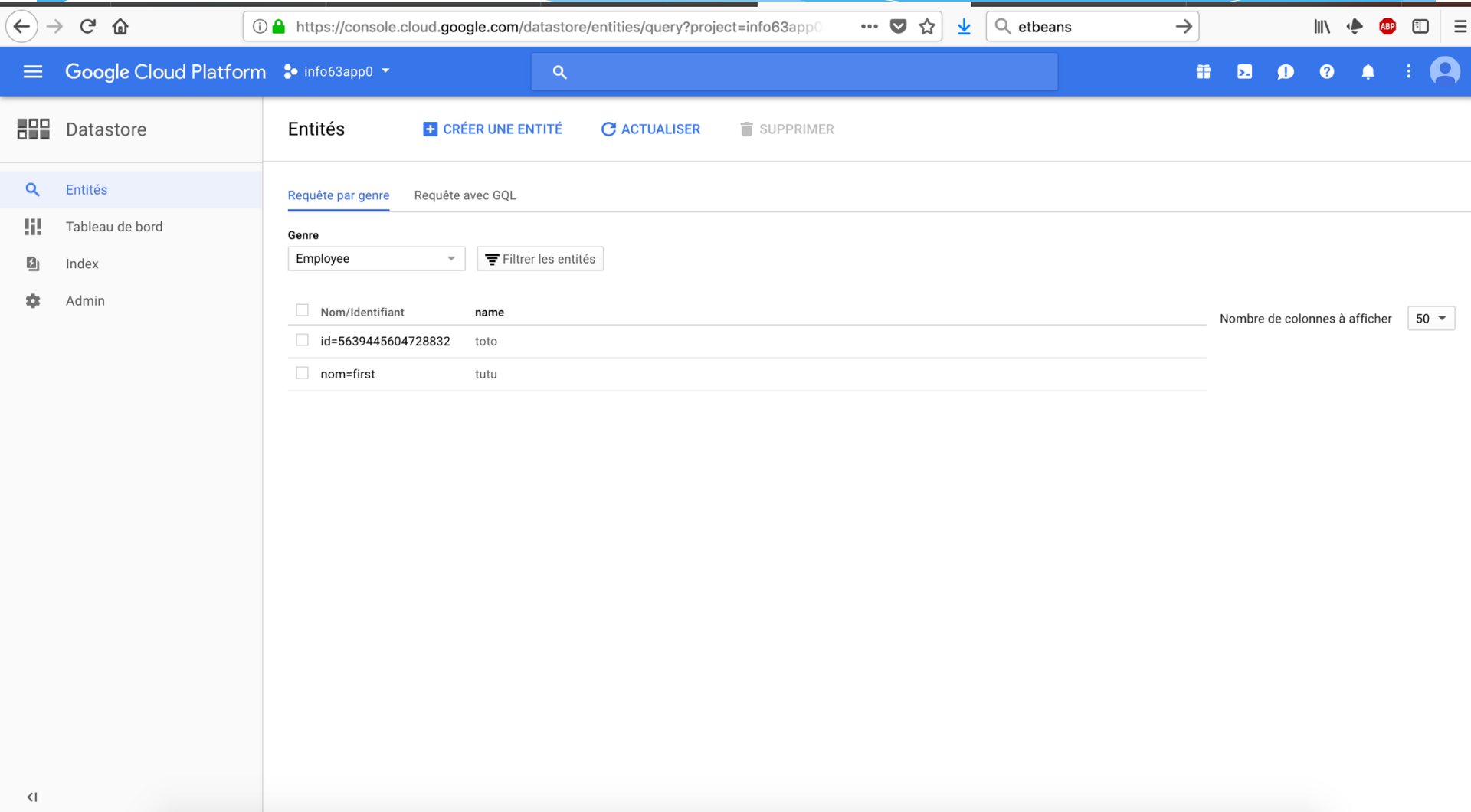
Utilisation de JDBC (avec une Servlet)

```
Public class GuestbookServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse resp) {
      try {

         Connection c = null;
         DriverManager.registerDriver(new AppEngineDriver());
         c = DriverManager.getConnection("jdbc:google:rdbms://simple-it.fr:testcloudsql:test-cloud-sql/guestbook");

ResultSet resultats = c.createStatement().executeQuery("SELECT name, message FROM messages ORDER BY id DESC LIMIT 20");
         req.setAttribute("messages", resultats);
         this.getServletContext().getRequestDispatcher("/WEB-INF/guestbook.jsp").forward(req, resp);

    } catch (Exception e) {
         e.printStackTrace();
     } }
```

PaaS Google App engine

* Datastore
  * MAP (clé valeur)
  * Plusieurs accès possibles (entity,Rest, …)
* Persistance bas niveau via des objets Entity
  * Génial pour stockage
  * Limité pour effectuer des requètes (requètes par clé ou id)
  * Possibilité d'imbriquer des entités ensemble
  * https://cloud.google.com/datastore/docs/concepts/entities

PaaS Google App engine



PaaS Google App engine

* Datastore identités:
* gcloud auth application-default login
  * permet de donner votre identité pour accès au datastore (et au reste?)
  * A utiliser pour des tests

* sinon génération d'un fichier .json qui doit être dans votre env. de développement (export)
  * https://cloud.google.com/docs/authentication/getting-started

PaaS Google App engine

* Datastore, mode bas niveau avec lib. google-cloud-datastore

* CRUD avec mot clés get, put, delete update

Création d'une instance Entity

https://cloud.google.com/datastore/docs/reference/libraries

PaaS Google App engine

* Datastore

```
Datastore datastore
     =DatastoreOptions.newBuilder().setProjectId("info63appo").build().getService();
String kind = "Employee";
String name = "first";
// The Cloud Datastore key for the new entity
Key taskKey = datastore.newKeyFactory().setKind(kind).newKey(name);

// Prepares the new entity
Entity employee = Entity.newBuilder(taskKey)
                    .set("name",  t)
                    .build();

        // Saves the entity
        datastore.put(employee);
```

PaaS Google App engine

* Datastore

```
//Retrieve entity
Entity retrieved = datastore.get(taskKey);

System.out.printf("Retrieved %s: %s%n", taskKey.getName(),
retrieved.getString("name"));
```

PaaS Google App engine

* Datastore

Récupération avec requêtes

```
Query<Entity> query =
    Query.newEntityQueryBuilder().setKind("Employee").setOrderBy(OrderBy.as
    c("name")).build();
Iterator<Entity> it=datastore.run(query);
String ret="<html><body>";

while (it.hasNext()) {
    Entity e = it.next();ret+="iteration";
            ret+=e.getString("name")+"</br>";                    }
```

PaaS Google App engine

* Datastore

Récupération avec requêtes au exemple avec GQL (voir doc)
https://cloud.google.com/datastore/docs/reference/gql_reference

```
String kind = "my_kind"; String gqlQuery = "select * from " + kind; Query<Entity>
    query = Query.newGqlQueryBuilder(Query.ResultType.ENTITY,
    gqlQuery).build(); QueryResults<Entity> results = datastore.run(query); // Use
    results          }
```

PaaS Google App engine

* Datastore

* Objectify (https://github.com/objectify/objectify)
* Java data access API specifically designed for the Google App Engine datastore.

PaaS Google App engine

* Datastore

* Objectify (https://github.com/objectify/objectify)

Création d'une entité dans une classe avec annotations:

```
@Entity class Car {
@Id String vin; // Can be Long, long, or String
String color; }
```

Manipulation

```
ofy().save().entity(new Car("123123", "red")).now();
Car c = ofy().load().type(Car.class).id("123123").now();
// ou .filter("color", "red").
ofy().delete().entity(c);
```

PaaS Google App engine

* Datastore

* Objectify (https://github.com/objectify/objectify)

Besoin d'enregistrer les Entités à la connexion avec le Datastore (voir doc et TP)

Il faut déclarer une servletcontext (pas toujours simple)

```
public class YourBootstrapper implements ServletContextListener { public
    void contextInitialized(ServletContextEvent event) { ObjectifyService.init();
    ObjectifyService.register(YourEntity.class); // etc... } }
```

PaaS Google App engine

Déploiement d'un service Rest Jersey 2 (eclipse):

1. Créer un projet Google Standard App Engine,
2. Implanter un WS (je recommande MVN)
3. Modification du Pom.xml (pour jetty, persistence, Jaxb, etc.)

```
package wsrest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
 @Path("/helloworld")
public class HelloWorldResources {
@GET
@Produces("text/plain")
public String getClichedMessage() {                    return "Hello World";
            }}
```

PaaS Google App engine

4. Modifier Web.xml

```
<servlet>
  <servlet-name>myrest</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-
class>
  <init-param>
   <param-name>jersey.config.server.provider.packages</param-name>
   <param-value>wsrest</param-value>
  </init-param>
  <init-param>
   <param-name>unit:WidgetPU</param-name>
   <param-value>persistence/widget</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>myrest</servlet-name>
  <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
```

Nom du service

Nom du package englobant le SW

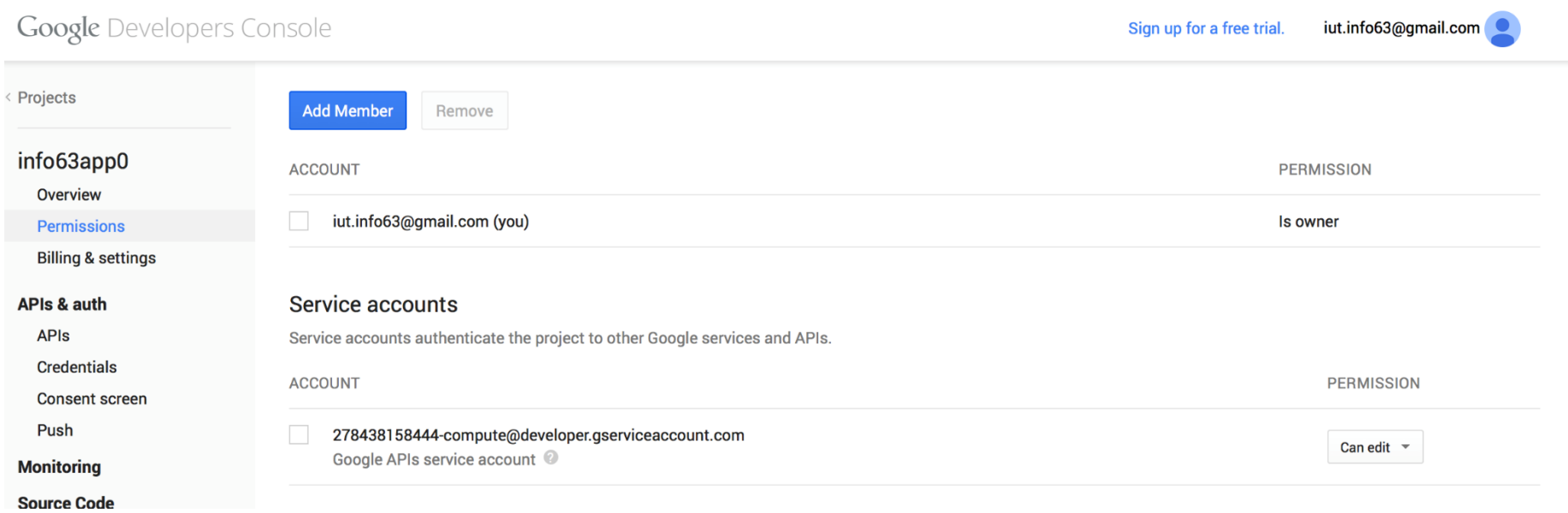
PaaS Google App engine

Déploiement d'un service Rest Jersey 2 sans IDE
https://cloud.google.com/java/getting-started/hello-world?hl=fr

1. Utiliser Maven pour créer un projet Jersey (voir doc jersey)
2. Modifier le pom.xml pour ajouter Jetty
3. Créer / utiliser une application GAE avec le client gcloud
4. Se logger avec Gloud *gcloud auth application-default login*
5. Créer un fichier app.yaml (fichier de deployement) dans src/java
6. Déployer : *gcloud config set project YOUR_PROJECT_ID mvn appengine:deploy*

GAE authentification

* Authentification : Donnez les permissions à vos applications



GAE sécurité

* https://cloud.google.com/appengine/docs/standard/java11/application-security

  * Passer à HTTPS
  * Limiter les accès
  * creer un parefeu (?)
  * Utiliser Oauth 2 ou basicAuth (voir TP)

Introduction a Heroku

Présentation héroku

* Offre PaaS (repose sur AWS) depuis 2007
* Langages:
  * Java, PHP, Ruby, Go, Scala, python, node.js
  * Des addons (redis, mongodb, etc.)
  * https://addons.heroku.com/
* Compte gratuit avec 1 dyno et accès BD (postgresql) <=10000 lignes

Présentation héroku

* A base de Git
  * Push à partir de dépots locaux ou Github
  * Utilisation de Maven pour gestion des dépendances

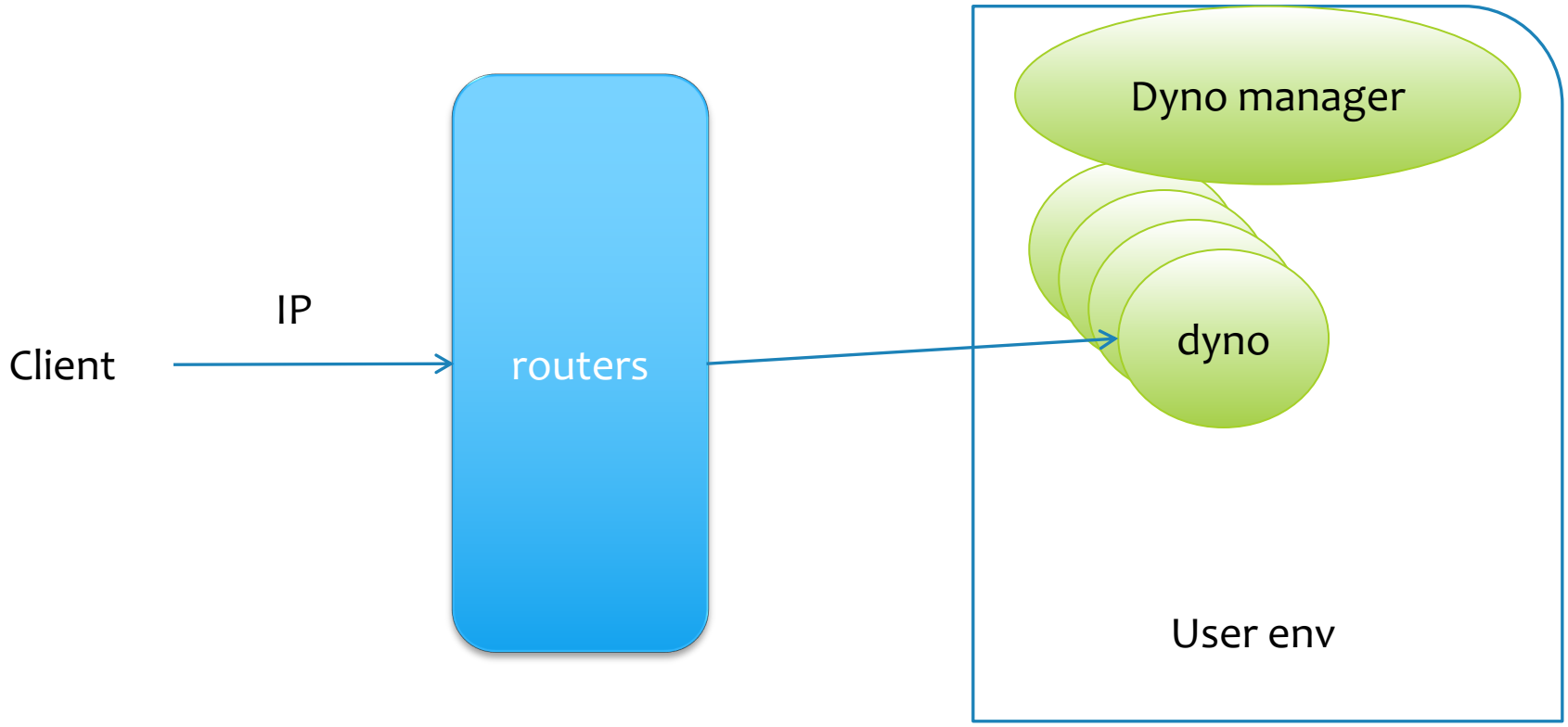
* A base de Debian,
  * Se contrôle via ligne de commande
  * Prix se calcule sur le nb de dyno (container debian) + addons
* (mais aussi dropbox, travis, etc.)

Présentation héroku

* dyno = container à base de cedar (ubuntu)
  * Exécute une seule commande à la fois (1 instance de serveur)

* types de dyno:
  * web,
  * worker (background)
  * one-off dyno: dyno temporaire pour tâches d'admin (migration etc.) (ex: heroku run bash)

Présentation héroku



Héroku

* Prérequis (pour ce cours au moins):
* Maven
  * Gest. De projet et de dépendances
  * Pom.xml-> décrit les deps (mvn clean install les télécharge et les installe)
* Git
  * Add, commit et push
* Heroku toolset

Héroku

* Gestion des dynos
* https://devcenter.heroku.com/articles/dynos

* heroku ps:scale web=X , X nb d'instances
  * pour 1 , 2 -> processus mis en veille après 1 heure

* heroku ps

Héroku, applications Java

* https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/getting-started.html#heroku-webapp

* Principe:
  1. Maven -> crée une application Web (et les tests), télécharge les deps
  2. Heroku create -> crée une appli sur Heroku
  3. Git add, git commit -> crée un dépot local
  4. Git push -> lance les tests, upload l'application, la compile, la déploie

Héroku, applications Java

* **Gestion des dépendances dans pom.xml**

```
<dependencies>
    <dependency>
      <groupId>org.glassfish.jersey.containers</groupId>
      <artifactId>jersey-container-servlet</artifactId>
    </dependency>

    <dependency>
      <groupId>org.eclipse.jetty</groupId>
      <artifactId>jetty-servlet</artifactId>
      <version>${jetty.version}</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.eclipse.jetty</groupId>
      <artifactId>jetty-webapp</artifactId>
      <version>${jetty.version}</version>
      <scope>provided</scope>
    </dependency>
```

Héroku, applications Java

* Gestion des dépendances dans pom.xml

<dependency>
<groupId>postgresql</groupId>
<artifactId>postgresql</artifactId>
<version>xxx ex: 9.0-801.jdbc4</version>
</dependency>

</dependencies>

Héroku, Web service Rest Jersey

* https://jersey.java.net/documentation/latest/getting-started.html#heroku-webapp

1. Création d'une application

```
mvn archetype:generate -DarchetypeArtifactId=jersey-heroku-webapp \
-DarchetypeGroupId=org.glassfish.jersey.archetypes
-DinteractiveMode=false \
-DgroupId=com.example
-DartifactId=simple-heroku-webapp
-Dpackage=com.example \
-DarchetypeVersion=2.xx
```

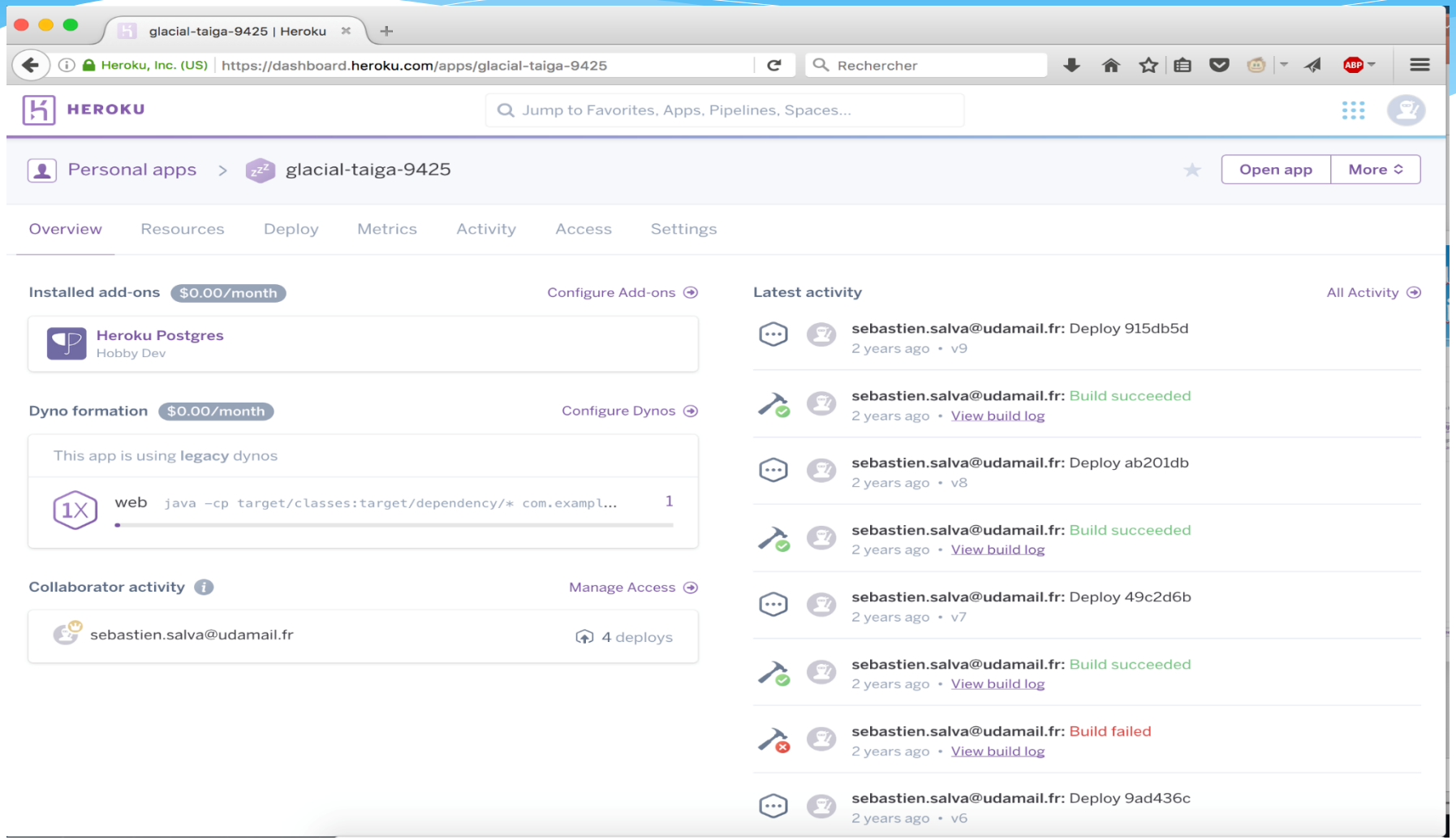
Héroku, Web service Rest Jersey

2. Création d'une appli Web packagée
   * mvn clean package

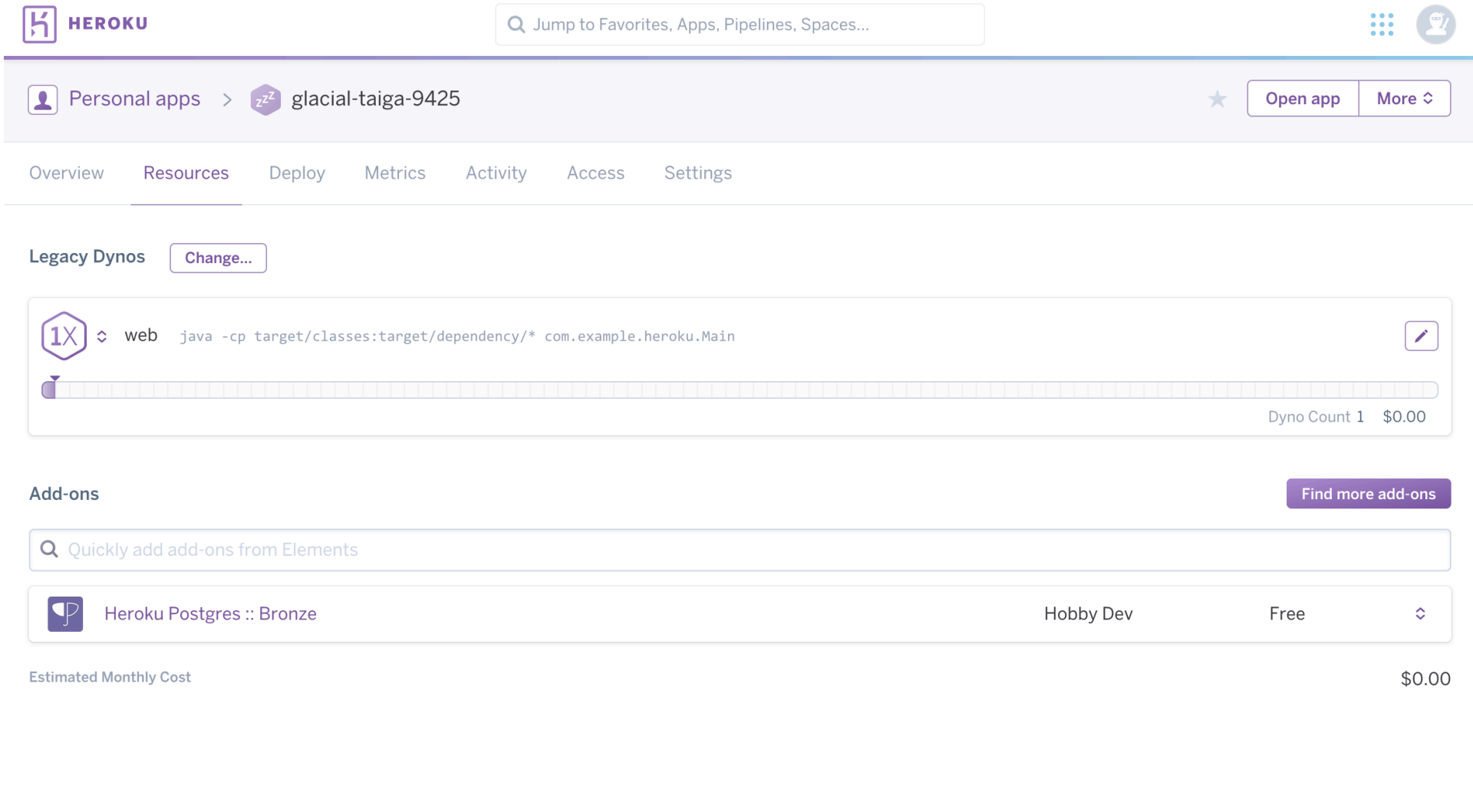
3. Déploiement
   * Git init, heroku create, git add, git commit, git push,

4. Accès:
   * URL fourni par ligne de commande
   * Ex: https://glacial-tiaga/-9425.herokuapp.com/wsrest/appel

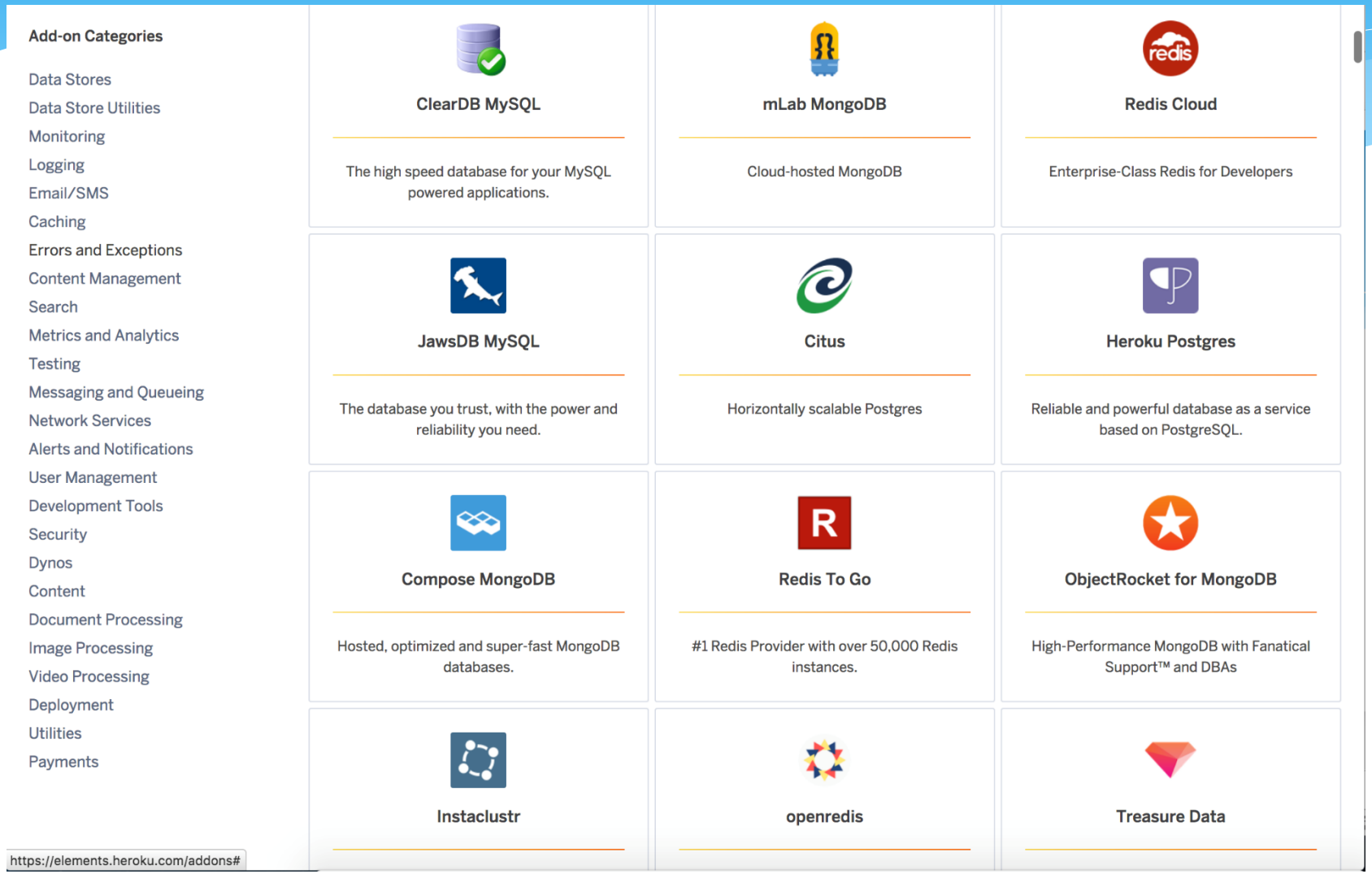
Héroku, interface d'administration



Héroku, interface d'administration



Héroku, interface d'administration



Choregraphie, orchestration ?

Composition de services Web

* Composition
  * Faire interagir des services Web ensemble
  * Déployés sur le même serveur, ou sur des Clouds/serveurs différents
* La composition appelé service composite, services invoqués appelé des composants de service
* D'un point de vue Client, service composite = service
* 2 types de composition
  * Orchestration
  * chorégraphie

Composition de services Web

* Difficultés:
  * Gestion des erreurs
  * Si 1 composant remonte une erreur, elle doit être gérée par le service appelant pour un retour vers le client

* 2 types de composition
  * Orchestration
  * chorégraphie

Gestion des services web

Orchestration des services

•Lorsqu'un service web coordonne d'autres services

•1 processus global avec appel vers d'autres services, gestion des erreurs

•Compositions simples en Java etc.

•Compositions complexe, besoin de meta langages -> BPEL,

Gestion des services web

Orchestration des services

- Langage BPEL

  - processus BPEL (processus écrit en XML qui décrit comment interagissent les WS suivant des stimuli extérieurs)

  - Besoin d'un serveur qui exécute les processus BPEL
    la gestion des erreurs doit être gérée par le processus (mécanisme de replis, re-exécution du processus)

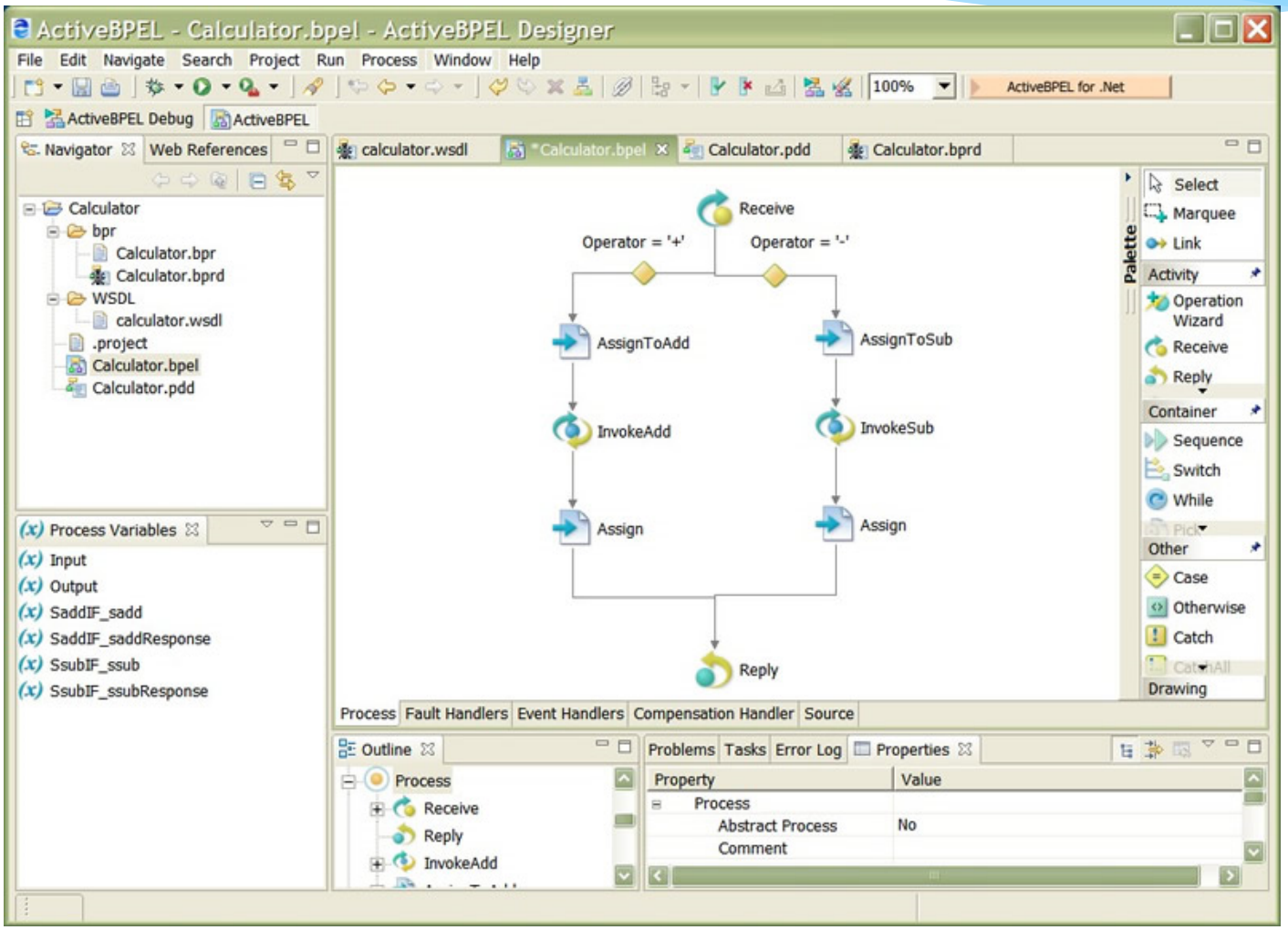
  - Langage de programmation de processus mais aussi interface graphique (boites)

Apercu de WSBPEL

- Definition des partenaires
- Utilisation de variables, assignation de valeurs (assign)
- Activités basiques (invoque, receive, reply, wait, throw)
- Activités structurés (while, switch, sequence,pick(temporisation)
- Correlation = session
- Scope découpage d'un processus en plusieurs parties
  - Pl. handler possibles par scope (conpensation, fault, event)

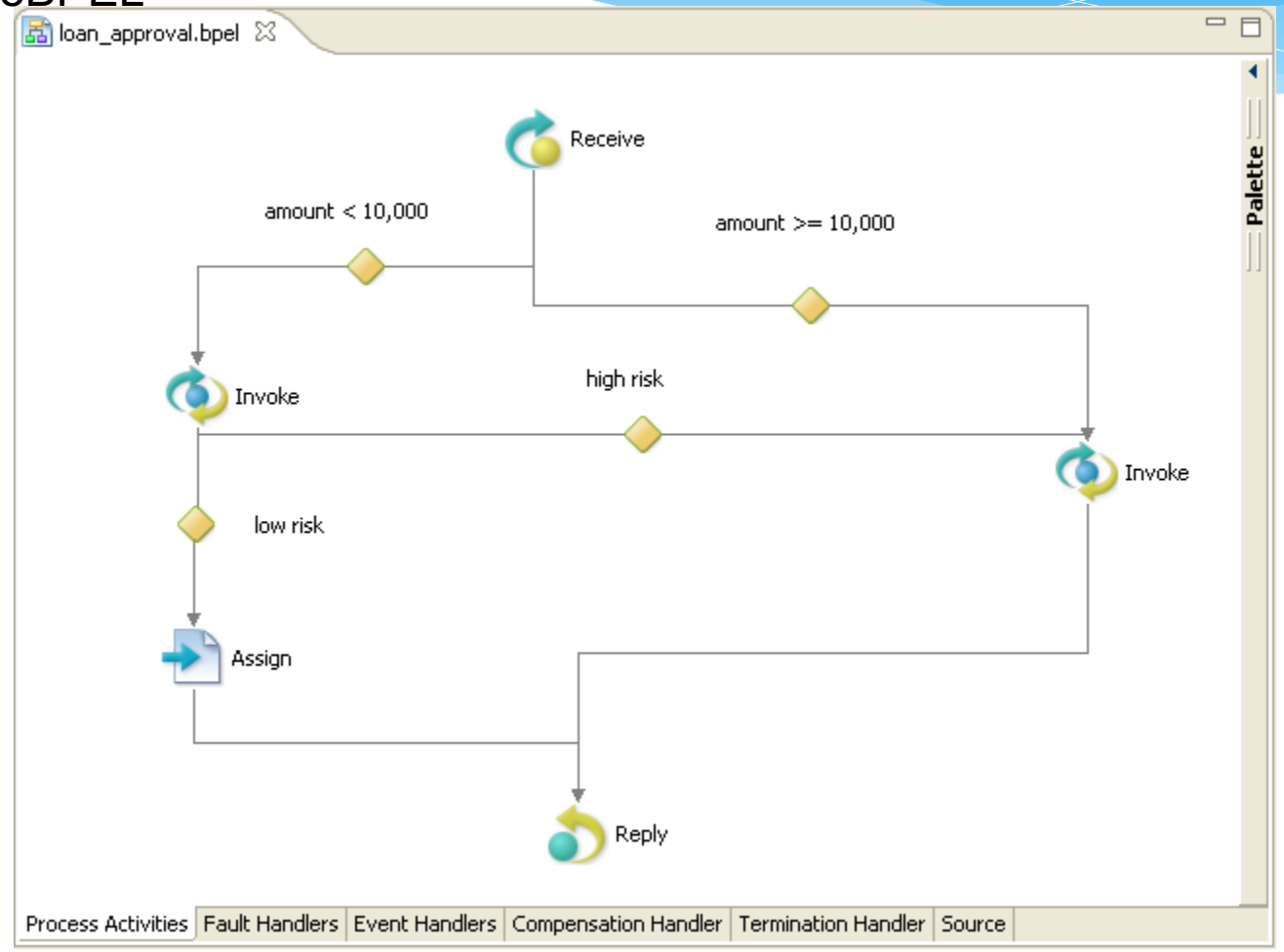
Apercu de WSBPEL

Avec ActiveBPEL



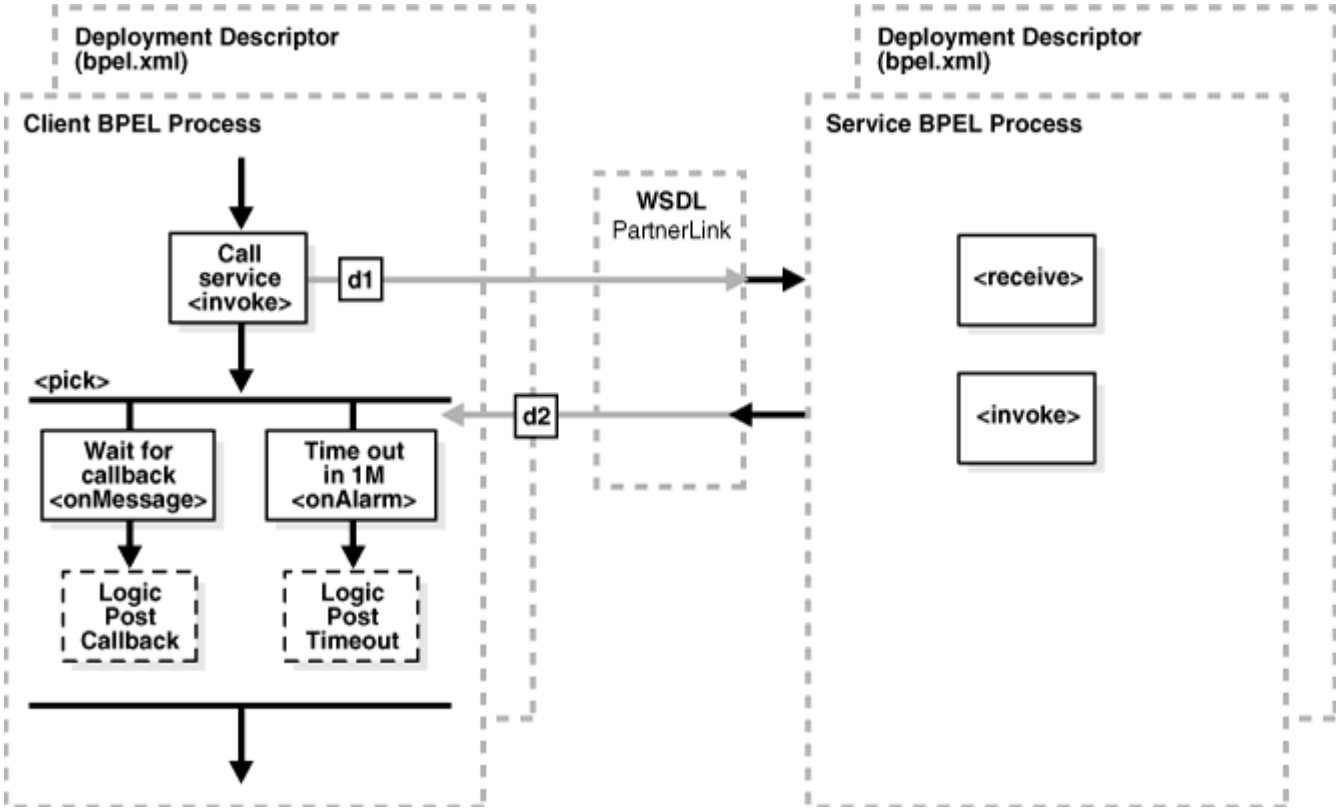
Aperçu de WSBPEL

Avec ActiveBPEL



Apercu de WSBPEL

Avec ActiveBPEL
Le pick



WS-BPEL

pl. moteurs

- Websphere,bpel process manager,biztalk, bpelmaestro
- Activebpel,pxe, twister

- Limitations
  - Service composite centralisé (si moteur tombe en panne, …)
  - Schéma de composition statique

Chorégraphie de services

•Comportement global basé sur les interactions des services entre eux.

•Chaque service web mêlée dans la chorégraphie connaît exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu.

•Services font office de services et de clients vers d'autres services

Chorégraphie de services

•Description des interactions de service uniquement de pair à pair

•Pas de processus, chaque service connait les actions à effectuer par rapport aux messages reçus

•Langages standards de description de choregraphies
- en XML WS-CL ou WSCI
- Descrption des messages
- Ordre des messages
- ne définit pas un processus global

•Travaux de recherche sur composition dynamique