PostgreSQL Service with Backup and Recovery for Cloud Foundry

Anton Kozmirchuk, Andrey Kokorev, Vyacheslav Nesterov, Elena Mikhailova
kozmirchuk@gmail.com, a.d.kokorev@yandex.ru, vyacheslav.nesterov@emc.com, e.mikhaylova@spbu.ru
Saint Petersburg State University
St. Petersburg, Russia

***Abstract*—Cloud Foundry is open source PaaS project, designed for the application developers to get them rid of the problems with the hardware. Cloud Foundry provides users with the ability to run applications in the container with different micro-services giving access to database management systems (DBMS), middleware software, development and testing tools. Each DBMS needs a broker that implements the main scenario of interaction with the applications: work with the database. This paper presents an architecture of the service broker providing the DBMS with the capabilities of backup and data recovery. The broker is implemented on PostgreSQL based on the proposed architecture. The broker gives two kinds of storage options: the local (temporary) storage of data and storage in the cloud. The designed architecture can be easily adapted to different DBMS. The approach contains various advantages: ease of development, a weak dependence on the internal structure of the DBMS, a clear task distribution between the modules.**

I. Introduction

Currently third platform [1] for building IT-infrastructures becomes increasingly popular. This platform is based on different hardware, software and network technologies including mobile devices, mobile internet, social networks, cloud infrastructures. It is being used to build all kinds of solutions for ”smart” economy.

In general, the third platform is characterized by rapid technological development in four areas: social networks, big data analysis, fast mobile access to internet, including corporate infrastructures, and cloud computing and services. Users of the growing number of mobile devices are producing more and more content that is easy to store in the clouds.

The number of mobile devices constantly grows and increases the need of cloud storage. Users are more and more often turning to social services and it is due to growing user activity in social networks. The content placed in the cloud may be available to the owners of various mobile platforms in the relevant format and can be distributed among users in social networks.

The challenges of ”big data” require more spending on hardware and software, and usage of cloud technologies allows to reduce spending on IT. Cloud technologies offer a new type of services — provision of storage and database management in one service. Also there is a type of cloud service – Platform as a service.

Cloud Foundry is used to build next-generation (3rd platform) applications. These applications often rely on modern data services such as No-SQL, KV stores, object stores and HDFS, in addition to leveraging more traditional data services such as SQL data bases.

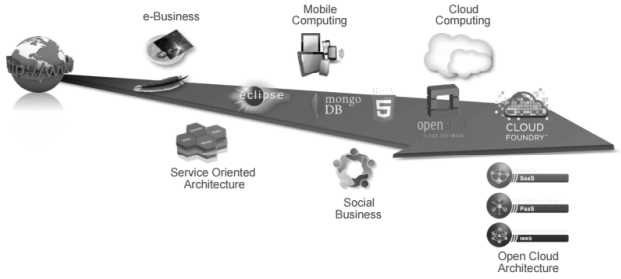

Fig. 1. Open cloud architecture

Cloud Foundry supports the full lifecycle, from initial development, through all testing stages, to deployment. Applications deployed to Cloud Foundry access external resources via Services. In a PaaS environment, all external dependencies such as databases, messaging systems, files systems and so on are Services.

There are a lot of social applications that are making use of cloud computing technologies. These applications usually involve using the existing user management capabilities of the social network to use cloud resources much like the content that is already being shared by social networking users.[17]

The goal of this research is developing Cloud Foundry additional services and services enhancements, providing high availability (HA), data protection, snapshots, and fault tolerance. In particular an architecture of broker which connects PostgreSQL and Cloud Foundry is presented. The broker provides also backup functionality for local and external storages.

II. Related work

The growing popularity of cloud computing is due to convenient, on-demand network access to a shared pool of configurable computing resources. A Cloud computing architecture consists of four layers: Hardware (managing the physical resources of the Cloud, including physical servers, routers etc), Infrastructure (servers, storage, networking and virtualization software that are necessary to support the computing requirements), Platforms (operating systems and application frameworks) and Application (usually with automatic scaling feature to achieve better performance, availability and lower operating cost). There are different types of Cloud Computing

ISBN 978-952-68397-6-9 (paperback), ISBN 978-952-68397-7-6 (PDF)

services: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). SaaS (software-as-a-Service) [13] gives an opportunity to use the provider's applications running on a cloud environment.

The user can access the application from various client devices through a web browser not paying attention to manage or control the underlying cloud infrastructure: hardware or software resources.

Platform as a service (PaaS) is a category of cloud computing services that allows customers to develop, run, and manage web applications without the complexity of creating and maintaining the infrastructure [15]. PaaS vendors offer a development environment to application developers. In the PaaS models, cloud providers deliver a computing platform, typically including operating system, programming-language execution environment, database, and web server. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers.

IaaS (Infrastructure-as-a-Service) provide resources as services to the user — in other words, they basically provide enhanced virtualization capabilities. According to that, different resources may be provided via a service interface: Data and Storage Clouds deal with reliable access to data of potentially dynamic size, weighing resource usage with access requirements and / or quality definition. Characteristics of infrastructure as a service are mentioned in [13].

Compared with the SaaS (Software-as-Service) to end-user, PaaS has more flexibility. Google App engine is a typical PaaS platform, which allows developers to write and run their own applications on the platform. It also helps developers store data and manage the server. However, it is not convenient for developers to write programs using java or Python API development. LongJump, which provide a platform for quickly building enterprise Web applications, significantly shorten the development cycle. But it does not have good scalability and mass data management capabilities, and can only provide service for some developers. This paper describes the techniques and frameworks used by the two platforms. And then we bring forward an improved framework for PaaS platform which combine with the advantages of GAE and LongJump platform by adopting Mashup technology. Using the improved framework, we can build the PaaS platform which is easier and more flexible for development of web applications [11].

Some of PaaS systems are commercial, some others are open source cloud platforms. Cloud platforms like OpenStack, CloudStack and Eucalyptus are open source alternatives to proprietary solutions for commercial businesses. These models provide many similar benefits that distinguish any open source program: community access, shared resources, rapid updates and freedom to not be tied to the limits of the software owners. IBM's cloud software and services is based on open standards (Fig.1) [14]. All platforms offer application hosting and deployment environment, along with various integrated services. Services offer varying levels of scalability and maintenance. Developers can built an application and upload it to a PaaS that supports their software language of choice, and the application runs on that PaaS. PaaS offers many usefull services for application developers. One of such services is application backup and recovery.

Evolution of sphere cloud technologies is the reason of appearance various kind of PaaS platforms for different use-cases. Most of modern PaaS platform such as Heroku, Google AppEngine, Amazon Elastic MapReduce, Microsoft Azure, Cloud Foundry provide various environments for application development. Despite similar model of environment providing, PaaS platforms are disparate. The main differences are choice of frameworks, programming languages, services, distribution model.

Heroku is the proprietary platform oriented on development of Web-application with wide range of programming languages (JavaScript, Ruby, JVM languages, Go, PHP, Python). The PaaS contains big amount of services and capabilities to develop custom.

Google App Engine provides proprietary service for Web development involving Java, Go, Python, php languages with access to Google services such as Google Search, Security Scanner, Google Cloud SQL. Also there is a capability of usage NoSQL data storage and Memcache.

Yandex Cocaine - fast growing open-source Paas platform for Web development with support of most popular programming languages and containers. Also there are support various of modules such as Elasticsearch, URL Fetcher, tools for logging, access to storage services Elliptics and MongoDB.

Amazon Elastic MapReduce is the proprietary platform provides environment for development application based on MapReduce model.

Microsoft Azure is the propriety platform based on .Net framework, also support the most popular languages and frameworks, provides environment for Web and Enterprise development. Azure offers opportunity to pick DBMS, object storage, archive services.

Cloud Foundry is the open source PaaS platform offers multiple language support, different services for data storage, data processing and etc. Cloud Foundry runs on most popular IaaS platforms and provides service customisation.

Data storage and processing systems are an essential part of almost any application. Relational DBMS such as Oracle, MySQL, Microsoft SQL Server, PostgreSQL and others are successfully used in this capacity. Additionaly, in the moment various NoSQL solutions are gaining popularity: MongoDB, Cassandra, Redis and others. Cloud applications, as desktop ones, need such systems. Specific adaptations, called services, are required in order to use theese systems in a cloud.

Modern relational DBMS, in spite of NoSQL newcomers, not lose their relevance and occupy solid positions in a wide range of problems. At the time of writing this paper, for PostgreSQL (one of the most popular open-source RDBMS [16]) exist a few services at Cloud Foundry platform. However, from our point of view they have some disadvantages, which are discussed later.

III. Cloud Foundry

Cloud Foundry — a PaaS platform for cloud-based systems intended to create another abstraction layer for a virtual

environment. Cloud Foundry provides an opportunity to run a multi-purpose application that is independent on any specific infrastructure. In other words Cloud Foundry operates not machine-specific resources, such as IaaS and containers with applications.

Cloud Foundry provides users with the ability to run applications in the container with different micro-services such as operating systems, database management systems, middleware software, development tools and testing. The application containers are a mean of logical isolation of application. The operating system is virtualized in the container with all the resources and services that can be scaled as needed.

A. Cloud Foundry Architecture

The architecture of Cloud Foundry comprises many micro-services that interact with each other. They can be split into logical components (Fig. 2), each of which provides running an application in Cloud Foundry.

1) Router: routes incoming traffic to virtual machines with applications.

2) Authentication: is required for user authentication and logical access isolation between applications.

3) App lifecycle: components running an application, monitor the state of application and manage its lifecycle.

4) App Storage & Execution: Components providing application execution, keeping application source code, buildpacks, droplets.

5) Services: Most applications use various external services and API's. Service brokers are necessary to bind services to applications.

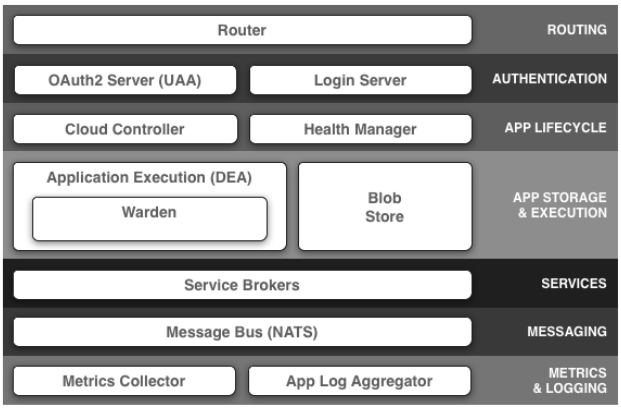

Fig. 2. Architecture of Cloud Foundry

6) Messaging: Lightweight publish-subscribe and distributed queueing messaging system written in Ruby.

7) Cloud Foundry installation: BOSH is a system designed for installing Cloud Foundry on top of stack of existing infrastructure. BOSH is able to create hundreds of virtual machines and install necessary software. In addition BOSH performs system monitoring, failure recovery, software updates.

B. Cloud Foundry services

Cloud Foundry services enable applications to use resources, granted by external sources. Each service consists of software, providing certain resources, and a service broker – an application implementing integration of service with Cloud Foundry. Service broker provides catalog of services and plans, which are instances of certain services. Broker must be able to reserve resources for applications (i.e. create service instances) and bind applications to service instances (i.e. grant access for an application and give necessary meta data for communication with service).

For purpose of integration with Cloud Foundry a service have to implement API which is used in interaction with Cloud Foundry (Fig. 3). API composed of 5 functions: getting offered services, service instance creation, service instance deletion, bind service to an application, unbind service from an application. Any valid implementation of this API is a service broker. It can be deployed as a user application, installed on a BOSH virtual machine, or executed remotely.

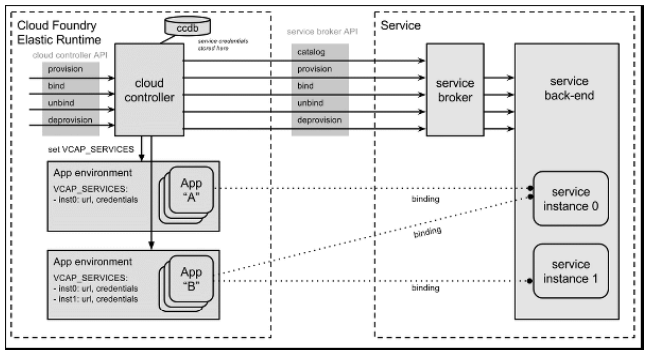

Fig. 3. Service broker interaction with Cloud Foundry

C. PostgreSQL and Cloud Foundry

Now there are a few simple ways to integrate PostgreSQL [3] to Cloud Foundry. The first one is a manual control of the DBMS that needs to add a separate service for each application that provides a single database instance. Another way is to use SaaS platform ElephantSQL [4] that allows using database instances for applications. There are also open source project implementations that allow you to get the database resources for the application instances.

The first method has a disadvantage: the administrator must manually create a database for each instance of an application that requires resources database, and configure db for specific needs required by the application. It's a tedious and complex task that requires specific knowledge of the SQL dialect used in PostgreSQL. The advantages of this method include the possibility to configure the database in arbitrary way according to the requirements of an application.

The usage of SaaS platform ElephantSQL allows to forget about manual database configuration and supporting its efficiency. ElephantSQL offers a wide range of tariffs, which define the parameters of its service. In addition, the user has the option to create a backup and to recover the application instance. The disadvantages of this method is proprietary of ElephantSQL, each user needs to make monthly payment for

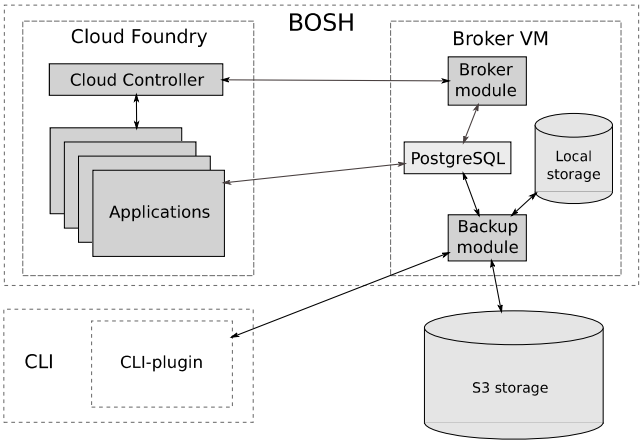

Fig. 4. General architecture

the resource use. Therefore, the application developer has no opportunity to tune the database specifically.

The third approach involving the use of open source brokers is good because the developer can easily obtain the necessary resources, and the administrator of Cloud Foundry can add new functionality that may be needed by applications. There is no open source broker that would provide backup/restore features.

TABLE I. SOLUTION COMPARISON

	Easy Deployment	Customization	Open source	Backup& Restore
ElephantSQL	+	–	–	+
Manual approach	–	+	?	–
Another service brokers	+	–	+	–

IV. GENERAL PROJECT ARCHITECTURE

The project consists of two fundamental parts: command line interface plugin (CF CLI) and the broker (Fig. 4). The plugin may be installed by any user of CF CLI. The broker is a BOSH job (which is being deployed as a virtual machine created by BOSH), dividing into RESTful-service, default-configured P ostgreSQL, b ackup m odule a nd l ocal backup storage. In addition the broker can access external storage using S3-API.

A. Broker

The broker consists of the following parts:

- CF broker. Provides CF $\leftrightarrow$ Broker interaction.
- Backup module. Responds for backing up, restoring and data transferring between DBMS and storages.
- Storage interface. Gives access to local and external storages.

B. Backup storages

There are two options for backup files storage: local and external storages. The broker communicate with storages using

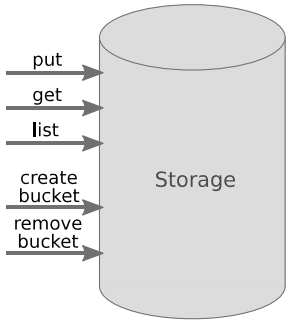

Fig. 5. Storage API

simple API (Fig. 5). Two types of storages designed for different purposes (compared at Table II):

Local storage is based on the file system of broker's virtual machine. Storage uses directory, specified in the configuration file. In this directory backups from all existing service instances are stored.

Local storage intended to be used for temporal backup storing. One of the use cases may be following: in situation, when newly made backup have to be downloaded and moved somewhere from the cloud, it is very inefficient trying to upload backup into external storage and then download from there, as backup file can have significant size. Same for the opposite, when you need to restore from backup, which is not stored in cloud.

We should notice that this storage is not designed for long-term storing and doesn't guarantee data safety. For example in case of BOSH-job restart all data will be lost.

External (supporting S3-API) storage designed for reliable data storing. The broker uses login credentials to access storage. Backup is being stored directly, avoiding long-storing in the local storage. This option implies requirement of high connection quality to external storage.

The implementation is based on AWS SDK for S3 storages. Firstly, the backup is being saved to the local storage. Then the bytes stream from the backup is being redirected to the socket that is connected to remote storage. This approach implies repeated uploading in case of failure. In case of succesfull upload the local copy of the backup will be deleted. This approach avoids recreation of the backup if the backup process fails.

The choice of the S3 storage is not accidental. This interface is very popular for long-term storage facilities. In addition to the well-known "Amazon S3", there are many storage systems, implementing S3 intterface. As a result, the service can be integrated with a variety of different data warehousing systems.

TABLE II. LOCAL AND EXTERNAL STORAGES COMPARISON

	Local	**External**
Backup&restore time	fixed	depends on connection quality
Reliability	unreliable	reliable
Accessibility	always	depends on outer world
Disk space	limited by platform, small	limited by external storage plan, potentially large

C. Backup module

Backing up and data restoration are being performed using standard utilities distributed with PostgreSQL: *pg_dump*, *pg_restore*. This approach allows to preserve all the guarantees given by them (such as data consistency and non-blocking execution). Furthermore, processes of backup and recovery remain transparent for the experienced users.

When request for data backup is received, backup is being created by *pg_dump* tool and saved directly to chosen storage. In case of success the user receives meta data including name and creation time for further use. Otherwise backup module resends an error message received from backup tool or one of storages (for example in case of inaccessibility or insufficient disk space). Similar process is performed when request for data recovery is received.

D. CF broker

The CF broker module is a RESTful service implementing CF broker API: it services catalog, provision, deprovision, bind and unbind requests. Each instance (in terms of Cloud Foundry) maps to database in PostgreSQL.

For requests sent by CF cloud controller broker performs the following actions:

1) catalog: specified meta data is returned
2) provision: broker creates new database (cloud controller creates new service instance)
3) bind: broker creates the database user in the already created database (provision request) and returns jdbc url, containing necessary data, which an application uses to connect to the database
4) unbind: user is removed, now database is unavailable for the application.
5) deprovision: database and all users are removed (cloud controller deletes this service instance)

For purpose of backup and restore a special API allowing performing different actions is designed:

1) `/backup_instance/<instance_id>?storage=<storage_type>`
   Create backup in specified storage and get backup file name
2) `/restore_instance/<instance_id>/<backup_name>?storage=<storage_type>`
   Restore data from specified backup file in chosen storage
3) `/upload_backup/<instance_id>/<backup_name>?storage=<storage_type>`
   Upload backup file into specified storage
4) `/download_backup/<instance_id>/<backup_name>?storage=<storage_type>`
   Download specified backup file
5) `/backups/<instance_id>`
   Get list of available backups

Requests arguments are the following:

1) `instance_id`
   Unique identifier provided by cloud controller for each instance
2) `backup_name`
   Name of backup file stored in of storages
3) `storage_type`
   Storage name, optional. Available values: 'local', 's3'. Default is local storage.

E. Plugin

The plugin for the CF CLI is executed on the user's computer and has access to the same API as the user. This allows users to automate some sequences of actions. In this work plugin is used to simplify human interaction with broker. Plugin provides special commands corresponding commands to all API calls.

However, this approach in the context of developing a broker has a significant drawback: capabilities are limited by user access rights. This means that in most cases the plugin could not locate the broker virtual machine which is needed for using backup/restore API.

Our solution to this issue at the moment is to fix broker vm address in the infrastructure. As a result, the plugin performs queries on a pre-known IP address, bypassing the stage of search for a broker vm. This approach lacks flexibility and can lead to significant costs in case of the need to change the IP address of the broker.

Another solution is to have a proxy application which would resend all requests from plugin to the broker. In that case there is no need in special address for the broker, but now there is another need in special application name. Also this application should be installed by each plugin user. In other words this solution replaces one dependency with another and makes usage more complicated.

V. Results and Prospects

In this paper we discuss problem of data preservation in cloud applications with backups in DBMS. Many modern cloud platforms use service brokers model in purpose of providing DBMS for applications. We use open source platform Cloud Foundry in our research. We developed PostgreSQL service broker with backup and data recovery capabilities.

The developed broker architecture can be easily adapted to different DBMS. There are several advantages of our approach: ease of further development, limited dependency on the DBMS structure, clear distribution between modules. Minor drawbacks of the architecture result from limitations of the Cloud Foundry platform which can be overcome in the future.

References

[1] F. Gens, The 3rd Platform: Enabling Digital Transformation, Web: http://www.tcs.com/SiteCollectionDocuments/White-Papers/3rd-Platform-Enabling-Digital-Transformation.pdf
[2] Cloud Foundry Documentation, Web: http://docs.cloudfoundry.org/
[3] PostgreSQL Documentation, Web: http://www.postgresql.org/docs/
[4] ElephantSQL Documentation, Web: https://www.elephantsql.com/docs/index.html
[5] AWS Documentation, Web: https://aws.amazon.com/documentation/
[6] Microsoft Azure Documentation, Web: https://azure.microsoft.com/en-us/documentation/

[7] Product Documentation for OpenShift Enterprise, Web: https://access.redhat.com/documentation/en/openshift-enterprise/

[8] IBM Bluemix Documentation, Web: https://www.ng.bluemix.net/docs/

[9] Salesforce Developers Documentation, Web:https://developer.salesforce.com/docs/

[10] T. Cordeiro, D. Damalio, N. Pereira, P. Endo, A. Palhares, G. Gonçalves, D. Sadok, J. Kelner, B. Melander, V. Souza, J.-E. Mangs, *Open Source Cloud Computing Platforms*, 2010 Ninth International Conference on Grid and Cloud Computing. IEEE, 2010. XCP, Eucalyptus and Open Nebula.

[11] Z. Shu-Qing, X. Jie-Bin, *The Improvement of PaaS Platform*, First International Conference on Networking and Distributed Computing, IEEE, 2010.

[12] M. Boniface, B. Nasser, J. Papay, S. C. Phillips, A. Servin, X. Yang, Z. Zlatev, S. V. Gogouvitis, G. Katsaros, K. Konstanteli, G. Kousiouris, A. Menychtas, and D. Kyriazis, *Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds*, Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services, Washington, DC, USA, 2010, pp. 155–160.

[13] N. K Salih, T. Zang, *Variable service process for SaaS Application*, Research Journal of Applied Sciences, Engineering and Technology, 2012.

[14] D. Krook, OpenStack and Cloud Foundry – Pair the leading open source IaaS and PaaS, Web: http://www.slideshare.net/DanielKrook/joint-open-stackcloudfoundrymeetup

[15] B. Furht and A. Escalante, *Handbook of Cloud Computing*, Springer, 2010.

[16] DB-Engines Ranking – popularity ranking of database management systems, March 24, 2016, Web: http://db-engines.com/en/ranking/

[17] Sanjay P. Ahuja and Bryan Moore, *A Survey of Cloud Computing and Social Networks*, Network and Communication Technologies, Vol. 2, No. 2, pp. 11-16, 2013.