Sample of Hypothetical Project for Banks using AWS and NodeJS

by Weverton de Souza Castanho
Bluefin Resources – Specialist Recruitment Solutions – Anna-Maria Julie

Introduction

The Bluefin ask me more details about my projects in NodeJS and I decide to write a documentation with brief details about a sample of Architecture using NodeJS and AWS Cloud enviroment. Sometimes write details about your projects in a resume will expand many words where you need to simplify, so the resume lost your mais goal, simplify details, my decision about a document is explain more about me and my professional experiences. I have many others examples of project, but probably these will expand days and days drinking the amazing Australian Wine! :)

I have many Non Disclosure Aggremments (NDA) sign with my customers, I can’t expose customers name details and codes about projects I work with, this is not just an respect for a NDA signed, but it’s an ethical behaivor. I decided build from zero a sample of document, there aren’t UML diagrams, requirement documents, integration documents, but just an idea about a proposal of project, all details could be explainned more in a posible interview if the customer have interest to talk more about. Please don’t hesitate to ask me more about this document, it’ll be a great pleasure explain everthing!

This example consider an hypothetical bank, where the bank decide to start new projects using AWS Cloud enviroment, my responsability and of my time is present a new sample of architecture where security, performance, operationality, availability and scalability are the main elements to integrate this new kind of discipline (cloud) with legacy system in using high platform, low platform and possible the mobile platform. This project is drive to banks where they are moving to digital marketing, where the speed of development, low cost and quick deploy of applications are mandatory for a competitive marketing. The architecture definition use resources of elastic feature of AWS where new instances of server are create on demand, the mais goal of this technology is not just the simple code o NodeJS but the simple script of CloudFormation tool of AWS, where all the infrastructure is a CODE! There is an example of cloudformation template available with this documentation, is an example that I built for a test, I haven’t and I couldn’t show one original sample of my customer where I sign an NDA, but these example is similar the use of this project.

How I told to Bluefin my team is little, we are 3 solutions architects working with projects for customers like Santander Bank, InfoSys, Unilever… All staff is qualify to all Software Life Cycle Development, requirement, analysis, development, deploy, quality assurance… In the most time in the project all of staff is programming, we expand almost of 70% of the projects programming hands on, not just software but scripts to deploy in cloud enviromments.

This is an example of project that I and my team are responsable when we start a project, the example is a proposal for a Bank system based this requirements:

- AWS Cloud Enviroment
- Virtual Private Cloud (Public and Hidden Security Group)
- Elastic Load Balancer
- OAuth
- AWS Aurora Database Cluster
- Angular JS
- Node JS
- JQuery JS
- BootSrap JS
- Integration with IBM Security Access Manager (I just show the integration of IBM Security with RACF)

Security considerations

The entire infrastructure is encapsulated into a VPC; all Internet traffic goes through:
VPC Internet Gateway: gateway to Internet for VPC components (PublicSubnet*).
There are two main Security Groups which separate the internetfacing parts of the architecture, meaning the webservers (and associated components), from the private parts, meaning the database cluster mainly.

Public Security Group: HTTP/HTTPS/SSH access permitted from outside.

Hidden Security Group: Database, access permitted only from web stack to DB stack.

There are two Subnets associated with each group (in distinct availability zones):

- **PublicSubnet A/B:** web servers stack, the subnets are publicly accessible.
- **HiddenSubnet A/B:** database stack, the subnets are not publicly accessible.

The routing for these subnets is as follows:

Public Route Table: opens traffic from the public subnets to the Internet.
Hidden Route Table: ensures privacy for the hidden subnets.

Currently the hidden subnets have no access to the outside world (saw no point yet).

Scalability considerations

Web Cluster: the web servers are governed by an AutoScalingGroup and sit under an ElasicLoadBalancer instance for load-balancing and fault-tolerance.Vertical scaling: web servers can be upgraded to larger memory/compute/storage capacities without downtime.
Horizontal scaling: The AutoScalingGroup implements policies for scaling up or down based on CPU usage metrics of the nodes (implemented via CloudWatch Alarms). This is

Database Cluster: The RDS Aurora cluster has a master (writer) instance and also a read replica at this time.
Vertical scaling: instances can be upgraded to larger memory/compute/storage capacities without downtime.

Horizontal scaling: up to 15 read replicas can be added without downtime; to scale write operations partitioning would be an option.

Networking infrastructure: All networking (glue) components such as ELBs, ASGs, InternetGateway, VPC Router (and probably many more) are scaled-out by the AWS ecosystem.

Availability considerations

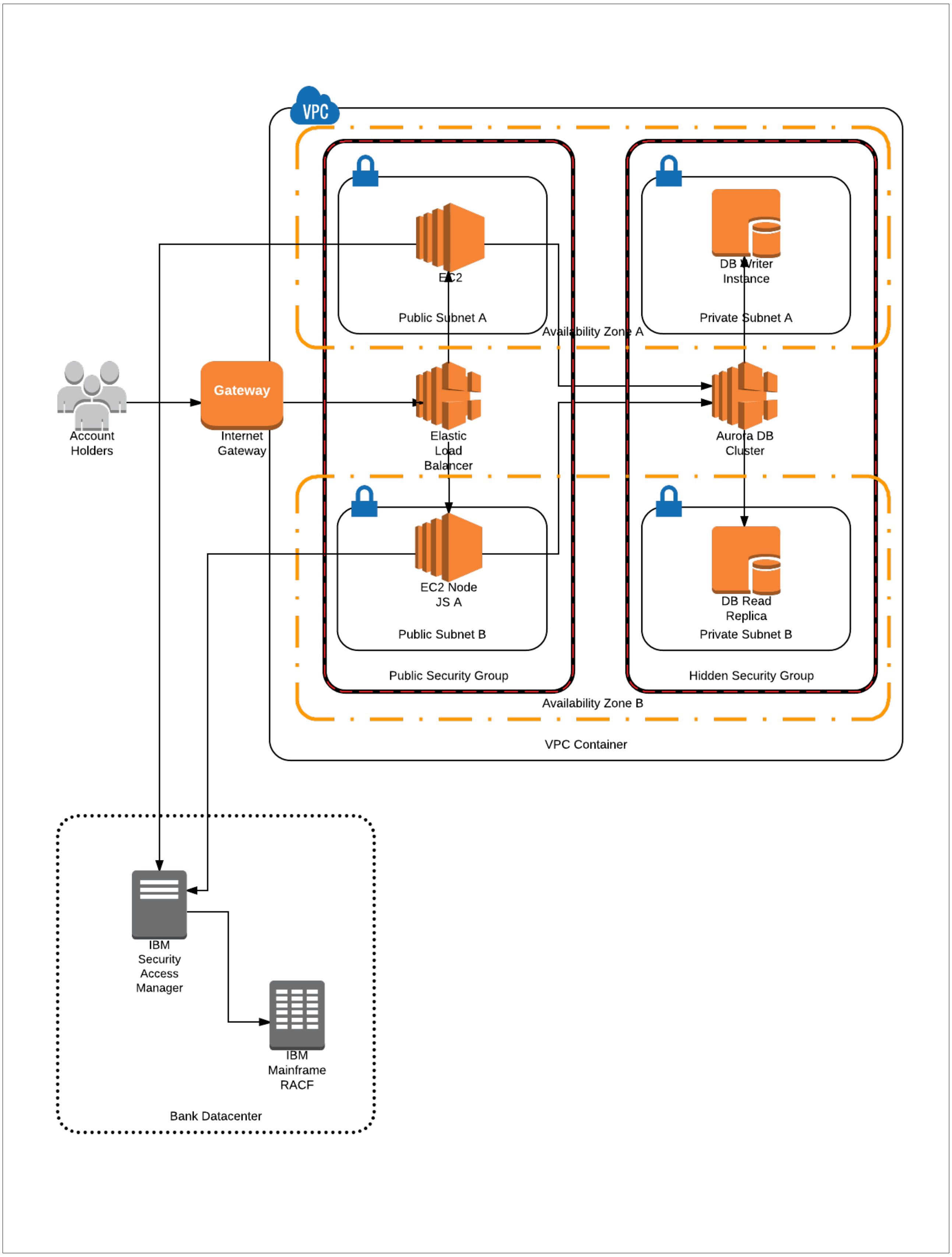
Web Cluster: Both the ASG and ELB instances which govern the web servers currently span over two Availability Zones (although probably all should be used).

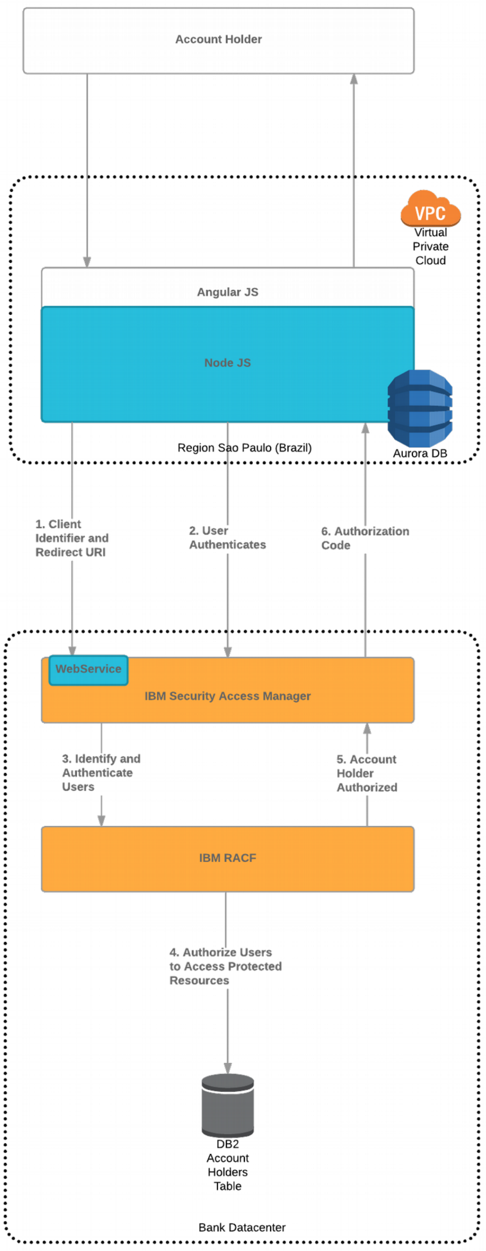
Database Cluster: The RDS Aurora deployment has better fail-over behavior (less downtime) than an analogous installation of ELB + RDS/MySQL instances; although downtime is still possible (in some cases of fail-over), it is much less likely (usually 100 - 200 seconds).

DDoS by attack or failure: I tried as much as possible to avoid any SPoFs (single-point-of-failure) to keep this infrastructure robust and my intent was that DDoS attacks would either:

be avoided; mostly by encapsulation (VPC, subnets, traffic rules etc.) or… translate to high loads which AWS can take without experiencing service outage; obviously this is still bad, but manageable.

Overall Architecture Design




Account Holder
VPC
Virtual Private Cloud
Angular JS
Node JS
Aurora DB
Region Sao Paulo (Brazil)
1. Client Identifier and Redirect URI
2. User Authenticates
6. Authorization Code
WebService
IBM Security Access Manager
3. Identify and Authenticate Users
5. Account Holder Authorized
IBM RACF
4. Authorize Users to Access Protected Resources
DB2 Account Holders Table
Bank Datacenter

RACF IBM zOS

Resource Access Control Facility or RACF provides the tools to help the installation manage access to critical resources.

Any security mechanism is only as good as the management control of the people who access the system. Access, in a computer-based environment, means the ability to do something with a computer resource (for example, use, change, or view something). Access control is the method by which this ability is explicitly enabled or restricted. It is the responsibility of the installation to see that access controls that are implemented are working the way they are supposed to work, and that variances are reported to and acted on by management.

- Identify and authenticate users
- Authorize users to access protected resources
- Log and report various attempts of unauthorized access to protected resources
- Control the means of access to resources
- Allow applications to use the RACF macros

RACF uses a user ID and a system-encrypted password to perform its user identification and verification. The user ID identifies the person to the system as a RACF user. The password verifies the user's identity. Often exits are used to enforce a password policy such as a minimum length, lack of repeating characters or adjacent keyboard letters, and also the use of numerics as well as letters. Popular words such as "password" or the use of the user ID are often banned.

OAuth 2.0 workflow

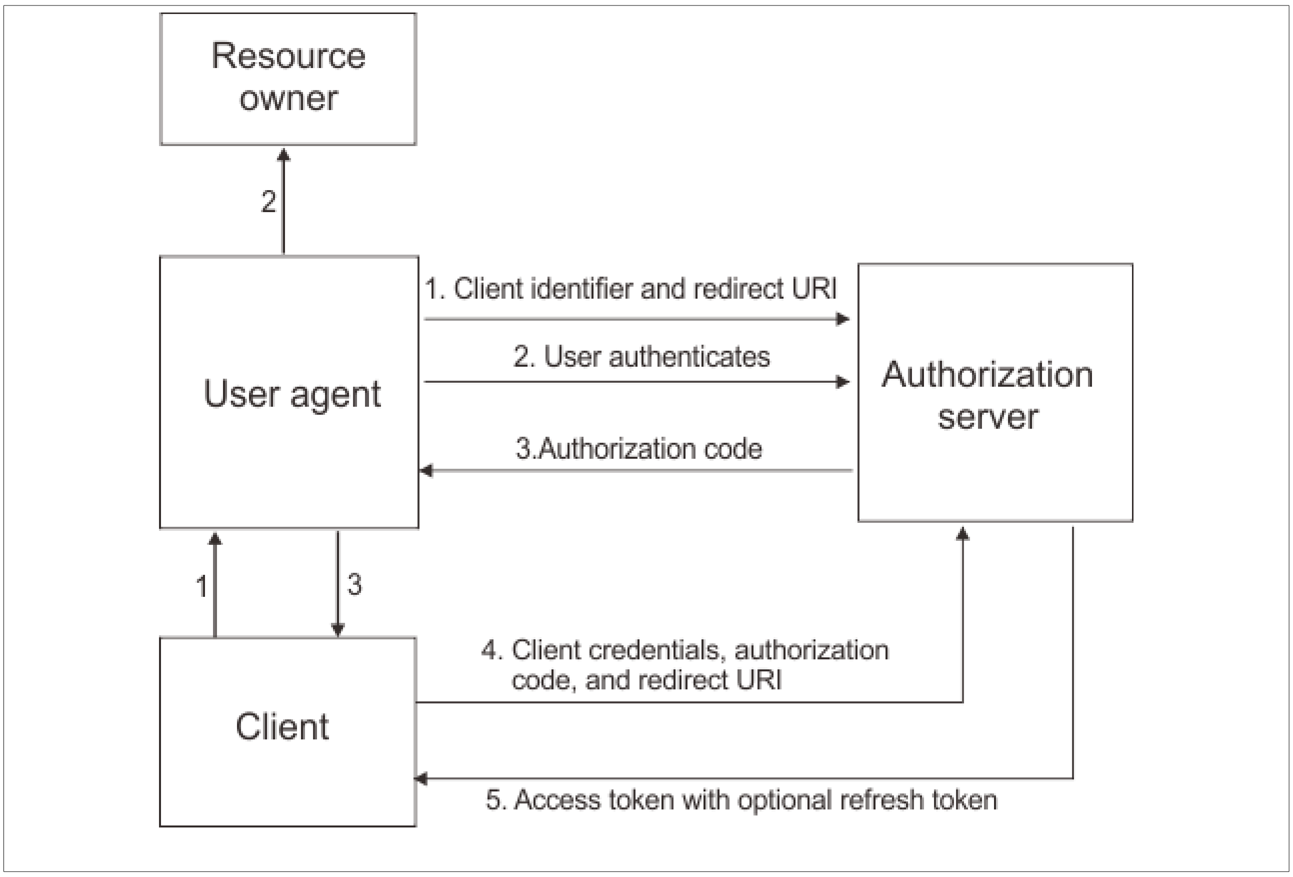
Advanced Access Control supports the following OAuth 2.0 workflows.

Authorization code flow

The authorization code grant type is suitable for OAuth clients that can keep their client credentials confidential when authenticating with the authorization server. For example, a client implemented on a secure server. As a redirection-based flow, the OAuth client must be able to interact with the user agent of the resource owner. It also must be able to receive incoming requests through redirection from the authorization server.

Reference

This file is written in Markdown format.
https://daringfireball.net/projects/markdown/basics


Resource owner
2
1. Client identifier and redirect URI
2. User authenticates
User agent
Authorization server
3.Authorization code
1
3
4. Client credentials, authorization code, and redirect URI
Client
5. Access token with optional refresh token

Descripton example of authorization(OAuth with IBM Security System Manager) code workflow diagram involves of the following steps :

1. The OAuth client initiates the flow when it directs the user agent of the resource owner to the authorization endpoint. The OAuth client includes its client identifier, requested scope, local state, and a redirection URI. The authorization server sends the user agent back to the redirection URI after access is granted or denied.

2. The authorization server authenticates the resource owner through the user agent and establishes whether the resource owner grants or denies the access request.

3. If the resource owner grants access, the OAuth client uses the redirection URI provided earlier to redirect the user agent back to the OAuth client. The redirection URI includes an authorization code and any local state previously provided by the OAuth client.

4. The OAuth client requests an access token from the authorization server through the token endpoint. The OAuth client authenticates with its client credentials and includes the authorization code received in the previous step. The OAuth client also includes the redirection URI used to obtain the authorization code for verification.

5. The authorization server validates the client credentials and the authorization code. The server also ensures that the redirection URI received matches the URI used to redirect the client in Step 3. If valid, the authorization server responds back with an access token.