**National College of Ireland**
**Project Submission Sheet – 2015/2016**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Apurv Deshpande |
| **Student ID:** | x15001687 |
| **Programme:** | Cloud Computing |
| **Year:** | 2016 |
| **Module:** | MSc Reserach Project |
| **Lecturer:** | Manuel Tova-Izquierdo |
| **Submission Due Date:** | 22/08/2016 |
| **Project Title:** | Comparing AWS vs Azure Cloud Patterns Compatibility with Object Oriented Design Patterns |
| **Word Count:** | 5558 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

| | |
|---|---|
| **Signature:** | |
| **Date:** | 22nd August 2016 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS:**

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Comparing AWS vs Azure Cloud Patterns Compatibility with Object Oriented Design Patterns

Apurv Deshpande
x15001687
MSc Reserach Project in Cloud Computing

22nd August 2016

**Abstract**

Cloud Computing provides Scalability, Reliability, Pay Per Use features that will benefit organisations by moving their applications to the cloud. Object Oriented Design patterns have been used in Software Industry for a long time now and many applications are designed using it. Many Cloud Computing Service Providers has provided patterns to help migration of Applications to cloud. Various terminologies like owl web ontology, ODOL ontology has been used to describe design patterns. This can also be extended to define cloud patterns.
This research consist of two methods. In the first method, A score based methodology is proposed to decide whether AWS or Azure Patterns are more compatible with object oriented design patterns. In the second method, actual components of AWS and Azure patterns are considered which takes active part in implementing this pattern to check the compatibility with design patterns. To compare the design and cloud patterns, a schematic based representation using ODOl ontology is proposed which describes the properties and components of design and cloud patterns.

# Contents

# 1 Introduction

The design patterns have been used for a long time now for software development. With the evolution of technology cloud computing has come as a cheap, reliable, secure source for hosting applications. With the growth of cloud computing, many organisations are moving their business to the cloud. But with it the problem comes on how to migrate the legacy applications to cloud that were designed to work in only specific environments. It is not possible to completely redesign the application as it is too costly and time consuming. Also it is difficult to migrate applications that are written in specific languages and those which are platform dependent. Even if different applications has different architecture and design, mostly they share the common design pattern. For example, singleton pattern, Observer Pattern and Decorator pattern are used in many applications. With the advancement in technology different techniques have been developed to define these design patterns. Mostly the design patterns used are the 23 standard design patterns defined by gang of four. Gamma (1995). Also as advancement in Cloud computing many cloud vendor like AWS and Azure researchers have published patterns to help application portability. In this research,the patterns used are published by cloud vendors AWS and Azure. A ODOL ontology has been used to define both design patterns and cloud patterns. This research is dependent on score based methodology as proposed in Di Martino et al. (2015) in which compatibility of design patterns with AWS Patterns is done. With some modifications in formulas,the same methodology is used to examine and compare Microsoft Azure's compatibility with design pattern. The main aim of this research is to find out which cloud patterns of AWS and Azure are more suitable for particular design patterns. Are AWS Patterns more compatible with design patterns or is Microsoft Azure

Patterns?. Section 1 is introduction. The rest of the paper is structured as follows. Section 2 is about literature review in which previous work done in same field are discussed. Section 3 presents the Methodology in which it is explained why particular design and cloud patterns are selected and also some insights are given on chosen methodology, Section 4 explain the design used with description of designs main components ,Section 5 describes the implementation process in which formulas are defined to check the compatibility of design patterns with cloud patterns. In Section 6 Evaluation is done with case study with selected design and cloud patterns and also Results are explained to decide whether AWS or Azure is good solution. Section 7 concludes the research with future work.

# 2 Literature Review

In this paperDi Martino et al. (2015) a score based methodology, based on schematic representation of patterns is proposed to find the common characteristics between design patterns and cloud patterns. The design pattern used in case study is observer and AWS cloud patterns are used. Observer Pattern was used in the study because it is very well known design pattern and the most commonly used one and also the components of Observer Pattern can be easily used for mapping with cloud pattern components. Hence due to this qualities of observer pattern I am using it in my research. This research is dependent on semantic representation of cloud and design patterns as discussed in Di Martino and Esposito (2013). In this paper Object Design Ontology Layer (ODOL), a formal pattern language is used. It explains the structural and behavioral properties of design pattern and can also be used to describe the cloud patterns. ODOl ontology was used in research because it is not limited. It can be expanded and properties can be defined dynamically. The main concept and properties used from ODOL ontology are Intent, Context, Consequence and Participants of Pattern. While the Motivation behind the pattern, Implementation of Pattern and Source Code of Patterns were discarded. In brief, Intent can be defined as Main Aim of Pattern, Context is how the pattern is Applicable, Consequence is the positive and negative effects of pattern, and participants are the active components of the pattern. I am using this properties to decide whether AWS Patterns are more compatible with Design Patterns or is Azure. Intent property will help me to decide which design and cloud patterns will share the common Aim, Context will help to decide the applicability of this patterns, Consequence will help me to know the positive and negative effects a particular Cloud pattern have on design patterns. And most important is the participant concept. It helps to decide which cloud services of both AWS and Azure can be used with a particular design pattern. Hence participant concept will help in a practical way to choose the Cloud Services and Patterns to be used with particular design pattern.

# 3 Methodology

In this research we are checking design patterns compatibility with cloud patterns AWS and Azure. The first important task is to find a set of design patterns and cloud patterns to do the hypothesis. The rest of the section is divided into following section. Section 3.1 gives introduction to design Patterns and reasons for selecting a Particular design patterns. In section 3.2 a brief introduction to cloud patterns and reasons are given for

selection of cloud patterns. In section 3.3 reason for selection of Method and Application is given.

## 3.1 Design Pattern

The design patterns used in this research are the 23 standard design patterns which are defined in Gamma (1995). Design patterns can be defined as the solution for the recurring problems that came while developing a software. This 23 design patterns are divided into three main types Creational, Structural and Behavioral Patterns. "Creational patterns concern the process of object creation. Structural patterns deal with the composition of classes or objects. Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility." In this research one design pattern used for research include prototype pattern from creational category and observer from behavioral category. The observer is selected because it is the most common design pattern used in application development and also Observer pattern share common properties to Singleton Design Pattern which is used while developing an application and is present in every application with or without user having any knowledge about it. The most common used structural pattern is Singleton but singleton is not used as it has common characteristics with observer pattern Hence protoype pattern is used. Another reason for selecting prototype pattern is that it has some common properties with cloud computing like parallel processing and is similar to decorator pattern which is also commonly used pattern and it covers all the properties of decorator pattern. Prototype Pattern is also related to other patterns like Abstract Factory pattern.

## 3.2 Cloud Patterns

Cloud patterns can be defined as the solution to recurring problem in cloud computing. Many researchers and cloud vendor has proposed patterns to facilitate application portability to cloud. This patterns are both platform dependent and platform independent. The patterns described by researchers who has wrote books as inFehling (2015) are platform independent. Many cloud vendors like AWS2012 (2012) and AzureHomer et al. (2014) has designed their own patterns specific to their environment. In this research AWS and Azure patterns are used to check compatibility with design patterns. The results of AWS patterns with design patterns is compared with results of Microsoft Azure patterns with design patterns. In this research the patterns of AWS and Azure are selected with one to one correspondence with each other. The selected cloud patterns are of different types and have different properties from one another. They are broadly classified into basic patterns, Patterns to increase the availability, cloud deployment patterns and patterns to handle the static and dynamic content.
The full list of AWS cloud Patterns can be found here 2012 (2012)
[1]http://en.clouddesignpattern.org The full list of Azure cloud Patterns can be found hereHomer et al. (2014).
The Cloud Patterns and their properties used in this research are from above two sources.

[1] `\unskip\penalty\@M\vrulewidth\z@height\z@depth\dp`ff

### 3.3 Selection of Method Applied

Many different techniques has been used to describe design patterns. The UML diagrams and textual descriptions of patterns have been explained in Gamma (1995). This book gives all the information about the design patterns like Intent of Pattern, Context, Implementation of Pattern, components of design pattern and effects of design pattern. This properties can be extended to be applied to cloud patterns. Many cloud providers vendors like AWS and Azure has proposed their own pattern which helps in application portability to cloud. This cloud patterns can be explained in same way as design patterns to find the intent, context ,consequences and components of cloud patterns. By describing the cloud patterns in the same way as design patterns it is possible to know the common Aim(Intent), Context, Consequences and Components of design and cloud patterns. Intent, Context, Consequence and Components are important concepts to know how much a particular Cloud and Design pattern has in common as this common characteristics will help in application portability to cloud. This properties are explained in more detail in the next section.

## 4 Design Specification

The following diagram explains the basic design of pattern with its components and also explain how components are connected with the pattern.

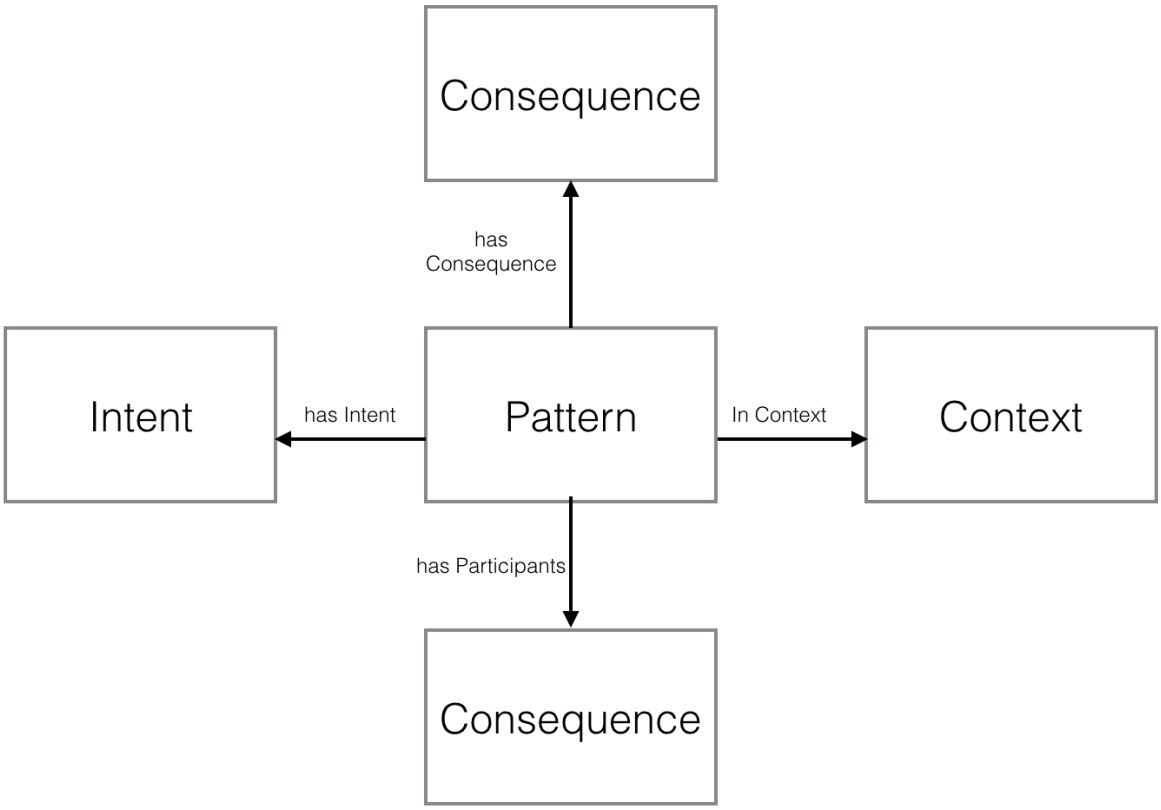


Fig1: Design

The research is based on semantic representation of patterns. The concepts included are from ODOL ontology are Intent, Context, Consequence and Participants as described in Di Martino et al. (2015) and Di Martino and Esposito (2013) This four concepts are used to describe both design and cloud patterns. The concepts are as follows: 1) Intent defines the main purpose and objectives of the patterns. Intent can also be defined as the main Aim for which pattern is described. It is connected to Pattern concept via hasIntent property.
2) Context defines the applicability of the solution that is described by the pattern. Context is important property because it help in modification of patterns. It is connected to pattern concept via a usedInContext property.
3) Consequence is defined as the result or effect of the pattern.It should fulfil the goals presented by Intent Concept derived from the Intent property of the pattern. The results can be both positive and negative. By using Consequence property we can know the benefits, limitations and side effects of the pattern.It is connected to pattern concept through hasConsequence property.
4) Participant can be defined as the entities who actively participate to implement the functionality of the pattern. They are the core components of patterns on which pattern implementation, design and working of the pattern depends. It is connected to pattern concept through hasParticipant Property. Participants are also called as components

# 5 Implementation

As stated above, the main goal of this research is to find out whether AWS Patterns or Azure Patterns are more compatible with design patterns. A score based methodology is proposed which takes into account the intent, context, consequence and participants of design and cloud patterns to find out the compatibility between design and cloud patterns. The compatibility can be found out in two ways. In the first one we are calculating rank of AWS and Azure on design pattern based on Intent, Context and Consequence properties of design and cloud patterns.In Second one we are using the participant concept for finding the matching score between the participants of design patterns and cloud patterns. This participants are the actual components of design and cloud patterns which make the implementation of this patterns possible in real life. The difference between two methods are the first methodology is developed in a more theoretical way using the general descriptions of both design and cloud patterns taking into account, like the aim and effects of patterns while in the second actual components that make this patterns to be practically implemented is taken into account.

## 5.1 Method I: Rank Calculation of Cloud Patterns on the basis of properties of Design and Cloud Patterns

The Intent, Context and Consequence Properties of Design and Cloud Patterns is used to calculate score in the following way:
1.Using the intent OWl it is possible to know which design and cloud patterns share the same intent.
2. Using the context concept it is possible to know which design and cloud patterns share the same solutions.
3. Using the Consequence concept it is possible to know about the positive and negative

effects of cloud patterns on design patterns.
Note that one design pattern may be compatible with different cloud patterns. It is many to many relationship between the design and cloud patterns.
The formulas used to calculate the Intent Score, Context Score and Consequence Score of Cloud Patterns on the basis of properties of Design and Cloud Pattern are as follows as defined in Di Martino et al. (2015):
1. The formula for Intent calculation is as follows:

$$\text{Intent Score} = \frac{\text{CorrIntents}}{\text{NIntentsDP}}$$

where CorrIntent are the common intent share by both cloud and design patterns and NintentsDp are the total number of intents of design pattern.
2. The formula for context calculation is same as that of intent

$$\text{Context Score} = \frac{\text{CorrIntents}}{\text{NContextsDP}}$$

where CorrContext are the common context share by both cloud and design patterns and NContextDp are the total number of context of design pattern. 3. The formula for consequence is as follows:

$$\text{ConseqScore} = \frac{\text{CorrConseq - AddedNegConseq}}{\text{NConseqDP+NconseqCP}}$$

where CorrConseq are the common consequences defined in both design and cloud patterns. AddedNegConseq are the total no of negative consequences that appear in cloud pattern which are not present in design patterns. NConseqDp are the number of consequences of design Patterns and NConseqCp are the number of consequence of Cloud Patterns.
The above formulas are defined in this research. Di Martino et al. (2015).

The formulas for intent and context are going to be the same. But I am proposing a slightly different formula for consequence. The proposed formula is as follows:

$$\text{ConseqScore} = \frac{\text{CorrConseq} - \text{AddedNegConseq}}{\text{NConseqDP}}$$

where CorrConseq are the common consequences defined in both design and cloud patterns. AddedNegConseq are the total no of negative consequences that appear in cloud pattern which are not present in design patterns. NConseqDp are the number of consequences of design Patterns. This formula is suggested for the equivalence of all the three properties. As we are taking design pattern as base to compare the cloud patterns hence in denominator only the total number of consequence of design pattern should be there. This formula will help to form equivalence between all the three properties of the pattern. As all the properties will be explained by same formula all properties will have equal weight-age to decide the rank. The other reason is some cloud patterns has negative consequences also. Hence subtracting the negative consequence from numerator and then also adding it in denominator will reduce the consequence score further.
The range of intent and context scores are from 0 to 1. Note But if the ConseqScore is less than 0 between the patterns then the pattern is rejected as cloud pattern will have more negative impact on particular design pattern.

Once we get the results of intent, context and consequence scores the following formula can be applied to calculate the overall score called Rank.
The following is the formula to calculate the rank.

$$\text{Rank} = a*\text{IntentScore} + b*\text{ContextScore} + c + \text{ConseqScore}$$

Here a,b,c are constants. The value of a,b,c will depend on user as some users want only the intent concept and discard context and Consequence. But the basic condition is a+b+c should be equal to 1(a + b +c = 1) as defined in Di Martino et al. (2015). In this research to keep things simple we are using

a=b=c=1/3

## 5.2 Design and Cloud Patterns Participants Matching

Participant are defined as the main components of the pattern which are needed to implement the pattern. They are the basic buildings blocks of both design and cloud patterns. It is an important concept because if you have no equivalence between components of design patterns and cloud patterns then application portability is difficult to achieve. This concept gives us information on exactly same components used in design and cloud patterns. It also helps to know exact relationship between design pattern component and cloud patterns components. Participants also help in choosing the cloud services according to the needs of design patterns. The formula for Matching the Participants of Cloud and Design pattern as defined in Di Martino et al. (2015) is as follows.

$$\text{Matching Score} = \frac{\text{Matching participants}}{\text{NParticipantsDp}}$$

where Matchingparticipants are the common participants in both design and cloud patterns and NParticipantsDp are the total number of participants Of Design pattern.

# 6 Evaluation

In this section the above procedure is matched to a set of design patterns and cloud patterns.

## 6.1 Design Patterns

The design patterns used are Observer pattern and Prototype pattern.

### 6.1.1 Observer Pattern

Definition: "Observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods." *Observer pattern* (2016) The intent, context, consequence and participants of observer pattern is described in the table No: 1.
[2]https://en.wikipedia.org/wiki/Observer$_p attern$

### 6.1.2 Prototype Pattern

"The prototype pattern is a creational design pattern in software development and it is used when the type of objects to create is determined by a prototypical instance, which is cloned to produce new objects." *Prototype pattern* (2016).The intent, context, consequence and participants of prototype pattern is described in the table No: 4.
[3]https://en.wikipedia.org/wiki/Prototype$_p attern$

## 6.2 AWS Patterns

The following is the list of AWS Patterns used:

1.Scale out pattern: How to automatically add/remove virtual resource according to computational load as needed.

2.Priority Queue pattern: Assigning of job according to priority by using queue

3.Queueing Chain Pattern: It helps the sequential task to decouple into independent tasks by using queue.

4.Multi-Datacenter Pattern: By using Multi-Datacenter it is possible to improve availability and help in disaster recovery.

5.Deep Health Check Pattern: It checks the status of services and helps in application monitoring.

[2]\unskip\penalty\@M\vrulewidth\z@height\z@depth\dpff
[3]\unskip\penalty\@M\vrulewidth\z@height\z@depth\dpff

6.Inmemory DB Cache Pattern : This pattern help in improving the performance of application by caching the data that is used frequently .

This is just brief introduction of the patterns. Detailed description of patterns is found on website of AWS. 2012 (2012)
[4]http://en.clouddesignpattern.org

## 6.3 Azure Patterns

The following is the list of AWS Patterns used:

1. Autoscaling guidance: Dynamic allocation and deallocation of resources as needed by application

2. Priority Queue Pattern: Assigning of job according to priority by using queue

3. Queue-Based Load Leveling Pattern: Improve the performance and availability of application by using queue by managing the resources during peak time and normal time

4. Multiple Datacenter Deployment Guidance: Increase the availability and gain better user experience by deploying your application to multiple Datacenters.

5. Health Endpoint Monitoring Pattern: This pattern helps in verifying that services and application are performing correctly.

6. Cache-Aside Pattern: This pattern improve performance and help in maintaining the consistency of data both in cache and data storage.

This is just brief introduction of the patterns. Detailed description of patterns can be found here Homer et al. (2014)

## 6.4 List of AWS Participants

The list of Participants are already describedDi Martino et al. (2015) and used as are as follows:

1.Simple Queue Service(SQS): SQS is a queue management service to store data, messages while travelling from one component to other,

2.Elastic Compute Cloud(EC2): EC2 is a service that provides scalable computing capacity in the form of virtual machine in the cloud.

3. CloudWatch: It is monitoring service used to check applications and services in cloud. Auto feature of notification can be set to do automatic check in application.

[4]`\unskip\penalty\@M\vrulewidth\z@height\z@depth\dp`ff

4. Autoscaling: It helps in scaling up and down EC2 instances automatically according to the need of services.

5. Elastic Load Balancer (ELB): It is used to automatically balances the load between EC2 instances according to need of application.

Further information can be found on AWS website.*Amazon web services (AWS) - cloud computing services* (2016) [5]https://aws.amazon.com

## 6.5 List of Azure Participants

1.Microsoft Windows Azure Monitoring Services Management Library: It is an API that that provides Automatic Autoscaling and also provides automatic notifications and metrics to check the load of the application

2. Microsoft Azure Web Role: It provide separate storage for data while multiple instance are running concurrently to access the same data store.

3. Message Queue: Message queue is used to balance the load between Web Role and storage services.

4. Windows Azure Service Bus: It provides a mechanism to handle the priority request of the queues.

Further Information can be found here Homer et al. (2014)

## 6.6 Case Study: No 1

In this case study we are using Observer Design Pattern and Scale Out Pattern, Oueuing Chain Pattern and Priority Queue Pattern of AWS which are compared with Autoscaling, Queue based Load Levelling and Priority Queue Pattern Of Azure.
The table No 1 shows the characteristics and properties of above Patterns.

### 6.6.1 Method I: Rank Calculation of Cloud Patterns on the basis of properties of Design and Cloud Patterns

Both the scores of consequences, based on old formula as proposed by Di Martino et al. (2015) and one based on new formula as defined are used. There are some modifications done as service decoupling in cloud patterns means loose coupling in design patterns and also liability management is same concept as resource management. Monitoring and System Monitoring are also same. This discrepancies can be solved by defining a common vocabulary for both design patterns and cloud patterns.

[5]`\unskip\penalty\@M\vrulewidth\z@height\z@depth\dpff`

Table No1:- Schematic Description of Patterns used in Case Study

| Pattern | Intent | Consequence | Context | Participant |
|---|---|---|---|---|
| **Observer Pattern** | Loose Coupling<br>Automatic Update<br>Monitoring<br>State Tracking | Loose Coupling(p)<br>Component Reuse(p)<br>Parallel Processing(p)<br>Liability management(p) | Notification System<br>Change Propogation<br>Monitoring System | Subject<br>Observer<br>Concrete Observer<br>Concrete Subject |
| **Scale Out Pattern (AWS)** | Load Balancing<br>System Monitoring<br>Resource Scaling<br>Service Decoupling | Parallel Processing(p)<br>Increase Scalability(p)<br>Service Reuse(p) | Notification System<br>Change Propagation<br>Monitoring System<br>Automatic Scaling<br>AMI | Elastic Load Balancer<br>Cloud Watch<br>Auto Scaling |
| **Autoscaling(Azure)** | System Monitoring<br>Resource Scaling | Parallel Processing(p)<br>Resource Mangement(p)<br>Horizontal Scaling(p) | Change propagation<br>Dynamic Allocation<br>Automatic Scaling<br>Monitoring System | Windows Azure<br>Autoscaling |
| **Queuing Chain Pattern(AWS)** | Service Decoupling<br>Parallel Processing<br>Failure Resiliency | Service Decoupling(p)<br>Parallel Processing(p)<br>Failure Resiliency(p) | Asynchronous Messaging | Elastic Compute Cloud<br>Simple Queue Service |
| **Queue based Load Levelling (Azure)** | Load Balancing<br>System Monitoring<br>Increase Availability<br>Service Decoupling | Increase Scalability(p)<br>Autoscaling(p)<br>Service Decoupling | Scalability<br>Asynchronous Messaging<br>System Monitoring | Microsoft Azure Web Role<br>Message Queue |
| **Priority Queue Pattern(AWS)** | Service Decoupling<br>Parallel Processing<br>Failure Resiliency | Service Decoupling(p)<br>Parallel Processing(p)<br>Failure Resiliency(p) | Asynchronous Messaging<br>Priority Management | Elastic Compute Cloud<br>Simple Queue Service |
| **Priority Queue Pattern(Azure)** | System Monitoring<br>Resource Scaling | Parallel Processing(p)<br>Dynamic Allocation(p) | Asynchronous Messaging<br>System Monitoring<br>Change Propogation | Windows Azure Service Bus |

The scores for Observer Pattern with Scale Out Pattern(AWS) in the following way:

1)Intent Score Calculation:There are total four intents defined in Observer Pattern namely Loose Coupling, Automatic Update, Monitoring and State Tracking .There are total four intents defined in Scale Out Pattern(AWS) namely Load Balancing, System Monitoring, Resource Scaling and Service Decoupling. The common intents in both are Monitoring and Coupling. Hence by the formula of intent, Intent score is 2/4= 0.5.

2)Context Score Calculation:There are total three contexts defined in Observer Pattern namely Notification System, Change Propagation and Monitoring System .There are total five contexts defined in Scale Out Pattern(AWS) namely Notification System, Change Propagation and Monitoring System, Automatic Scaling and AMI. The common context in both are Notification System, Change Propagation and Monitoring System. Hence by the formula of context, Context Score is 3/3= 1.

3) Consequence Score Calculation(new formula): There are total four consequences defined in Observer Pattern namely Loose Coupling, Component Reuse, Parallel Processing and Liability Management .There are total four consequences defined in Scale Out Pattern(AWS) namely Parallel Processing, Increase Scalability and Service Reuse. The common consequences in both are Service Decoupling and Parallel Processing. Hence

by the new formula of consequence, Consequence score is 2/4= 0.5.

4) Consequence Score Calculation(old formula): The only difference from above formula are total no of consequences of cloud and design patterns are taken into consideration. They are 7. Hence by the old formula of consequence, Consequence score is 2/7= 0.3.

5) Rank Score Calculation(New Consequence Score taken into consideration): For calculation of Rank Score the Intent Score is 0.5, Context Score is 1 and Consequence Score is 0.5. Also we are equal weightage to all the three constants a=b=c=1/3. Hence rank score by using the rank formula is = 0.5/3 + 1/3 + 0.5/3 = 0.66

6) Rank Score Calculation(Old Consequence Score taken into consideration): The intent and context score are same 0.5 and 1 respectively but consequence score in this case is 0.3. Hence rank score by using the rank formula is = 0.5/3 + 1/3 + 0.3/3 = 0.60

The same methodology and formulas are used to calculate the remaining scores of cloud patterns.
The following table show the scores of cloud patterns while comparing with observer design pattern .

Table No: 2 :- Scores calculated for examined Patterns

| Pattern comparison with observer | Intent Score | Context Score | Consequence Score(new formula) | Consequence Score(old formula) | Rank (new consequence score) | Rank (old consequence score) |
|---|---|---|---|---|---|---|
| **Scale Out Pattern (AWS)** | 0.5 | 1 | 0.5 | 0.3 | 0.66 | 0.60 |
| **Autoscaling (Azure)** | 0.25 | 1 | 0.5 | 0.3 | 0.58 | 0.52 |
| **Queuing Chain Pattern (AWS)** | 0.25 | 0 | 0.5 | 0.3 | 0.25 | 0.18 |
| **Queue based Load Levelling (Azure)** | 0.5 | 0.33 | 0.25 | 0.14 | 0.36 | 0.33 |
| **Priority Queue Pattern (AWS)** | 0.25 | 0 | 0.5 | 0.3 | 0.25 | 0.18 |
| **Priority Queue Pattern (Azure)** | 0.25 | 0.66 | 0.25 | 0.14 | 0.39 | 0.34 |

.

### 6.6.2 Results Evaluation of Method I

The first thing to take in to consideration is if the consequence score is less than zero then the cloud pattern is not compatible with design patterns. The positive effects of cloud pattern are shown by(p) and negative effects are shown by(n). Here there is one to one dependence between selected AWS pattens and Azure cloud patterns. The Scale Out Pattern of AWS is quite similar to Autoscaling Pattern of Azure. Similarly Queuing Chain Pattern of AWS is similar to Queue based Load Levelling pattern and Priority Queue Pattern of AWS is similar to Priority Queue Pattern of AWS. The most promising results are for Scale Out Pattern (AWS) and Autoscaling Pattern of Azure while Scale Out Pattern of AWS is better than (8 percent) than Autoscaling Pattern of Azure. Also Queue based Load Levelling and Priority Queue Pattern of Azure are far better(almost double) in terms of rank than Queuing Chain Pattern and Priority Queue Pattern of AWS. Also the context score is zero for Queuing Chain Pattern and Priority Queue Pattern of AWS.
The next method explains the concept of Participants. It can be used when users want to know more about the compatibility of design patterns with cloud pattern based on the components of patterns.

### 6.6.3 Method II: Score Calculation on the basis of Design and Cloud Participants Matching

The following table shows how the participants of design and cloud patterns are matched and their scores

Table No 3 :- Participants and Participants Matching Scores Of Patterns

| Observer Pattern | Scale Out Pattern(AWS) | Autoscaling (Azure) | Priority Queue Pattern(AWS) | Priority Queue Pattern(Azure) |
|---|---|---|---|---|
| **Subject** | | | | |
| **Concrete Subject** | CloudWatch<br>EC2 | Azure Application Insights | SQS<br>EC2 | Windows Azure Service Bus |
| **Observer** | | | | |
| **Concrete Observer** | ELB<br>CloudWatch<br>Autoscaling | Azure Application Insights<br>Autoscaling | | |
| **Matching Score** | 0.5 | 0.5 | 0.25 | 0.25 |

Here there can be one to many dependencies between the components of design pattern with Cloud pattern. It is dependent on how participant play their role during the execution. For example Concrete Subject Component is equivalent to CloudWatch and EC2 participants of Scale Out Pattern(AWS). As subject notifies the observer so Cloud watch is a service used .And to execute it you need computing power hence EC2 component

is equivalent to Subject of observer Pattern. But this two are compared as one unit of Matching Participant as both are matched with concrete subject while applying formula. The Matching Score Of Participants is similar both in Scale Out Pattern(AWS) and Auto-scaling (Azure) Also Priority Queue Pattern(AWS) and Priority Queue Pattern(Azure). It is also similar in third set of patterns. Hence we can infer that participant of cloud patterns(both AWS and Azure) will not help in choosing which pattern is more suitable AWS and Azure for observer design pattern. It can be also extended to other design patterns. As services in AWS and Azure are similar as defined in Winkle (2015). Hence the rank system previously explained based on intent, context and consequence property only can be used to infer which design patterns are suitable for which cloud patterns of AWS or Azure.
[6]http://www.tomsitpro.com/articles/azure-vs-aws-cloud-comparison,2-870-2.html

## 6.7 Case Study: No 2

In this case study we are using Prototype Design Pattern and Inmemory DB Cache Pattern, Deep health Check Pattern and MultiData Center Pattern of AWS which are compared with Cache Aside Pattern, Health End Point Monitoring Pattern and MultiData Center Deployment Pattern Of Azure.
The following table shows the characteristics and properties of above Patterns.

Table No4:- Schematic Description of Patterns used in Case Study No: 2

| Pattern | Intent | Consequence | Context | Participant |
|---|---|---|---|---|
| **Prototype Pattern** | Dynamic Allocation<br>System Monitoring<br>Reusable Components | Flexibility(p)<br>Component Reuse(p)<br>Simplicity(p) | Dynamic allocation<br>Parallel Processing<br>Monitoring System | Prototype<br>Concrete Prototype<br>Client |
| **Inmemory DB Cache(AWS)** | Improve Performance<br>Reusable Components | Load balancing(p)<br>Improve Performance(p) | Automatic Recovery<br>Change Propagation | Elastic Compute Cloud<br>Relational Database<br>Elastic Cache |
| **Cache Aside Pattern(Azure)** | Autoscaling<br>Improve Performance | Load balancing(p)<br>Improve Performance(p)<br>Data Consistency(n) | Change propagation<br>AutoScaling | Windows Azure Cache |
| **Deep health Check Pattern(AWS)** | Load Balancing<br>System Monitoring | System Monitoring(p)<br>Single Point of Failure (n) | Monitoring System<br>Parallel processing | Elastic Load Balancing<br>Health Check Program with DB access |
| **Health End Point Monitoring Pattern (Azure)** | System Monitoring | System Monitoring (p) | System Monitoring<br>Parallel Processing | Windows Azure Traffic Manager<br>Microsoft System Centre Operations Manager |
| **MultiData Center Pattern(AWS)** | Increase Availability<br>Improve Security | Disaster Recovery(p) | Fault Tolerance<br>Load Balancing | Elastic Compute Cloud<br>Elastic Load Balancing |
| **MultiData Center Deployment (Azure)** | Increase Availability<br>Increase Scalability | Disaster Recovery(p)<br>Increase Cost(n) | Improve Performance<br>Load Balancing | Windows Azure Traffic Manager |

[6]\unskip\penalty\@M\vrulewidth\z@height\z@depth\dpff

### 6.7.1 Method I: Rank Calculation of Cloud Patterns on the basis of properties of Design and Cloud Patterns

The score of cloud patterns with prototype design pattern is carried out in the same way as explained in Method I of Case Study No 1.
The following table show the scores of cloud patterns while comparing with prototype design pattern .

Table No: 5 :- Scores calculated for examined Patterns in Case Study No: 2

| Pattern comparison with Prototype | Intent Score | Context Score | Consequence Score(new formula) | Consequence Score(old formula) | Rank (new consequence score) | Rank (old consequence score) |
|---|---|---|---|---|---|---|
| **Inmemory DB Cache (AWS)** | 0.33 | 0 | 0 | 0 | 0.11 | 0.11 |
| **Cache Aside Pattern (Azure)** | 0 | 0 | -0.33 | -0.17 | -0.11 | -0.6 |
| **Deep health Check Pattern (AWS)** | 0.33 | 0.66 | -0.33 | -0.2 | 0.22 | 0.26 |
| **Health End Point Monitoring Pattern (Azure)** | 0.33 | 0.66 | 0 | 0 | 0.33 | 0.33 |
| **MultiData Center Pattern (AWS)** | 0 | 0 | 0 | 0 | 0 | 0 |
| **MultiData Center Deployment (Azure)** | 0 | 0 | -0.33 | -0.2 | -0.11 | -0.07 |

### 6.7.2 Results Evaluation of Method I

Here there is one to one dependence between selected AWS pattens and Azure cloud patterns. The Inmemory DB Cache of AWS is quite similar to Cache Aside Pattern of Azure. Similarly Deep Health Check Pattern of AWS is similar to Health End Point Monitoring Pattern of Azure and MultiData Center of AWS is similar to Multi Data Center Deployment of Azure. The first thing to take in to consideration is if the consequence score is

less than zero then the cloud pattern is not compatible with design patterns. The positive effects of cloud pattern are shown by(p) and negative effects are shown by(n). Hence it rules out Cache Aside Pattern(Azure), Deep Health Check Pattern(AWS) and Multidata center Deployment(Azure) Hence Inmemory DB Cache Pattern of AWS, Health End point Monitoring of Azure . So Inemory DB Cache pattern(AWS), Health End Point Monitoring Pattern(Azure) are cloud patterns that are compatible with prototype design pattern. Also note that MultiData Center Pattern(AWS) can be used with prototype pattern but it doesn't share any characteristics with protoype.
The Participant Concept is not used here as out of three above sets of AWS and Azure Patterns there is no set of cloud Patterns in AWS and Azure that are both compatible with prototype pattern

### 6.8 Conclusion from Case Studies

From the results of Case Study No.1 and No.2 it can be inferred that sometimes some patterns AWS will have more compatibility with particular design pattern and sometimes Azure will have more compatibility pattern. In case study No 1, Scale Out Patten(AWS) is more compatible with observer design pattern than Autoscaling(Azure) while Queue based Load Levelling and Priority Queue Pattern of Azure are better than Queuing Chain Pattern and Priority Queue Pattern of AWS. In case Study No 2, only Inmemeory DB Cache(AWS) is compatiable with prototype design pattern while Cache Aside Pattern(Azure) is not. Similarly Health End Point Monitoring Pattern(Azure) is compatiable with prototype while Deep Health Check Pattern(AWS) is not.
Hence it depends on particular type of design or cloud pattern. It can't be concluded that AWS is superior than Azure or vice versa.
Also from the results of Case Study No.1, it can be inferred that participants of AWS and Azure Patterns will not help in choosing which Patterns of AWS and Azure Patterns are more compatible with design patterns as Matching Score is same in all cases because for every service available in AWS there is similar service available in Azure.

## 7 Conclusions and Future Work

In this Paper, two types of methodologies were examined based on ODOl ontology in order to find which cloud pattern(AWS or Azure) is more compatible with the design patterns based the characteristics, properties and components of design and cloud patterns. In the first method,(Rank Calculation on the basis of on the basis of Design and Cloud pattern) a score based methodology was used in which intent, context and consequence properties of design and cloud pattern were taken into consideration .The result for this implementation showed that every design and cloud pattern has their unique characteristics .It is not possible to decide whether particular AWS or Azure Cloud Provider patterns are more compatible with design patterns. In some cases the results shows that AWS patterns are more compatible with design patterns than Azure patterns and vice versa. In Second type of methodology, components called as participants of design and cloud patterns are compared . The result shows that components(participants) of both AWS Cloud and Azure are quite similar and hence components(participants) of patterns can't be a factor to choose between the services of AWS and Azure. A GUI has been build to show the results in graphical way.
The whole process is conducted manually and future work can be done to completely

automate the process. Automation can be done by using a software that can parse the given description of pattern and able to find it properties and components automatically. The mapping process between design and cloud patterns also can be automated using matchmaking algorithm.

## Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Manuel Tova-Izquierdo. His friendly attitude was as an added advantage which help me keep my calm throughout the project. I would also like to thank Vikas Sahni for giving me direction and guiding me in right path.

## References

2012, A. W. S. L. (2012). Aws cloud design patterns.
**URL:** *En.clouddesignpattern.org*

*Amazon web services (AWS) - cloud computing services* (2016).
**URL:** *https://aws.amazon.com*

Di Martino, B., Cretella, G. and Esposito, A. (2015). Mapping design patterns to cloud patterns to support application portability: a preliminary study, *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ACM, p. 50.

Di Martino, B. and Esposito, A. (2013). Towards a common semantic representation of design and cloud patterns, *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, ACM, p. 385.

Fehling, C. (2015). Cloud computing patterns: identification, design, and application.

Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*, Pearson Education India.

Homer, A., Sharp, J., Brader, L., Narumoto, M. and Swanson, T. (2014). *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*, Microsoft patterns & practices.

*Observer pattern* (2016).
**URL:** *https://en.wikipedia.org/wiki/Observer_pattern*

*Prototype pattern* (2016).
**URL:** *https://en.wikipedia.org/wiki/Prototype_pattern*

Winkle, W. V. (2015). Side-by-side feature & services comparison - microsoft azure vs. amazon web services: Cloud comparison.
**URL:** *http://www.tomsitpro.com/articles/azure-vs-aws-cloud-comparison,2-870-2.html*

# 8 Appendix

1)A GUI has been build to showcase the result. It is the graphical demonstration of the research conducted.

2)It is built in Java Language

3)To run this file you need JDK (Java Development kit ) or you can copy the project and also run it in Integrated development like NetBeans or Eclipse

4) The image path has to be given as input to see the results. It depends on where the image is stored in the computer. The following line of code needs to be changed according to different computers

5) BufferedImage image1 = ImageIO.read(new File("/Users/apurvdeshpande/Desktop/GUIdiagrams/D

6)The syntax written in double quotes need to be changed according to location of image in the computer.

The following figure is the snapshot of GUI build. Here two design patterns are seen Observer and prototype. The following are the properties of Observer and Prototype Pattern.