Fault Tolerant, Self-Healing and Vendor Neutral Multi-Cloud Patterns and Framework Focusing on Deployment and Management

by

Andrey Rybka

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

School of Computer Science and Information Systems

Pace University

October 2017

Version 2.5

We hereby certify that this dissertation, submitted by Andrey Rybka, satisfies the dissertation requirements for the degree of *Doctor of Professional Studies in Computing* and has been approved.

_____-_____

Dr. Lixin Tao Date

Chairperson of Dissertation Committee

_____-_____

Dr. Charles Tappert Date

Dissertation Committee Member

_____-_____

Dr. Ronald Frank Date

Dissertation Committee Member

School of Computer Science and Information Systems

Pace University 2017

Abstract

Fault Tolerant, Self-Healing and Vendor Neutral Multi-Cloud Patterns and Framework Focused on Deployment and Management

by

Andrey Rybka

Submitted in partial fulfillment

of the requirements for the degree of

Doctor of Professional Studies in Computing

October 2017

Many organizations are looking to migrate to the cloud and looking for the best way to do it securely, reliably and without vendor lock in. Most organizations have to pick a cloud provider that uses proprietary APIs and Software. Most vendors currently do not implement any cloud API standards i.e. TOSCA or OASIS CAMP. Therefore, to date, the standard approaches to cloud computing have not been successful. In addition, cloud providers experience outages, frequently with serious business impact – so fault tolerance in cloud environment still needs more research and clear and prescriptive guidance. Variable performance is also an issue because most cloud providers overprovision their virtualized infrastructure and it results in degradation of performance and quality of service for customers depending on overprovisioning factor set by the cloud provider.

This study focuses on development of multi-cloud vendor neutral framework and patterns that deliver non-proprietary APIs and Software above IaaS layer with the functionality that will be on par with proprietary software or service offered by an individual cloud provider. We demonstrate how to design Fault Tolerant, Self-Healing, Performant, Secure and Cost Efficient Deployment using Patterns in the Multi-cloud environment without vendor lock in. We detail and catalog the developed solutions to common multi-cloud problems via patterns and multi-cloud framework using open source software which ensures portability across cloud providers. Framework and patterns can be re-produced by setup scripts, code and examples which are provided in the accompanying code repository. All of the failure scenarios are validated and demonstrated clearly showing the fault tolerance and limits of each solution.

Acknowledgements

First I would like to thank my advisor - Dr. Lixin Tao for all for all the help, patience, support and guidance provided over the years. In addition, I am deeply grateful to my family for their patience and support over all these years. I would also like to thank dissertation committee members Dr. Ronald Frank and Dr. Charles Tappert as well as DPS faculty who have helped me a great deal on this journey.

Table of Contents

List of Figures

Chapter 1 - Introduction

Most organizations are moving to the cloud or planning to move to the cloud. However, it quickly becomes clear that it's usually not a simple lift and load process moving from on-premises to the cloud.

So, what is one to do when they move to a cloud?

There are multiple stakeholders usually involved in this process:

- *Developers* just want to write code and deploy applications.
- *Operations and Support* professionals want to make sure that the solution is fault tolerant, not brittle and everyone can sleep well at night.
- *Security professionals* want to make sure there are no breaches and data is protected.
- *Business users* generally just want good SLA (Service Level Agreement) without any downtime for the cheapest price possible.

Some initial challenges and questions that are usually asked:

Which cloud provider should I choose?

Should I move to Amazon AWS, Azure, Google or someone else?

Is it just about a price? Should I go with the cheapest option?

What about data safety and security certifications?

My application or process needs a software stack i.e. J2EE, .NET etc. – how do I set this up in the cloud? How do I do it securely? How do I ensure there is no downtime/faults? More importantly, most organizations ask: I would like to move application X or process Y to the cloud – what is the best way of doing it?

Should I pick Infrastructure as a Service (IaaS) or Platform as a Service (PaaS)? Let's briefly illustrate the difference between the two:

Infrastructure as a Service vs. Platform as a Service

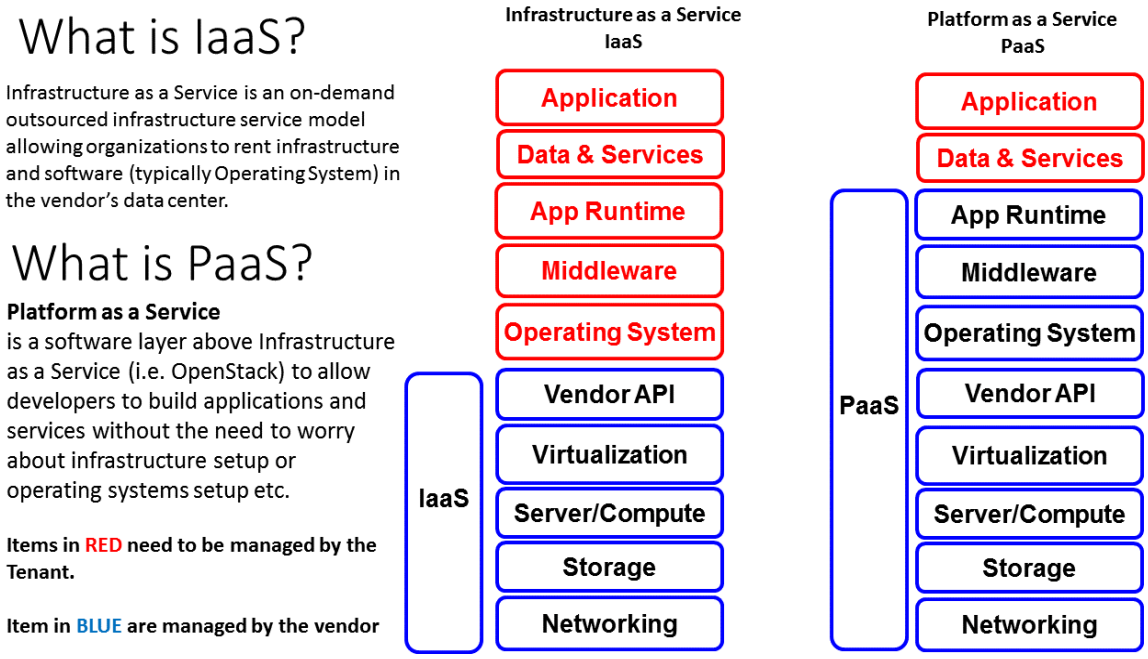
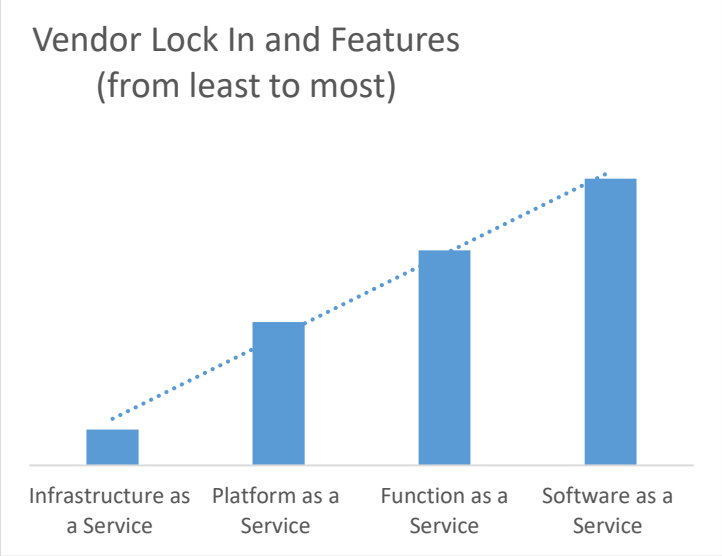


Figure 1 - Infrastructure as a Service vs. Platform as a Service

At this point, confused customers usually go to all cloud providers and every cloud provider promises that all of these will be addressed and they have excellent tools to help.

However, the reality is pretty grim. Even if a customer gets all of the questions answered and picks one cloud provider they end up locked in as most of these tools are proprietary and most cloud providers do not have or support standardized APIs.

The more features, automation and abstractions vendor provides usually results in more of lock-in for the customer. Here is high-level trend of vendor lock in of cloud offerings from least to most:



Infrastructure as a Service (IaaS): customer gets to control Resources, Virtual Machines/Compute, Networking, Storage. Example: AWS EC2, GCE.

Platform as a Service (PaaS): customer gets more features, but has less control than IaaS and more lock in. Example: Heroku.com

Function as a Service (FaaS): customer gets more features than PaaS but even less control than PaaS with even more lock in. Example: AWS Lambda, Azure Functions

Software as a Service (SaaS): customer gets more features and even less control than FaaS with even more lock in. Example: Salesforce

So is there a better way? It's usually very hard to quantify "better". However, one can argue the better way is to use a solution to a common problem which is generally defined as a pattern. In this research, we will provide patterns for common problems that organizations face when migrating to a cloud which ultimately will comprise multi-cloud framework. In addition, we will focus on building on lowest lock in layer which will be Infrastructure as a Service.

Originally most of the design patterns were software design patterns – i.e. Gang of Four Design Patterns. In this study, we will focus on infrastructure software deployment and design patterns.

What is a software infrastructure deployment pattern? It is a general reusable solution to a commonly occurring problem within a given context in software deployment. By infrastructure software we mean any software that is not application for example web service, application server software, database etc. This is also sometime referred to as software stack. Let us look at illustration of what Infrastructure Software or Stack is:

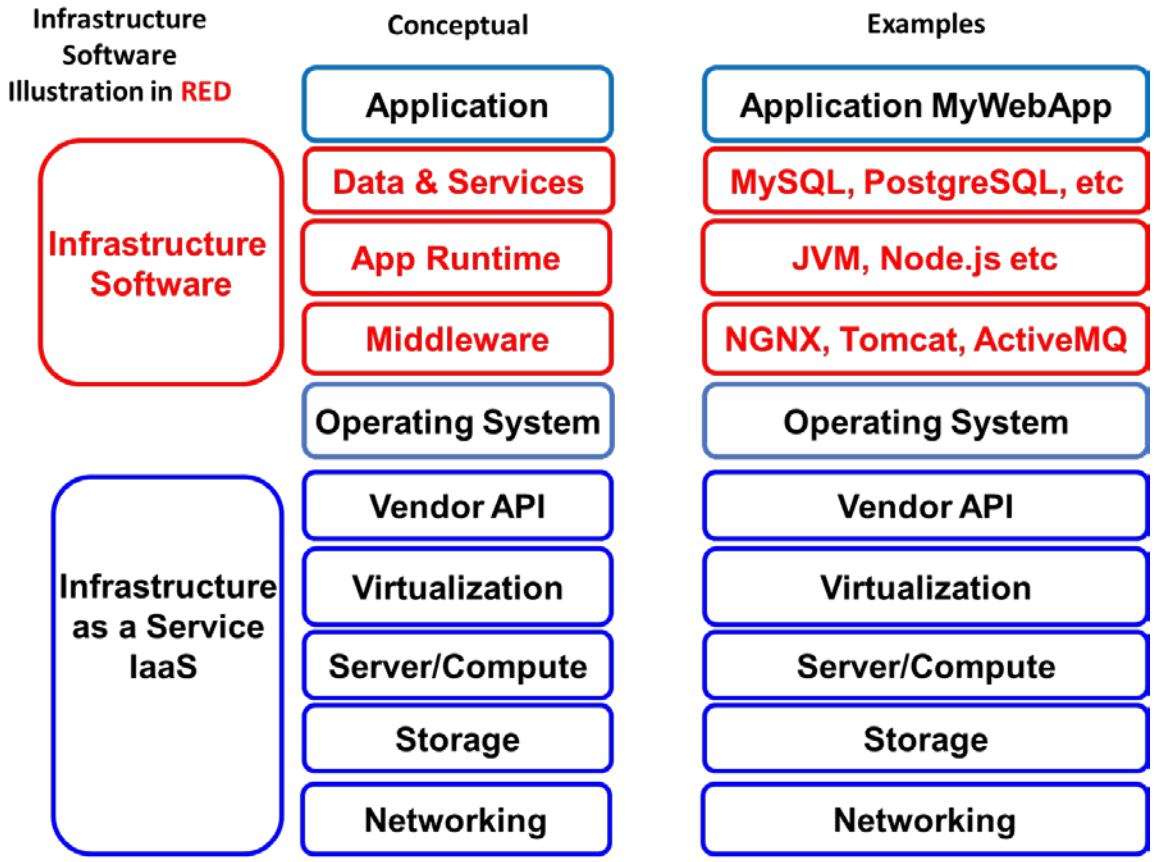


Figure 2 - Infrastructure Software

Applications need a specific software infrastructure stack to run on. Some of the common stacks can be classified as: Dynamic Web Application Stack, Media Streaming, Content Serving, Data Analytics Stack etc. These are examples of deployment stack patterns.

1.1 Multi-Cloud Deployment for Fault-Tolerance, Performance, Security and Cost Efficiency

What is Multi-cloud deployment? It is essentially the use of multiple cloud computing resources/services seamlessly as one coherent architecture. This might include Public and Private Cloud, as well as mix of SaaS, PaaS and IaaS computing resources.

Why does multi-cloud approach matter?

Fault Tolerance and High Availability

Every cloud provider experiences outages. These might be contained to one particular location, but in many cases those locations serve a very large group of clients for example U.S. Northeast or Europe. Many outages have demonstrated that high density deployments in one region with one cloud result in high impact outages to many customers. [Appendix A]. There are even services that track cloud outages in real time https://cloudharmony.com/status. (Valid by July 31, 2017)

Here the summary of total hourly impact of outages in 2015 by major cloud providers:

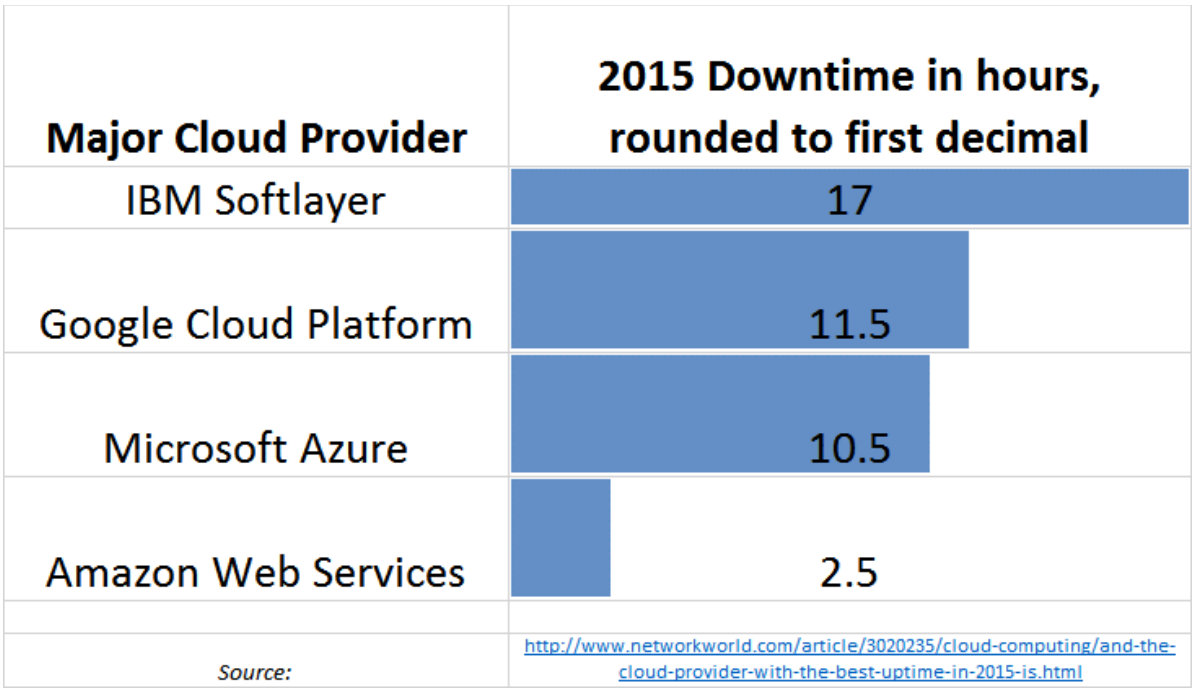


Figure 3 - 2015 Major Cloud Provider Outage in Hours

This is just a one year example - please reference a more detailed list of major outages of cloud providers for past 3 years:

Appendix A – Major Cloud Provider Outages

Performance

Performance generally has to do with how fast computer systems can perform a task X but more precisely it has to do with following computer system quality attributes: response time, throughput, latency and scalability.

One of the core issues is that not every provider has the same performance profile and at any point of time it might change due to multi-tenant nature of the cloud and because

most of workloads and component are virtualized. For example, let's take hypervisor which runs multiple virtual machines. You will most likely be sharing a hypervisor with multiple tenants and this results in a noisy neighbor problem – where one tenant might negatively impact performance for another tenant sharing the hypervisor.

In addition will focus on ability to scale Up / Vertically within the same Virtual Machine Instance and Out / Creating new instances / components.

Locality

Not every cloud provider has presence in all countries and in many cases there are different regulations that might drive the need to be in specific country. Latency and proximity matters to many use cases. In many cases you might want to run some workloads on premises and some in the cloud which can be described as Hybrid Private/Public Cloud deployment pattern.

Here is a high-level example of Multi-Cloud or Hybrid Deployment Pattern:

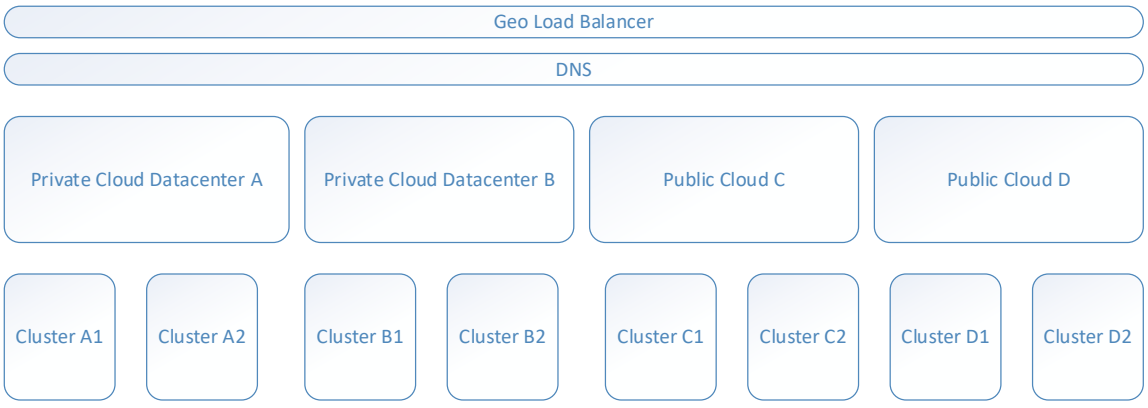


Figure 4 - Multi-Cloud Hybrid Deployment Pattern

Security

Another extremely important factor in any cloud deployment needs to be security. Cloud Providers have different services that solve this, but most of these are non-standard and proprietary which result in cloud provider lock in and customer confusion. One of the key areas we will focus on is how to protect secrets and data at rest. Security arguably should be part of every cloud pattern.

Cost Efficiency

Cloud providers change their prices quite frequently and generally the trend is to lower the price of services. In addition, there are discounted instances that cloud providers offer due to idling capacity. Multi-cloud approach allows you to shift workloads to leverage cheaper prices depending which cloud provider offers the better price. Furthermore, this body of work will demonstrate how this can be dynamically.

1.2 Current Solutions and Their Limitations

There are multiple cloud providers and some provide their own specific deployment patterns.

However, there are couple of problems with vendor provided patterns:

1). Patterns provided are usually vendor specific with specific APIs and software stacks that lock in the customer. The vendor generally has no insensitive to support standards so the customer can leave at will to another provider. Furthermore, some providers such AWS have stated that they will not support any partner that offers solution that runs on competitor's cloud platform. [36]

2). Generally, these proprietary services also cost more due to the fact that vendor can charge more for proprietary technology and convenience.

3). These patterns do not always take in account important factors such as system resilience properties i.e. fault tolerance.

4). Other patterns are very high level and theoretical and do not demonstrate the concrete software technology implementation in vendor neutral way i.e. with open source software which make these not very useful.

This leads to vendor lock in, poor deployment design and is prone to faults and poor service availability ultimately resulting in poor end user experience and data loss.

Let’s take a look at Microsoft Azure Big Data offerings – all components with exception of HortonWorks Data Platform and HBase are proprietary:

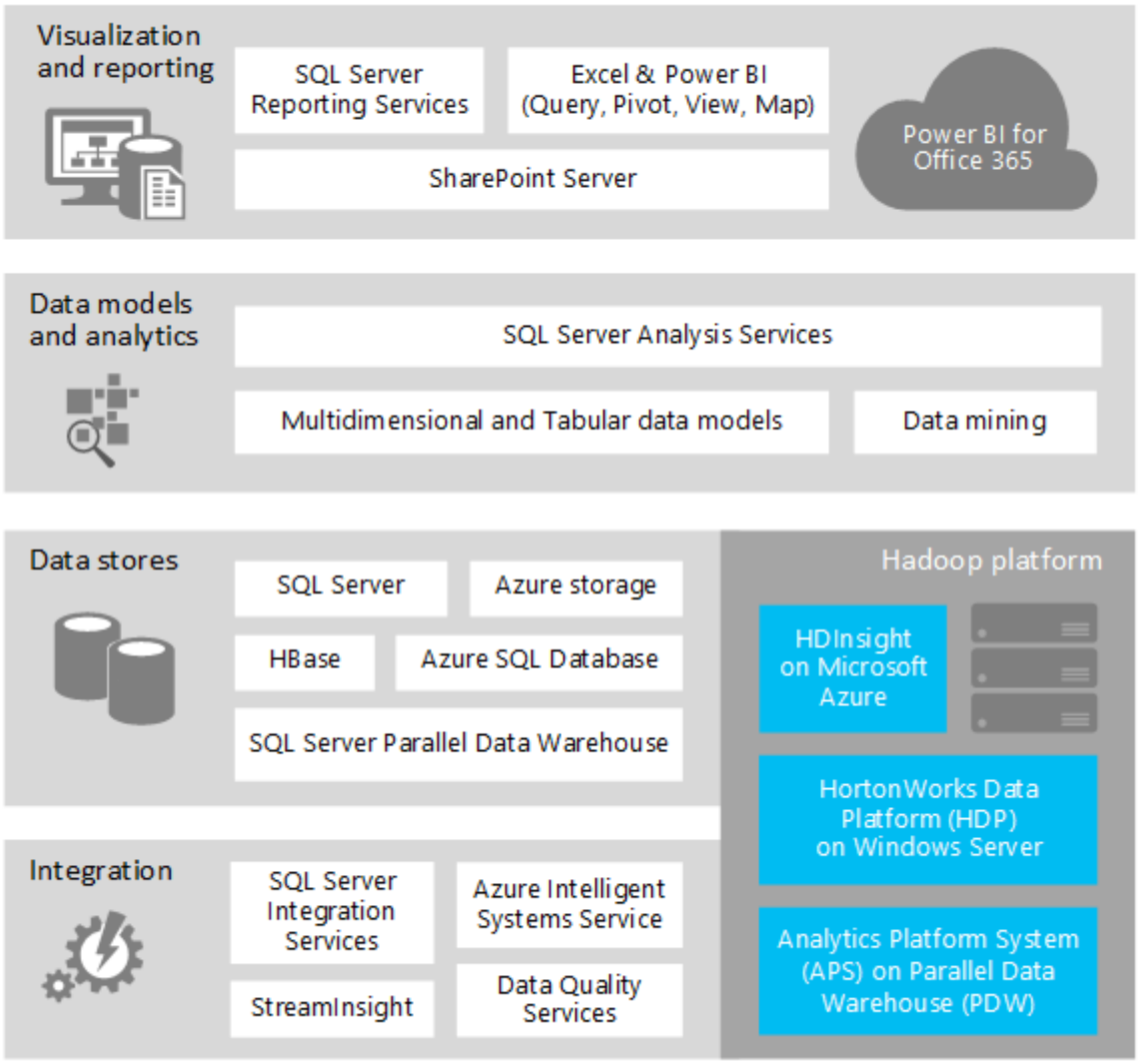


Figure 5 - Microsoft Azure Big Data Offerings Circa 2016

Source: https://msdn.microsoft.com/en-us/library/dn749804.aspx

We are not trying to pick on specific cloud provider - let's take a look at Google Offering for Big Data specifically for Mobile Gaming Analytics Platform.

Please note that all of the components provided are proprietary:

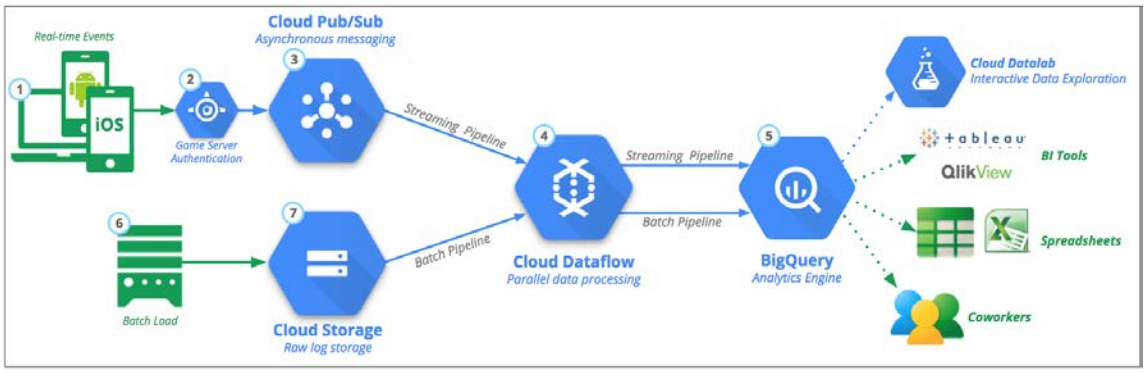


Figure 6 - Google Proprietary Big Data Offerings

Source: https://cloud.google.com/solutions/mobile/mobile-gaming-analysis-telemetry

Next let us illustrate the problem with one of the most popular cloud provider patterns – Amazon Web Services for Time Series Processing:

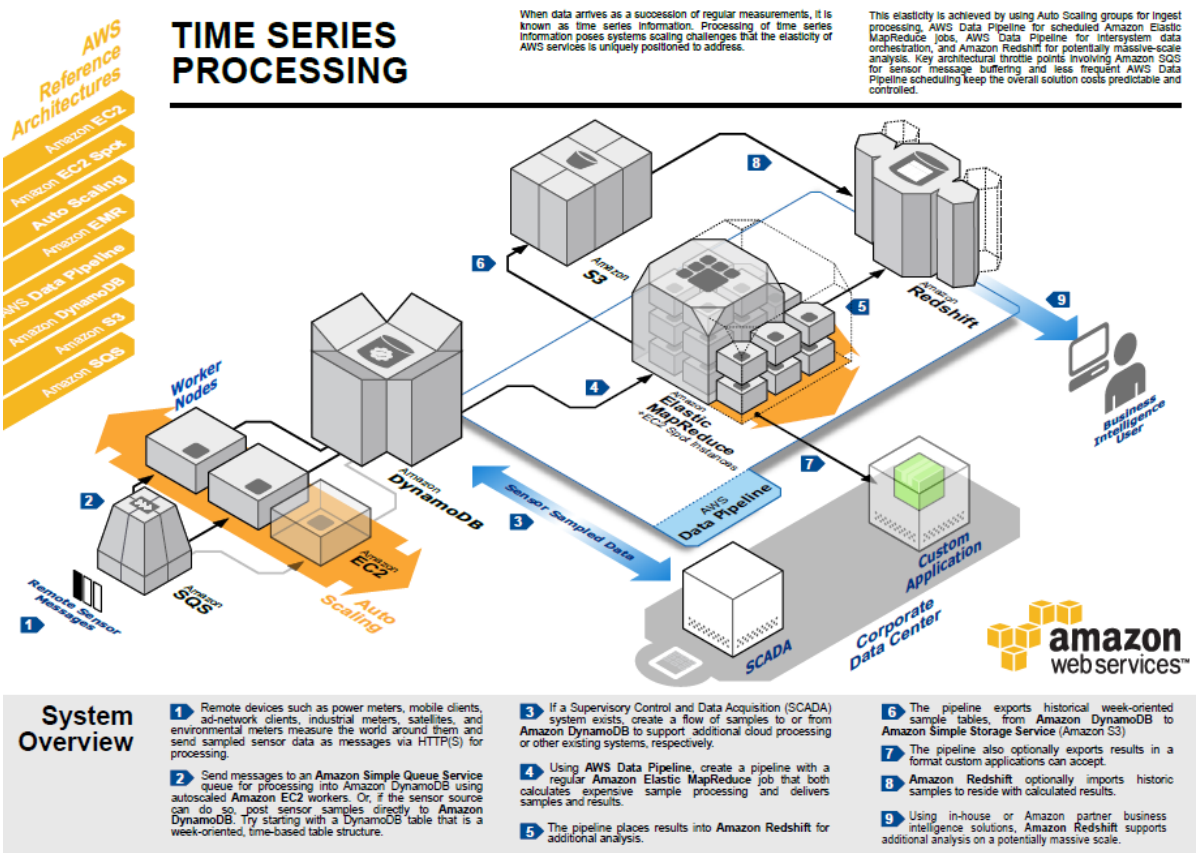


Figure 7 - Time Series AWS Specific Pattern

Source:

https://media.amazonwebservices.com/.../AWS_ac_ra_timeseriesprocessing_16.pdf

Note couple of issues with this:

1). Amazon only proprietary and non-standard components in this diagram are:

- Amazon SQS – proprietary and non-standard
- Dynamo DB – proprietary and non-standard
- AWS Pipeline – proprietary and non-standard
- Elastic MapReduce – proprietary and non-standard
- RedShift – proprietary and non-standard

- Amazon S3 – proprietary and non-standard

2). The diagram does not seem to factor in Security, Resilience and Fault Tolerance – i.e. multi-availability/multi-data center deployment. Failures can occur at multiple levels: hardware, virtual machine, network, availability zone and even data center. How do we recover from these? Ideally this should be fully automated without any human involvement.

3). The above diagram also does not factor in multi-cloud deployment mentioned before.

What would the Time Series/Internet of Things pattern look like improved over the AWS diagram mentioned above?

Let's start with decomposing the pattern with components that are Vendor Neutral and next we will take a look at how to validate it with open source software.

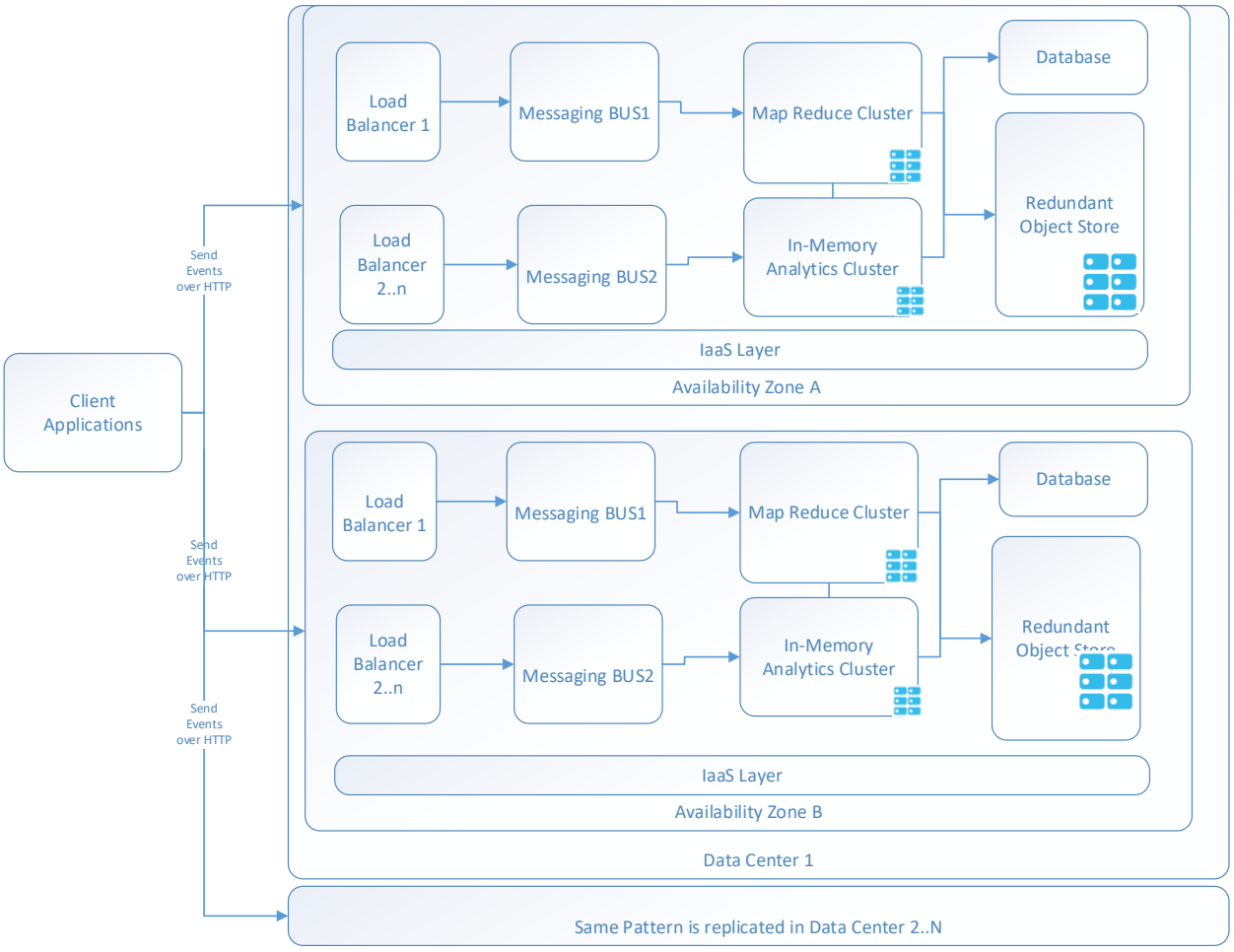


Figure 8 - Vendor Neutral Time Series Pattern

1.3 Problem Statement

Many organizations are looking to migrate to the cloud and looking for the best way to do it securely, reliably and without vendor lock in. Most organizations have to pick a cloud provider that uses proprietary APIs and Software. Most vendors currently do not implement any cloud API standards i.e. TOSCA [39] or OASIS CAMP [41]. Therefore, to date the standards approach to standardize cloud computing have not been successful. In addition, cloud providers experience outages, frequently with serious business impact – so fault tolerance in cloud environment still needs more research and clear and prescriptive guidance. [Appendix A].

Variable performance is also an issue because most cloud providers overprovision their virtualized infrastructure and it results in degradation of performance and quality of service for customers depending on overprovisioning factor set by the cloud provider. This has been well documented and various benchmark studies have been done. [42]

This study will focus on framework and patterns that deliver non-proprietary APIs and Software above IaaS layer with the functionality that will be on par with proprietary software or service offered by individual cloud provider. This study aims to answer the following questions:

How do we design Fault Tolerant, Performant, Secure and Cost Efficient Deployment Patterns in Multi-cloud environment without vendor lock in?

What are the common patterns that can help to solve this problem?

This research will focus on a framework and clouds patterns for multi-cloud deployment covering:

(a) Fault-tolerance – involving more than one cloud provider

(b) Vendor neutrality – ability to run on major cloud provider's IaaS using open-source solutions to avoid lock-in

(c) Performance - ability to shift workloads to another cloud provider if there is performance degradation of quality of service on one or more of providers

(d) Security – we will cover security related patterns that we allow you to host mission-critical components and data on premises

(e) Lastly Cost efficiency – ability to take advantage of lower priced compute resources in multi-cloud environment

Open-source implementation of the platform neutral patterns will be developed to validate patterns and guide adoptions. Using open source software to demonstrate how the patterns can be implemented makes this study more useful than generic patterns that are described here: http://www.cloudpatterns.org

Cloud provider costs vary and cost of switching could be very high if everything have to be re-designed from scratch. Using these "vendor neutral" patterns the goal would be to be able to change cloud providers at will or run the on premises without changing the software stack.

Another major benefit of this study is emphasis on fault tolerant patterns which implies that by the picking the pattern described there will be implied guarantee of system uptime and adaptation to stress or failure. The central premise here is that system resilience should be built by default.

In addition, this study will focus first on more complex patterns that were not widely studied yet - related to Big Data and Internet of Things first and will provide novel patterns that have not been yet documented.

Lastly, we will make sure these patterns can run on premises in a private cloud, because

- long term cloud cost can be higher than on premises due to operating expenses in the cloud eventually will cover the capital costs of servers and other equipment if you bought these
- for security/regulatory reasons it might be necessary to keep some workloads on premises
- performance with on premises can be better because you control the oversubscription rate and workloads that get to run on the same virtualized instance

1.4 Solution Methodology

We are going to identify and categorize the main challenges to achieve the main objective.

Next we will identify the main desired attributes for quality cloud deployment patterns.

After that we will propose quality solution patterns for each of them.

All of the patterns will be unified into one cohesive multi-cloud framework.

Next we will validate the key patterns with experiments for feasibility of all cases of fault tolerance and self-healing using open-source implementations patterns.

We will use the following standard template to be used for pattern documentation

- **Pattern Name and Classification:** A descriptive and unique name that helps in identifying and referring to the pattern.
- **Also Known As:** Other names for the pattern.
- **Problem** – problem description and applicability
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.
- **Structure:** A graphical representation of the pattern. (Component diagrams and Interaction diagrams may be used for this purpose.)

- **Participants:** A listing of the Software and Infrastructure Components used in the pattern and their roles in the design.
- **Collaboration:** A description of how component used in the pattern interact with each other.
- **Consequences:** A description of the results, side effects, and trade-offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Presents vendor independent logical pattern with specific technology implementation
- **Sample Template:** An illustration of how the pattern can be used using a deployment template.
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.
- **Strength:** every pattern might have different strength and applicability such as better at performance, security, big data volume, fault-tolerance etc.

1.5 Expected Contributions

This study will contribute:

- Multi-cloud deployment framework that is comprised of multi-cloud infrastructure patterns.

- Catalog of Multi-Cloud Patterns for various stacks and solutions using open source software.

- Every pattern will account for fault-tolerance, performance and security by default.

- As an illustration some patterns will also provide vendor neutral initial deployment that can be used for as-is deployment on private cloud or in the public cloud.

How do we prove that proposed patterns are realistic and competitive? The approach will be to:

- First we will define a few desired attributes
- Next we will use open-source software if available to show feasibility
- Lastly we will use selected prototypes to show how the desired attributes were satisfied

Next let's take a look at more detailed illustration of what this means. How would the pattern referenced in Figure 7 - Time Series AWS Specific Pattern above look if we map this Open Source software and deploy in multi-cloud environment with basic fault tolerance in mind?

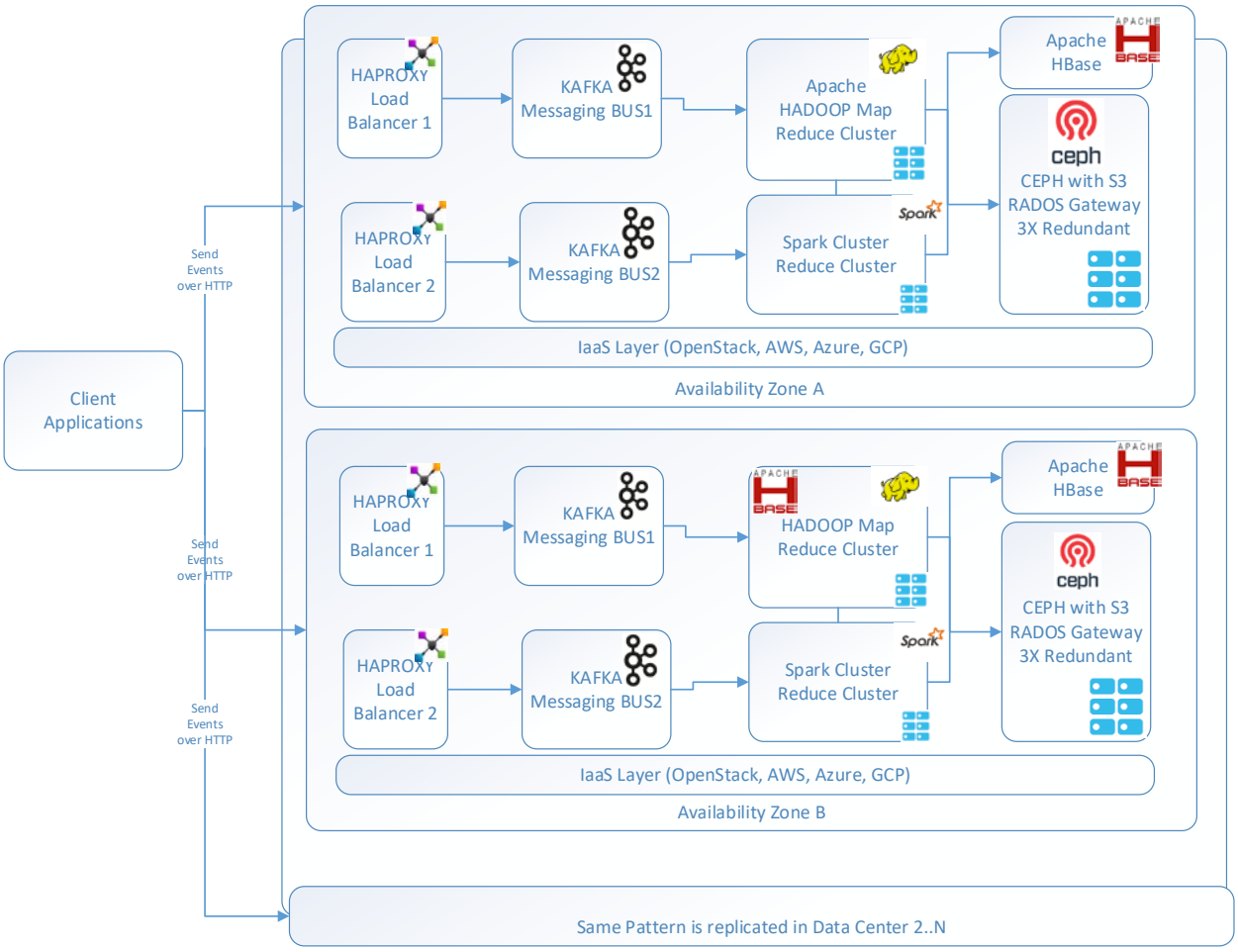


Figure 10 - Vendor Neutral Time Series Pattern with Open Source Software

Note Open Source Component Replacing Proprietary Components:

- Amazon SQS replaced by Open Source HAProxy and Kafka
- Dynamo DB replaced by Open Source Apache HBase – Part of Apache Hadoop Cluster
- Elastic MapReduce replaced by Apache Hadoop Map Reduce framework
- Amazon S3 – Replaced by Open Source CEPH with S3 RADOS Gateway

There is still more room for improvement, but this should give a good illustration of where this research might be headed.

The result of the study will be a Fault Tolerant, Secure and Highly Performing Cloud Pattern Catalog with mapping to open source software with examples of the deployment templates that can be used to stand up full infrastructure stack that will be OASIS Cloud Application Management for Platforms (CAMP) specification compatible. (https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=camp) and possibly later on OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) (https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca)

It would be another improvement to take this one step further by transforming the diagram into TOSCA template in YAML format

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Template for deploying a single server with predefined properties.

topology_template:
  node_templates:
    my_server:
      type: mytenancy.nodes.Compute
```

```
      capabilities:
       # Host container properties
       host:
        properties:
          num_cpus: 2
          disk_size: 8 GB
          mem_size: 8096 MB
       # Guest Operating System properties
       os:
         properties:
           # host Operating System image properties
           architecture: x86_64
           type: Linux
           distribution: ubuntu
           version: 14.04
….
```

So end to end flow will be:

1. Pattern Diagram Model	>>>	2. Deployment Blueprint/Template + VM/Container Image	>>>	3. Cloud Deployment on premises or public cloud

Next all the patterns will be validated using Open Source Software

Most of cloud offerings will not be possible without open source software let's just look at one example: hypervisor core of virtualization – most cloud providers use either Xen or KVM Linux based hypervisors.

Marc Andreessen the founder of Netscape has famously said: "Software is eating the World." Arguably it's more like Open Source Software is eating the World! Why?

- Cost – free scales much better than commercial. It will be extremely costly to build a cloud using only proprietary software.

- FREEdom - Ability to influence the features, roadmap, fix the bugs without waiting on a vendor X

- Community – Strength in numbers – many contributors, many companies

- Continuity - Open Source survives any vendor

- Stronger security - independent research validate that open source code has fewer defects per thousand lines of code than proprietary software code

- Happier developers – developers get to participate in the community and able to talk about contributions.

How do we pick the right open source project? We will need to answer some of the following questions:

- How Large and Active is Community?

- How many active committers?

- How many of contributors from different organizations or this is one company open source project?

- Is This Read Only or Read/Write Open Source? Some companies open source software but do not accept contributions from outsiders.

- Ideally should have a commercial company backer of the project so that you can pay for bug fixes per incident if your company doesn't have enough expertise or reputation.

1.6 Dissertation Roadmap/Outline

Now that we covered introduction let's take a look at brief outline of what's to come

2. Chapter 2 - Survey of Relevant Research

3. Chapter 3 – Patterns and Open-Source Framework for Effective Multi-Cloud Deployment

4. Chapter 4 - Multi-Cloud Deployment Design Patterns Based on Open-Source Technologies

5. Chapter 5 - Experimental Validation

6. Chapter 6 – Conclusion and Future Work

7. Reference list

1.7 Conclusion

In the introduction, we have briefly covered the following topics building foundation for next chapters to come:

- Infrastructure as a Service vs. Platform as a Service
- Objectives of Multi-Cloud Deployments factoring in Fault-Tolerance, Performance and Security
- Current Solutions and Their Limitations
- Problem Statement
- Solution Methodology and Standard Template to be used for Pattern Documentation

Chapter 2 – Survey of Relevant Research

2.1 Relevant Topics for Literature Review

There has been a great deal of research done in Cloud Computing area for the purposes of this particular study let's take a look of some relevant research covering:

- Design and Cloud Patterns
- Multi-Cloud Deployment
- Cloud Fault Tolerance
- Cloud Standards
- Cloud and Multi-Cloud Security
- Distributed Systems
- Cloud Cost Efficiency

2.2 Relevant Definitions and Examples

To start with let's review the key cloud computing concepts definitions.

Virtualization is one of the key technologies for cloud computing. Virtualization in the cloud covers generally three key areas:

- Compute – this includes virtual machines and any other component that has a CPU
- Networking – necessary for all components to communicate and usually has an API (Application Programming Interface) around it

- Storage – covers virtualized storage attached to virtual machines / compute and unattached storage exposed as API etc.

Cloud Computing is type of shared multi-tenant computing which usually provides remote pool of computing resources via the internet. At high level it includes the following resources: compute, networking and storage. Cloud computing enables rental of these resources for a duration of time without a major upfront capital investment – pay as you go model. Virtualization, Internet Technologies and World Wide Web are the key foundations of cloud computing. Key Cloud Computing service offerings can be generally categorized as: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Function as a Service (FaaS), Software as a Service (SaaS).

Multi-Cloud Computing is an extension of cloud computing when you use more than one cloud system which can include public or private cloud or both/hybrid of public and private. Some of the benefits of Multi-Cloud are better fault tolerance, resilience, vendor agnosticism and better cost management.

Infrastructure as a Service (IaaS): virtualized datacenter as a service model which allows you to provision Compute Resources (Physical or Virtual Machines), Networking, Storage, Security Services from a service provider. IaaS is usually priced at time usage increments. Examples include:

- AWS EC2 - https://aws.amazon.com/ec2/
- Google Compute Engine - https://cloud.google.com/compute/

- IBM Softlayer - https://www.ibm.com/cloud-computing/in-en/infrastructure/softlayer/
- Microsoft Azure - https://azure.microsoft.com/en-us/services/virtual-machines/

Platform as a Service (PaaS): PaaS allows developers to run application or services without the need to worry about managing IaaS layer, Operating System or Runtime environment. PaaS is usually build on top of IaaS layer with managed Operating System and Application Runtime predefined such as Java, .NET, Node.js etc. In PaaS model customer gets more features, but has less control than IaaS and more lock in. PaaS is usually priced at time based usage increments. Example:

- Google App Engine - https://appengine.google.com/
- IBM Bluemix - https://www.ibm.com/cloud-computing/bluemix/
- Heroku (runs on top of AWS EC2) - https://www.heroku.com

Function as a Service (FaaS): is a cloud offering that can be used to run cloud functions without the need to write an application and everything is written usually as a function that responds to certain events. FaaS can be thought of an abstraction on top of PaaS. Another important difference is that FaaS is priced usually per function execution. Customer usually gets more features than PaaS (i.e. including web based IDE with pre-build code templates) but even less control than PaaS with even more lock in, because the general FaaS offerings usually deal with vendor specific integrations via non-standard APIs. Examples include:

- AWS Lambda - https://aws.amazon.com/lambda/
- Azure Functions - https://azure.microsoft.com/en-us/services/functions/
- Google Cloud Functions - https://cloud.google.com/functions/

Software as a Service (SaaS): in this model, everything in the stack including application or service managed by the vendor and customer gets more turn-key features than PaaS or FaaS without the need to program, but this results in even less control over the stack or what's running where than PaaS FaaS with even more SaaS provider lock in.

Examples:

- Google Apps (G Suite – Alternative to Microsoft Office): https://gsuite.google.com
- Microsoft Office 365 (Microsoft Office in the Cloud): https://www.office.com
- Salesforce CRM (Customer Relationship Management) Platform https://www.salesforce.com
- Workday (Human Resources Solutions) - https://www.workday.com

Fault tolerant system can be defined as the system that can continue operating even if one or more component of the system fails. "If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system in which even a small failure can cause total breakdown." Adaptive Fault Tolerance and Graceful Degradation, Oscar González et al., 1997, University of Massachusetts – Amherst

A fault-tolerant design refers to an ability of “a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.” Johnson, B. W. (1984). "Fault-Tolerant Microprocessor-Based Systems", IEEE Micro, vol. 4, no. 6, pp. 6–21

Self-Healing System can be defined as system that can detect a fault and automatically adjust to the desired/healthy state without human operator intervention. [37]

DNS – Domain Name System is core of the Internet, but also extremely important to multi-cloud deployments as discovery and resolutions of routes to different cloud provider instances will not be possible without it.

Horizontal Scaling and Auto Scaling

Other technologies of interests have to do with Horizontal Scaling which known as Elastic Scaling or spinning up of compute instances as the load increases. Most cloud solutions include this as part of an offering that usually labeled as Auto-Scaling or Auto-Scaling Groups.

Availability Zones are isolated locations in data centers that generally have to guarantee fault tolerance via redundant physical racks, its own power provider, network, storage and telecommunications providers.

DevOps Automation – in order to make anything scalable beyond what humans can do we need automation. DevOps is the overloaded term that implies greater communication between developers and operations and approach to operations (hardware and software) which will be similar to continuous building, testing, and releasing software frequently

and reliably. This includes continuous build, test and This approach is also generally referred to as Infrastructure as Code which we will cover later on.

"Pets" vs "Cattle" Server Management Approach

Operators and System Admins before the cloud era were generally responsible for operating servers one at time which was akin to treating it as special "Pets". In this analogy Pet (Servers) are special and If the pet is sick we take it to veterinarian and try to figure what's wrong with it. Another word if something goes wrong with server we need to fix it.

In the cloud environment or even more so with multi-cloud environment with hundreds of servers we cannot treat servers as Pets – it's not scalable and cost efficient from human capital point of view - so we need to treat them as heard of Cattle. Cattle ("Server") is not special and pretty much the same as other cattle and if one cattle dies or gets sick - we don't even try to fix it – have the rest of the heard and we simply replace with another "cattle". Another word if we automate everything and treat infrastructure as "cattle" we don't need to figure why the server died – we create a new one that replaces the "diseased" one.

Infrastructure as Code is an approach to defining and configuring your infrastructure via software configuration instead of user interfaces, interactive installers. This includes all of compute infrastructure as well networking and storage. All of the infrastructure configuration, scripts get stored in version control system and should include unit tests following standard software practices. This is foundation of DevOps (Developer

Operations) where all of the processes are automated and human involvement is minimal of non-existent to provision infrastructure and software running on it.

There are couple of key principles/practices that Infrastructure as Code practice promotes:

- All of software configuration code and files need to be versioned in source control system i.e. git and release tracked just like any software product.
- Elimination of **Configuration Drift** – which is when one server of the same role/type is in different state which frequently occurs when infrastructure and software is hand-crafted manually by a human.
- **Reproducible infrastructure at scale** – ability to reproduce exactly same infrastructure and software using infrastructure as code scripts and configuration to create large volume of infrastructure components without relying on humans/operators.
- **Immutable Infrastructure** refers to a practice when you need to do an upgrade of any part of the stack you do not update existing running component (i.e. virtual machine), but create and a new version and replace it the old with it. The same goes for a failed component / virtual machine – we do not try to repair it – we replace it with a new version.

Open Source Software

Most of cloud offerings will not be possible without open source software - let’s just take a look at one example: hypervisor core of virtualization – most cloud providers use either Xen or KVM Linux based hypervisors.

Cloud-native is newly emerged concept of designing, deploying and running service or applications that can leverage properties and interfaces offered by cloud computing providers or frameworks. Its roots can be traced to **The Twelve-Factor App** manifesto (https://12factor.net) which promotes concepts for optimal design and deployment in the cloud environment such as declarative formats for setup automation, clean contract with operating system / cloud provider, continuous build and deployment and minimum difference between development and production environments. It also generally promotes stateless application and services where state is stored not with the primary execution environment.

Other notable mentions that helped cloud computing made possible that we don't have to define: Internet, Word Wide Web and Service Oriented Architecture, Virtualization

2.3 Existing Work around Cloud Design Patterns

Cloud Design Patterns have been researched from multiple angles, however as we will see not much research has been done in actual multi-cloud design patterns with the fault tolerance, resilience and self-healing included.

Mapping design patterns to cloud patterns to support application portability: a preliminary study

Utilization of Design Patterns to support application portability in the cloud is very important topic. In software engineering concept of design patterns is well documented. Can we apply the same approach to Cloud Patterns? Cloud Patterns can be seen as an advancement of exemplary Design Patterns, since they give ideal answers for

programming improvement specific for Cloud situations. By mapping Design and Cloud Patterns components, authors think it is conceivable to build up a way to deal with backing the porting of legacy applications to the Cloud, in this way facilitating the modernization procedure and laying the premise for an interoperability and transportability benevolent programming advancement. In this paper authors study if it's possible to map Design and Cloud Patterns with score-based methodology which matches pattern categories to a solution. The comparison is accomplished via a semantic based representation. [9]

Cloud Network Architecture Design Patterns

Cloud Network Architecture Design Patterns research focuses on low level networking patterns on how to connect to public cloud and how public cloud should be designed with network resiliency in mind. The focus of the paper is primarily on low level data center design, network architecture patterns and how to build Infrastructure as a Service. This paper does not cover patterns built on top of Infrastructure as a Service that will be described later in this work. [16]

Moving to the cloud: patterns, integration challenges and opportunities

The focus of this research is on high level approach on how to migrate to the cloud and to be able to leverage patterns to overcome common challenges. The general approach presented here is focused on hybrid cloud. Specifically, it covers patterns around application deployment and how to deal with hybrid cloud deployments. Overall this paper provides a general high level overview but does not seem to provide the depth of the patterns. [24]

An architectural pattern language of cloud-based applications

This paper gives a very good overview of key properties of cloud deployment and how to do architecture design for a cloud with patterns. The goal of this research is to simplify cloud application design and development via use patterns. More specifically authors have developed a pattern language which seems to be icon based but with cloud context. Some patterns seem to be vendor independent and some seems are vendor specific. The patterns covered are not the same as in this current study and focus on higher abstract concepts i.e. IaaS, PaaS, SaaS etc. [14]

Cloud Migration Patterns: A Multi-cloud Service Architecture Perspective

Focus of this work is on creation of a pattern catalogue of **application** cloud migration patterns which serves as basis for the current paper. The authors offer steps for organization on how to migrate existing applications to the cloud. Patterns have been derived from empirical evidence and from a number of migration projects, best practices for cloud architectures. The flow of the process is to select application migration pattern, define a migration plan and possibly extend the pattern if this is a new context. Overall this is an excellent body of work, however, it does not provide any guidance around infrastructure software creation, fault tolerance, security and any kind of operational concerns. [20]

Evolution of Design Patterns: a Replication Study

This is an interesting paper exploring evolution of design patterns. The authors explain the role of design patterns in replication study. The goal of the study is to investigate and validate design patterns via empirical study. The patterns are validated with open source software. The authors have proven the significance of design patters and confirm that "patterns change more frequently when they play a crucial role in the software and when in newer releases they support more advanced features." One of the important outcomes of the study is the proof that theoretical design patterns matter in the real world. [30]

Semantic and Matchmaking Technologies for Discovering, Mapping and Aligning Cloud Provider's Services

This paper explores solutions to the problems with non-standard cloud provider Application Programming Interfaces (APIs). As stated before most cloud provider APIs are not standard and require specialized knowledge of interfaces and client tools/frameworks. This results in major challenges to port applications from one provider to another. Authors have taken an approach to solve this via vendor neutral API layer. They have developer a prototype project available at http://www.mosaic-cloud.eu. In addition, there is a dynamic discovery and mapping functionality to map specific cloud provider API to the vendor neutral APIs. The approach in general seems sound, however during validation and setup there seem to be quite a few gaps remaining. There does not seem to be any updates to the project since 2013. However, overall this is very valuable body of work. [7]

2.4 Existing Work around Fault Tolerance within Distributed and Cloud Systems

There are quite a few papers covering fault tolerance within cloud and distributed systems, but not much has been done in terms of multi-cloud computing and patterns. Let's review some of the relevant research and lessons learned.

Independent Faults in the Cloud

The research problem in this paper is focused on how can we categorize different failure levels in the cloud environment? The authors have selected Byzantine fault tolerant (BFT) protocols as a foundation to dealing with faults. BFT protocols are generally replication based solutions with a focus on keeping faults independent. However, the drawback of the solution is the cost associated with replication which includes replication delays and decreased bandwidth consumption. Authors especially focus on concept of availability zones in the cloud which we will cover extensively in some of the patterns. [17]

Fault Tolerance – Case Study

Fault Tolerance has been studied in depth and this body of work outlines various techniques of fault-tolerance in clustered compute environment treating every component as point of failure. "Any of this vast number of components can fail at any time, resulting in erroneous output." The paper discusses various fault models: Byzantine Faults, Fail-stop Faults, Fail-shutter Faults etc. The authors categorize and drill down in every fault category and offer very useful guidance how to deal with each use case. [25]

Paxos for System Builders: an overview

Paxos is a set of algorithms that help to find consensus across multiple distributed systems that are inherently unreliable. The general problem is how to figure out agreement among distributed system's participants. This process is sometimes referred to as "leader election" where a group of system components votes to figure out which one would be the leader when failure occurs and current leader has failed. The foundation of this lies in concept referred to as state machines replication which helps to create an algorithm for fault-tolerance in distributed systems. This paper focuses on making Paxos more accessible to a wider audience as well as exploring Paxos replication protocol implementation. The value of the work is also in providing performance benchmarks safety and availability properties guaranteed by use of the specific Paxos implementation. [23]

Raft Refloated: Do We Have Consensus?

In addition to Paxos there is another fault tolerance consensus algorithm – Raft that claims to be even more accessible or reason about and easier to implement than Paxos. Paxos is generally deemed as not easy to understand and even harder to implement. Raft on the other hand seems easier to understand, implement and guarantees that there will be no degradation in performance or correctness. The goal of this paper to explain raft and also analyze its performance. The authors have built an event-driven simulation framework for testing it on different distributed system topologies. In addition, authors propose improvements to Raft. Overall authors achieve validation and proof of Raft claims. [18]

2.5 Existing Work Around Multi-Cloud Deployments and Standards

There has been a great deal of research done in standardizing cloud APIs which vendors generally ignore. However not enough has been done to show that multi-cloud deployment patterns can be achieved via vendor neutral way with open source software.

Challenges in Achieving IaaS Cloud Interoperability across Multiple Cloud Management Frameworks

The main focus of this paper is federation of IaaS cloud services. The specific approach taken is via standards. One of the key contributions is the rOCCI standard Framework implemented in Ruby programming language, which implements OCCI standards. OCCI stands for Open Cloud Computing Interface, which is a standard, developed by the Open Grid Forum (OGF).

In addition, authors discuss important concepts how to pick a standard approach or non-standard approach that has a wide community adoption and backing. This is a great illustration of how open source community might be producing non-standard solutions but well implemented and ready to use. In my opinion the non-standard approach that has a large community will always win over a standard that might not be even implement. However, long term standards have shown to be the right choice especially if large vendors back these. Other issues discussed include authentication, virtual machine life-cycle management, object store management, monitoring and billing services. The author lists all the issues with OCCI which generally boil down to vendor not implementing standards, but also vendor might have offerings that there is no standard

for. As an example, the following are in AWS but OCCI doesn't support these: VPC (Virtual Private Cloud), availability zones and many others. [26]

CAMP: A Standard for Managing Applications on a PaaS Cloud

This is another paper focused on solving cloud via standardization specifically focusing on Platform as a Service (PaaS) APIs. CAMP stands for Cloud Application Management for Platforms. CAMP has been relatively successful because it was adopted by some vendors and also by large community open source projects such as OpenStack Solum and Apache Brooklyn. However, it has not been adopted by any of pubic cloud providers. CAMP standard defines the artifacts and APIs that need to be offered by a PaaS to manage building, running, administering, monitoring and patching of applications in the cloud. The standard has been developed by OASIS standards consortium. The effort is focused on allowing developers, users, and vendors to create tools and services that work with any PaaS that implements the standard. An example of a tool could be Integrated Development Environment (IDE) from where one could deploy to any PaaS that confirms to the standard. [22]

Managing elasticity across multiple cloud providers

This paper is focused on cloud federation and cloud elasticity. Elasticity is very important cloud computing concept that allows to scale computing resources up/increase and down/decrease. The paper demonstrates an architecture that allows management of elasticity across multiple cloud providers. "Currently, most Platforms as a Service (PaaS) manage application elasticity within a single cloud provider. However, the not so infrequent issue of cloud outages has become a concern that hinders the availability of

cloud-based applications. The most promising solutions to this issue are those based on the federation of multiple clouds. In this paper, we present a Multi-Cloud-PaaS architecture." [27]

Meryn: open, SLA-driven, cloud bursting PaaS

Very interesting paper describing using Platform as a Service (PaaS) for cloud elasticity and bursting pattern mixed with SLA Driven cloud controller system called Meryn. The problem authors are trying to solve is how to provide good support for SLAs (Service Level Agreements) on top of PaaS. Example of SLA could be response time of a transaction should not be less than 1 second. The more interesting part of the research is how to dynamically adjusting resource allocations to meet particular workload demands while meeting different workloads and applications. Proposed system named Meryn is open source framework that implements SLA guarantees and auto-manages resource allocation via cloud bursting. Overall the framework is sounds and feasible. However, the authors are making also claims of maximizing profit for providers which generally sounds good, but was not able to find any source code to validate the results. [11]

Towards a Common Semantic Representation of Design and Cloud Patterns

This is an excellent research study to represent cloud design patterns using formal language ODOL based on OWL ontology. The language described allows to represent both structural and behavioral part of the patterns. The authors extend OWL methods to describe behavior of a pattern as services. The approach taken would allow for automated generation of implementation from the pattern as well as semantic

understanding of the pattern is composed out of. The language used seems flexible and extensible especially within the cloud context. [8]

Workload Patterns for Quality-Driven Dynamic Cloud Service Configuration and Auto-Scaling

This is another good study into cloud SLA (service level agreements) and how to use these to auto-scale to maintain performance guarantees resulting in service workload patterns. The study explores the service level agreement qualities provided by public cloud providers. The problem is that availability part of the SLA is generally well understood but the SLA qualities related to performance are not well publicized or guaranteed. The authors use dynamic prediction techniques rooted in collaborative filtering to find and predict patterns that match a particular required workload. While this is an excellent research it does not take in account fault tolerance or multi-cloud deployments. [33]

A QoS and profit aware cloud confederation model for IaaS service providers

Quality of Service is a big problem in cloud environment as most providers do not make guarantees or provide very limited guarantees around availability, performance guarantees etc.

This paper is focused on multi-cloud, multi-datacenter quality of service challenges and introduces an "algorithm and a model for cost calculation, which enhances the decision-making process over all the VM types (on-demand, reserved, spot) to increase resources utilization and profit." Specifically, the focus is on low cost, scalability, robustness and availability. The main emphasis is on an algorithm and a model for cost calculation to

help decision making. Neither has it covered any recommendation on how to do deployments in multi-cloud environment. The paper seems to be also targeted towards AWS spot instances. [10]

2.6 Security Related Papers Focused on Cloud and Multi-Cloud Deployments

Security is extremely important topic in cloud computing, however as the follow review will show not a great deal was done in terms of securing multi-cloud deployments.

Towards Secure Inter-Cloud Architectures

This study explores how to develop secure inter-cloud communications resulting in secure inter-cloud communication architecture pattern. The work presented herein is extremely important to secure implementation of cloud federation and multi-cloud solutions. More importantly how do we share, federate and collaborate across multiple clouds securely? Authors solve the problem via patterns were the trust is established among all components. Patterns are important in this context as they help to communicate concepts of "the functional aspects and can be complemented with security patterns to achieve a Secure Inter-Cloud Architecture". Each pattern has threat analysis to better communicate the security context. [12]

Threat analysis and misuse patterns of federated inter-cloud systems

This is another good cloud security research study focused on misuse pattern and federated, inter-cloud communications. This dissertation focuses on minimizing cloud provider lock in therefore misuse pattern described here should be minimized. All of the patterns are presented from the attacker's misuse point of view. Next each pattern is supplemented by providing application and forensic properties. Every threat is detailed and helps to understand the design and how it relates to an attack surface. The misuse patterns are specifically focused on inter-cloud systems. One of the use cases is to show

how a malicious provider can misuse and leak customer's information. The paper specifically addresses the following:

- Analysis of actors with use cases with how it applies to the federated inter-cloud architecture
- Build a threat model based on the data collected.
- Create a catalog of misuse patterns

Ultimately the value patterns catalog can be used to improve the security of federated cloud systems. Overall this is extremely valuable body of work which helps to frame misuse of cloud systems in a security context and provides high level solutions and recommendations. [13]

CPL: a core language for cloud computing

This is an interesting paper focused on distributed application/systems deployment in the cloud using CPL - Cloud Platform Language. The objectives of the language at high level are to reduce bugs with type-safety, composability and what they refer to as "service combinators" which refers to composing of varies services together. CPL language uses JSON format to describe domain-specific cloud deployments. The main advantage of one language is to reduce number of bugs vs. multiple languages that are used among different deployment frameworks and methods. The language can also be used to validate the deployment before the runtime. CPL is statically typed and appropriate for any distributed system or cloud deployment. CPL provides interfaces that are also extensible and provides capability to extend custom cloud deployment services provided by cloud providers. The body of work also includes extensible libraries

of service combinators that can be used to abstract away cloud provider specific interfaces. [14]

Towards a pattern language for self-adaptation of cloud-based architectures

This study focuses on efficiency and elasticity of the cloud systems using IBM'S MAPE-K model for self-adaptation. As a result, cloud self-adaption patterns are derived and presented. Cloud computing helps to utilize resources as a per-per-use model. There is significant interest in 'build-once, use-often' solutions that can be used to compose cloud deployments. The product of this study is a pattern language that allows to compose cloud-based software architectures. This allows to pick a pattern and re-use it “off-the shelf”. All of the patterns have empirical grounding. Next authors explore how the patterns can be used with the IBM'S MAPE-K model for self-adaptation. The resulting work is reusable patterns and policies for self-adaptive cloud architectures. The patterns also can be can be composed from other patterns to solve a specific problem. [1]

A catalog of security requirements patterns for the domain of cloud computing systems

Primary purpose of this research is to catalog security requirement patterns focused on security and privacy in the cloud environment. Most customers moving to cloud computing are concerned with security and privacy and ultimately this leads to fundamental question how to trust your data and core infrastructure to a 3rd party cloud provider? This research highlights the importance of addressing security requirements earlier in software development life-cycle which generally is an afterthought. The main contribution is catalog of security and privacy requirement patterns that can help

developers to think about these patterns. The requirements are based on authors' practical experience and the work of public organizations such as ENISA and the Cloud Security Alliance. The goal is to be able to classify and re-use requirements for a particular problem and map to solution patter to the requirements. The authors also validated requirements patterns with industrial partners of the ClouDAT project. Overall this is very useful and important research that will help to improve cloud security. [4]

2.7 Self-Healing Cloud Research

A Multi-Agent System Architecture for Self-Healing Cloud Infrastructure

The problem authors are trying to solve is related to mitigating resource faults in a cloud environment. One should assume that anything can fail in the cloud environment, but for the end user the fault should be as transparent as possible. To solve the problem authors, propose to incorporate concepts of autonomic computing in the cloud environment. Specifically, the solution is to leverage intelligent autonomous agents that interact with IaaS provider interfaces and read resource state. If the resource is in failed state, the agent executes Checkpoint/Replication strategy or runs migration script to move the resource. [3]

Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds

Many organizations are running private cloud IaaS (Infrastructure-as-a-Service). Quite a few choose to use open-source IaaS cloud management frameworks such as Open Nebula, Nimbus,

Eucalyptus and Open Stack. The problem is that many of these systems have many weaknesses in terms of fault tolerance. Specifically, most problems related to single master node and Single Point of Failure (SPOF).

The solution authors present is scalable and autonomic (i.e. self-organizing and healing) virtual machine (VM) management framework called Snooze. Self-organizing hierarchical architecture is used in order to scale. Snooze manages Virtual Machines in scalable and fault tolerant manner with fault tolerance provided at all levels of the hierarchy. Authors prove the framework with tests running on top of 144 physical machines. Furthermore, the framework has no negative impact to application performance. The framework itself has very low overhead and scales very well. [15]

Utility Cloud a Novel Approach for Diagnosis and Self-Healing Based on the Uncertainty in Anomalous Metrics

The key focus of this study is self-healing of workloads in the data center. The problem is that failure anomalies are not always easy to diagnose. Since datacenters have large amount of application, software and systems we need to be able to detect the anomaly and heal automatically otherwise human operators cannot scale that well. Authors assert that diagnosis needs to occur at different "abstraction such as hardware and software, along with multiple time-series metrics for the use in environments like cloud computing". The normal approach is to focus on metric thresholds, but authors propose to classify and analyze "metrics that are qualitative can offer new methods for anomaly and fault diagnosis for their distribution". The approach is to gather all the metrics into one time-series database and then act on the derived symptoms. The analysis is performed via multi-value decision diagram (MDD) method which is proven to work well from analytical performance point of view. The approach also includes: "Naive

Bayes Classifier (NBC) with an influence diagram (ID) are used to create time-series diagnosis technique to categorize and detect anomalies/faults during runtime to approximate the influence of each self-healing system component as to systems functioning and reliability." The results are encouraging with about 0.89% improvement in the accuracy of anomaly diagnosis. False alarms were around 0.04% rates. [2]

2.8 Additional Relevant Distributed Systems and Multi-cloud Research

Next let's cover some research that will be pertinent to our multi-cloud framework and patterns presented later on.

Borg, Omega, and Kubernetes

Google has been running software in containers for at least 10 years. This paper discusses lessons learned from running orchestration and scheduling systems such as Borg and Omega (non-open source). These systems were used as an inspiration for the new open source project container orchestration system - Kubernetes. Borg was the original job scheduler, then Omega was implemented with Paxos for storing state and later on Kubernetes was built based on lessons learned from both. Authors also describe benefits of containers which goes beyond just higher utilization, but also a concept called Application Oriented Infrastructure (AOI). One of the biggest benefits of AOI is ability to enclose the consistent environment in a container so that it does not matter if you run on local machine on distributed system the environment is the same.

Building a replicated logging system with Apache Kafka

This paper is very much relevant to distributed systems and connecting things via distributed log/queue system. KAFKA plays very important role in enabling Internet of Things via scalable messaging system – Kafka. Kafka was originally open sourced by LinkedIn and at this point widely adopted in the industry as scalable publish-subscribe messaging/event logging system based on distributed commit log. "Over the past years developing and operating Kafka, we extend its log-structured architecture as a replicated logging backbone for much wider application scopes in the distributed environment. "The authors share their engineering experience to replicate Kafka logs for various distributed data-driven systems at LinkedIn. Some of the application use cases include source-of-truth data storage and stream processing. [32]

2.9 Cloud Cost Efficiency Related Research

Cloud cost is a very important factor when selecting a cloud provider. Although, there has not been a great deal of computing research done specifically in multi-cloud cost area. There was some general cloud cost research done as it relates to running different workloads so let's take a look at some examples of existing research related to cloud cost efficiency.

Cost Minimization and Load Balancing Issues to Compose Web Services in a Multi Cloud Environment

The authors explore various options of cloud services deployment and load balancing in multi-cloud environment. One of the secondary objectives is to minimize the cost. The

proposed approach is introduced via composer agent based algorithm to load balance across multiple cloud providers. Generally it seem that load balancing is well covered in this body of work but the cost factor does not seem be covered that well. [19]

Implementation of Costing Model for High Performance Computing as a Services on the Cloud Environment

There are High Performance Computing (HPC) workloads that are moving to the cloud. The authors propose a novel costing model with an algorithm to minimize the cost of running in the cloud. The model provides an easy way to calculate the cost of execution primarily based on specific processor architecture factoring the number of cores and networking, human capital cost, software licensing etc. It helps to calculate TCO – total cost of ownership and CapEx (Capital Expenditures) vs OpEx (Operating Expenditures).

The algorithms also help to calculate the profit for High Performance Computing as a Service when running this in cloud environment.

2.10 Conclusion

There has been a great deal of interesting research covering cloud standards, various patterns such as application cloud migration patterns, networking, SLA, distributed systems, security, fault tolerance and cost efficiency. However, there has not been enough research and guidance around self-healing multi-cloud infrastructure software

creation on top of IaaS and operations patterns with focus on fault tolerance, cloud provider independence and cost efficiency among primary concerns.

Next we will cover:

Chapter 3 - Patterns and Open-Source Framework for Effective Multi-Cloud Deployment

Chapter 4 – Detailed Multi-Cloud Design Patterns and Multi-Cloud Based on Open-Source Technologies

Chapter 5 - Experimental Validation

Chapter 6 - Conclusion and Future Work

Chapter 3 - Patterns and Open-Source Framework for Effective Multi-Cloud Deployment

3.1 Assumptions and Objectives for Supporting Open-Source Multi-Cloud Deployment

3.1.1 Problem Statement

Many organizations are looking to migrate to the cloud and looking for a best way to do it securely, reliably and without vendor lock in. However, most organizations have to pick a cloud provider that uses proprietary APIs and Software. Most vendors currently do not implement any cloud API standards such as TOSCA or OASIS CAMP. Therefore, to date the standard approaches to cloud computing have not be successful. In addition, cloud providers experience outages, frequently with serious business impact – so fault tolerance in cloud environment still needs more research and clear and prescriptive guidance. [Appendix A]

Variable performance is also an issue because most cloud providers overprovision their virtualized infrastructure and it results in degradation of performance and quality of service for customers depending on overprovisioning factor set by the cloud provider. This has been well documented and various benchmark studies have been done. [42]

This study focuses on framework and patterns that deliver non-proprietary APIs and Software above IaaS layer with the functionality that will be on par with proprietary software or service offered by individual provider.

This study aims to answer the following questions:

How do we design Fault Tolerant, Performant and Secure Deployment Patterns in Multi-cloud environment without vendor lock in? What are the common patterns that can help to solve this problem?

3.1.2 Key Assumptions

The key assumptions of this study include the baseline services and minimum set of APIs that generally provided by most IaaS cloud frameworks and cloud providers.

Typical Cloud Applications need the following basic IaaS services:

- Compute
- Networking
- Storage
- Virtual Machines or Containers to run in with some sort of run time environment such as Java, Node.js, Ruby Runtime, Python Runtime etc.

The study includes cloud service providers have a minimum of APIs that will be applicable to majority of cloud providers:

- Basic Authentication and Authorization APIs

- Compute – Ability to create virtual machines on demand as a minimum set of APIs Networking – Ability to add networking to Virtual Machines elastically/dynamically with routing and internet access
- Basic DNS API services
- Basic Security Groups as software/configurable defined firewalls
- Storage – Ability to add storage elastically/dynamically
- Ability to dynamically support deployments to a different Availability Zones with a minimum of hardware rack mapping to availability zone
- Basic Telemetry and Log Retrieval APIs
- Basic Billing APIs

Additional Multi-Cloud Assumptions

In our context the term Multi-Cloud covers any IaaS infrastructure private or public – here are some examples of popular IaaS:

- Usually Used in **Private** Cloud Deployment (these can also be used to create Public IaaS):
    - Apache CloudStack – open source IaaS framework https://github.com/apache/cloudstack
    - OpenStack – open source IaaS framework https://www.openstack.org/
    - VMWare vCloud – commercial closed source IaaS suite https://www.vmware.com/products/vcloud-suite.html

- **Public** Cloud Offering:
    - Amazon Web Services EC2 – commercial closed source public IaaS framework

      https://aws.amazon.com/ec2/
    - IBM BlueMix – commercial closed source IaaS framework

      https://www.ibm.com/cloud-computing/bluemix/
    - Google Compute Engine – commercial closed source IaaS framework

      https://cloud.google.com/compute/
    - Microsoft Azure – commercial closed source IaaS framework

      https://azure.microsoft.com/en-us/

Key motivation factors to use multi-cloud deployments

- **Hybrid Deployment:** Most enterprises would have private on premises deployment and will start exploring or augmenting workloads in public clouds, so in this study we will cover ability to deploy on private or public IaaS infrastructure. This is also might be referred to as hybrid private/public Multi-Cloud deployment.

- **Cost:** The operating cost might be cheaper in public cloud especially if you look at heavily discounted temporary Virtual Machine Instances
    - AWS Spot VM Instances (https://aws.amazon.com/ec2/spot/)
    - Google's Preemtible VM instances (https://cloud.google.com/compute/docs/instances/preemptible)

- Microsoft Azure's Low Priority Instances http://blog.spotinst.com/2017/05/14/microsoft-azure-low-priority-vms/
- Everything is offered at a significant discount to a standard VM pricing.

- **Exhausted Capacity:** Capacity in private cloud might be exhausted and usually you can't just add new servers fast due to lead times of server procurements, racking, cabling etc.

- **Cloud Failures:** Any cloud private or public has outages regularly [Appendix A] Failures covers not just internal cloud faults, but also external denial of service attacks that target specific cloud provider or site. Multi-Cloud approach will help with that.

- **Disaster Recover:** one cloud provider might not be enough to manage the risk of loss of service or data.

- **Avoid Vendor Lock In:** Since majority of the APIs and services are non-standard and vary among IaaS tech stacks one would to do multi-cloud to avoid single cloud provider lock in.

- **Local Geographical Proximity Presence:** some cloud providers do not have locations that are close enough to your customers. For some location such as China it might even be required from a regulatory perspective.

3.1.3 Objectives and Scope of the study

As stated in the problem statement this study focuses on framework and patterns that deliver non-proprietary APIs and Software above IaaS layer with the functionality that will be on par with proprietary software or service offered by individual provider. This study aims to answer the following questions:

How do we deploy software in Fault Tolerant, Self-Healing, Performant, Cost Effective and Secure manner in Multi-cloud environment without vendor lock in?

The study aims to identify what are the common patterns that can help to solve these problems and provide concrete implementation examples using open source software if available to an applicable pattern. Anyone should be able decide and pick any pattern for deployment on most popular cloud IaaS simultaneously.

The scope of the study covers cloud patterns and framework for multi-cloud software stack deployment covering:

Primary Objectives

(a) Fault-tolerance, resilience and ability to self-heal multi-cloud deployments without human intervention

(b) Vendor-independence so that we can deploy the same software stack easily on any cloud provider infrastructure as a service.

Secondary Objectives

(c) Performance - ability to shift workloads to another cloud provider if there is performance degradation of quality of service on one or more of providers

(d) Security – we need to include patterns covering security as arguably we should not promote any pattern that does not include security by default.

(e) Cost efficiency – public cloud provider costs give initial edge to the customer by not requiring major capital expenditures (CAPEX) to buy real estate, servers etc. However public cloud has ongoing operating expenditures costs (OPEX). [35] Each cloud provider frequently changes the OPEX rate customer pays and these generally tend to be getting cheaper over time. [47]. Multi-cloud approach should allow to leverage the cheapest offering from any cloud provider that might be cheaper at that moment.

3.1.4 Solution Methodology - how do we approach addressing these objectives?

We have covered some of the solution strategy in chapter 1 – let's quickly review the approach.

What is a design pattern and how does it help us in this context? Design pattern is a general solution to a frequently occurring problem within a particular context. In case of this study we will focus on multi-cloud deployment challenges and general patterns to solve them. Patterns help with effective knowledge sharing so that when someone references a general pattern X other people familiar with a pattern can understand the context quickly and efficiently.

In this chapter 3 we are going to identify and categorize the main challenges to achieve multi-cloud deployments. We will introduce Multi-Cloud Deployment Framework that addresses key challenges end to end. Next we will propose quality solution patterns for each of challenges and give a brief description.

In Chapter 4 we will provide solution details for each multi-cloud pattern and all of the patterns will be unified into one cohesive multi-cloud framework. All of the patterns will confirm to one structured template and provide implementation details.

In in Chapter 5 we will validate the key patterns with experiments for multi-cloud feasibility using open-source framework implementations, fault tolerance and any additional criteria depending on applicability to the pattern context. All of the patterns will have reference guide with code and scripts to implement it. Patterns will be deployed on multiple cloud providers to prove the feasibility. Failure conditions will be generated to illustrate the desired fault tolerance and self-healing properties.

3.1.5 Solution Limitations

This study focuses on automation and provisioning software components above the IaaS with the needs for IaaS APIs.

Majority of use cases focus on running on top of Virtual Machines, but running Containers is also possible using the same patterns.

Patterns covered in this study are applicable to most cloud providers. We will utilize 3 top public cloud providers AWS, Azure and Google Cloud and well as OpenStack

framework for private cloud provider deployment to demonstrate how we achieve the objectives. The code examples illustrating pattern implementation provided have been tested with the cloud frameworks listed above.

Next let’s cover key challenges with multi-cloud deployments.

3.2 Key Challenges while Deploying Enterprise Computing in Multi-Cloud Environment

What are the major challenges with multi-cloud computing?

Let's will look at categories at challenges starting with initial multi-cloud deployment challenges.

3.1.1 Initial Multi-Cloud Deployment Challenges

- How do we deploy on multiple cloud providers if all of them have different APIs? More importantly is there a way where you can define your infrastructure requirements once and something will take care of translating this into the API calls for the specific provider?
- Cloud providers require different virtual machine formats – how do create these images in an automated way?
- Where do we source and store secrets?
- How do we route to multiple providers all at once?
- Finally, where do we store and keep track of information about everything we deployed?

Now that we have everything deployed how do we manage it with proper fault tolerance so that end users will not see and service disruption?

3.2.1 Challenges related to Multi-Cloud Management after initial deployment and dealing with failures

- Once we deploy how do we find out the health of the instances?
- If the node virtual machine fail - can we recover them automatically?
- If availability zone fails – can we recover automatically?
- If the cloud provider fails, what do we do then? Can we route the requests and workloads to another cloud provider?
- What if Domain Name System fails – how do we continue operating?
- What if our deployment infrastructure fails – how do we re-deploy?
- What about the data if database fails what do I do?
- How do we fail over gracefully when fault occurs?
- Is there more intelligent way of enforcing Service Level Agreement if we know things are about to fail – before failure occurs?
- What if I want to deploy to public cloud only if private cloud capacity gets exhausted?
- What if private cloud fails how can recover from this disaster using a public cloud?

3.2.3 Cost Efficiency Multi-Cloud Challenges

- How do we actually aggregate billing and cost from all cloud deployments?
- How do we make sure that cost is not out of control and we know how much we are spending as well as deploy workloads where it is cheaper?

3.2.4 Security Multi-Cloud Challenges

- Where do we store and retrieve secrets?

- How do we ensure that deployment does not have any known vulnerabilities and gets patched if there is a vulnerability?
- How do we continuously audit the cloud and make sure there are no unauthorized or unintended changes?

3.2.5 General Application Deployment Multi-Cloud Challenges

- How do we apply to real application patterns such as Web Application etc.? More importantly - how do we deploy real applications in Multi-Cloud environment so if something fails the user or service does not experience and interruption?
- How do we apply to more complicated stack deployment i.e. Internet of Things or Big Data processing pipelines?
- How do we run Applications or Services Packaged in Containers in Multi-Cloud Environment?

3.3 Open-Source Multi-Cloud Deployment Solution Framework

Now that we have covered multi-cloud challenges - next let's take a look at big picture - key patterns that can help us with most of these challenges that combined into one Multi-Cloud Control Plane framework.

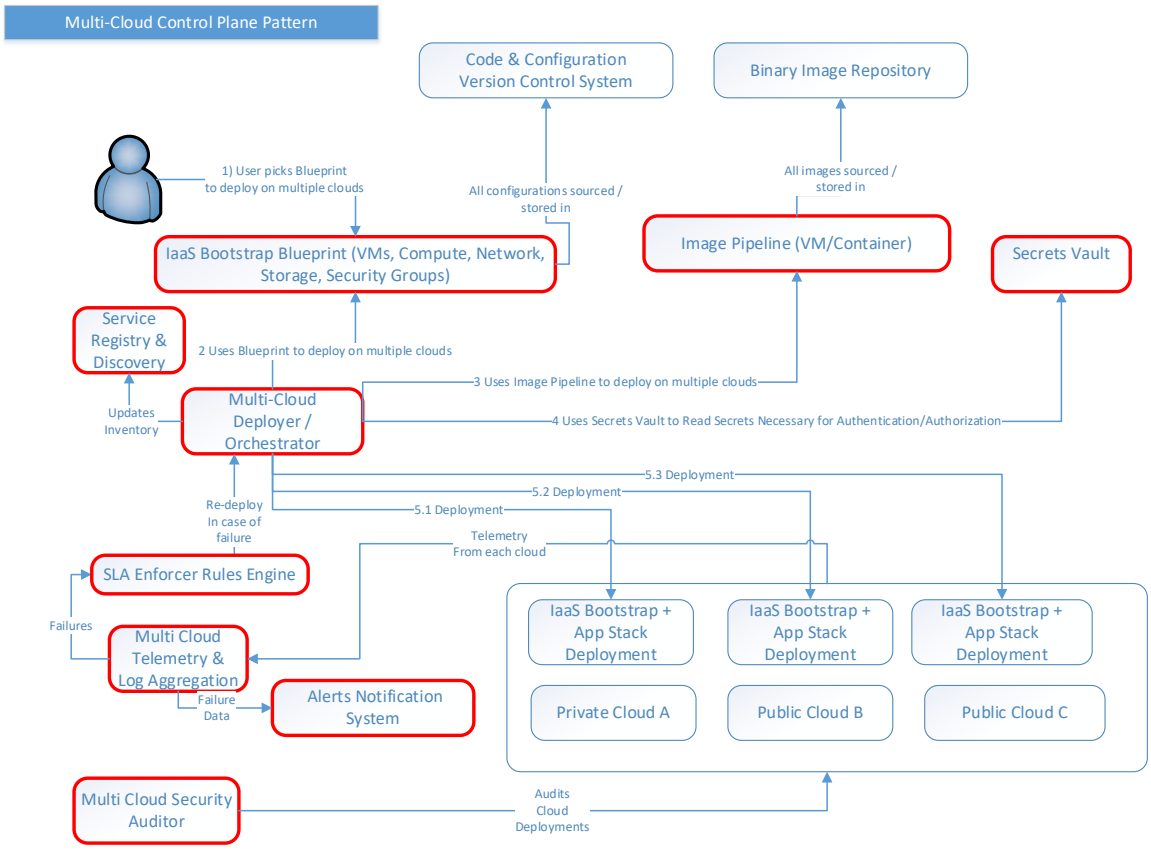


Figure 9 - Multi-Cloud Control Plane Framework

To illustrate the initial Setup of the Multi-cloud deployment let's take a look at following:

Let's say operator wants to create a deployment consisting of X Virtual machines, Network, Storage and Security Groups

1) Operator (i.e. Systems Admin) picks **IaaS Blueprint** to deploy on multiple clouds which is sourced from Code & Configuration Version Control System

IaaS Bootstrap Blueprint includes details and topology of Virtual Machines (with base Image to be used), Network, Storage and Security Groups

2) **Multi-Cloud Deployer** uses Blueprint and Base images are built via **Image Pipeline** and stored in binary repository to deploy on multiple clouds (Private Cloud A, Public Cloud B and Public Cloud C)

All security sensitive information is sourced from the **Secrets Vault**

Multi-Cloud Deployer registers running services in Service Registry

Dealing with events after initial deployment

Dealing with Failures and Performance Degradation

We need to deal with failures which is done by **SLA Enforcer Rules Engine** which in turn uses data provided by the **Multi-Cloud Telemetry** system: when a component fails, alert/event is generated and forwarded to **Telemetry System** which **SLA Enforcer Rules Engine** listens to and instructs **Multi-Cloud Deployer** to resurrect the failed component. The more complex rules about SLA and quality of service is also handled by **SLA Enforcer Rules Engine**.

Dealing with Security

Cloud Auditor – continuously runs reports against each cloud deployment comparing it against baseline blueprint. If a new component is detected that violates the policy, Cloud Auditor will send alerts and notifications, which in turn get picked up by **SLA Enforcer**

Rules Engine which will instruct the **Multi-Cloud Deployer** to delete the new component that was not in the original blueprint.

Dealing with **Cost**

Multi-Cloud Billing Pattern aggregates usage cost and billing from multiple cloud providers and allows to make better decision about where it would be more cost effective to run. The more complex rules about cost is handled by **SLA Enforcer Rules Engine**.

Next let's dive deeper into each one of deployment challenges mapped to the patterns.

3.4 Solutions to Major Multi-Cloud Computing Challenges

Now let's dive into each multi-cloud challenge and provide high level solution via an abstract pattern which will be further explored in depth in the next chapter.

3.4.1 Initial Multi-Cloud Deployment Solutions

3.4.1.1 Deployment Challenge #1 – Different Cloud APIs in Multi-Cloud Environment

Currently there is no standard cloud API or blueprint format and many cloud features are not uniformly implemented in the same way across cloud providers. How do we describe what desired deployment should be on multiple cloud providers if all of them have different non-standard blueprint formats? More importantly, is there a way where you can define your infrastructure requirements once and something will take care of translating this into a blueprint format for the specific provider?

This way you don't need to maintain IaaS definition for multiple cloud providers.

To solve this problem, we need to have a layer of abstraction independent from specific cloud provider API. Ideally you would define your cloud blueprint once and automate any cloud provider specific calls. In order to automate provisioning of multiple IaaS this pattern helps to describe your cloud infrastructure as a blueprint and use that blueprint to drive orchestrator to target specific IaaS provider and create all the resources necessary for your application and services to run on any cloud provider (Public or Private).

The solution to this problem is Multi-Cloud Cloud Blueprint Pattern will be detailed in the next chapter. In the meantime, here is the brief graphical preview of the pattern:

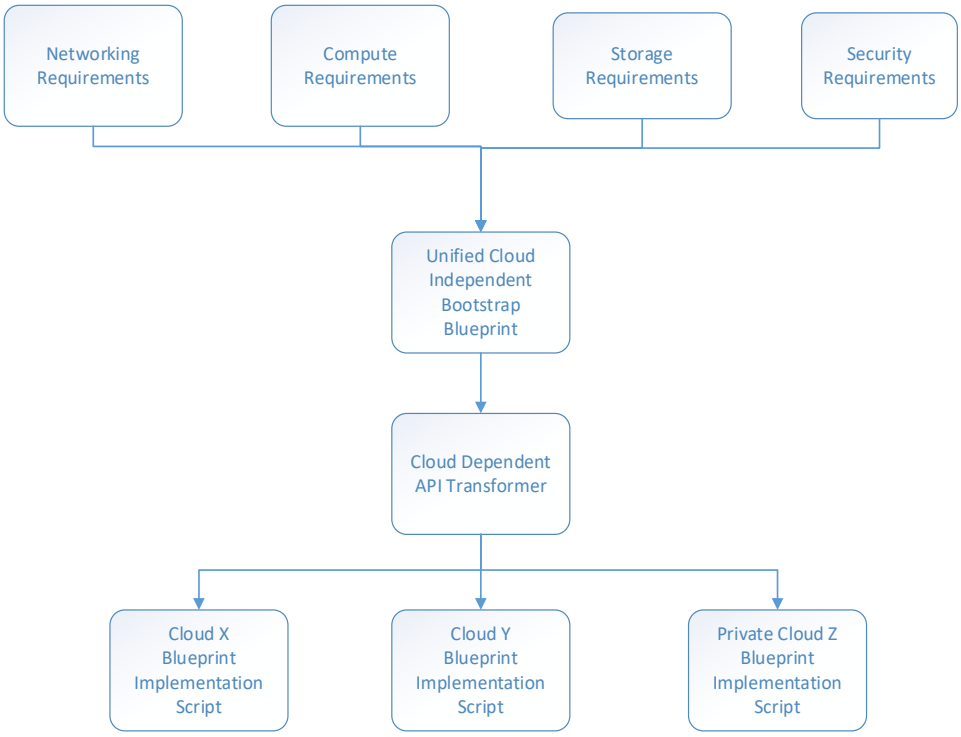


Figure 10 - Multi-Cloud Cloud Blueprint Pattern

As you can see cloud provider independent Networking, Compute, Storage and Security requirements are defined in one unified Blueprint and then translated into Script for each cloud provider specific implementation script. Operator (i.e. Systems Admin) picks IaaS Blueprint to deploy on multiple clouds which is sourced from Code & Configuration Version Control System IaaS Bootstrap Blueprint includes details and topology of Virtual Machines (with base Image to be used), Network, Storage and Security Groups

3.4.1.2 Deployment Challenge #2 – Different Image Formats in Multi-Cloud Environment

Cloud providers require different virtual machine formats – how do create these images in an automated way?

The proposed solution to this problem is Image Build Pipeline Pattern which is focused on building ready to boot images that contain entire software stack and even can contain application itself. (i.e. OS, Middleware, Application etc.) This is the concept referred to in previous section as Immutable Image Infrastructure. It is very important to have ready to boot image if you have a spike in load (via user requests, events etc.) and you need to scale out and create new instances quick. During run-time, this pattern relies on Resource Orchestrator Pattern. The build pipeline also might be tailored to produce multiple images targeting multiple providers and formats in order to gain the most efficiency and exploit any cloud provider format or API differences. There is no uniform API and it is not uniformly implemented in the same way across cloud providers. The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the Multi-Cloud Image Builder pattern:

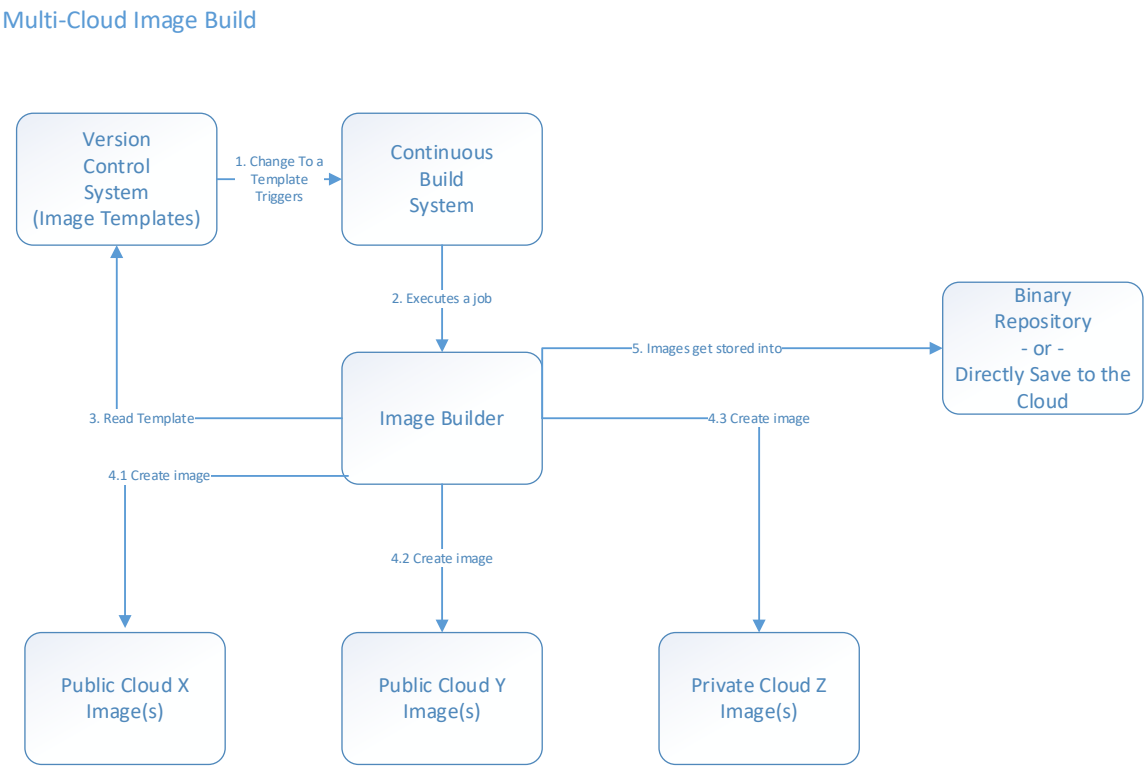


Figure 11 - Multi-Cloud Image Builder Pattern

Multi-Cloud Image Builder in combination with templates and continuous build system produces virtual machine or container images that can target public or private cloud. Images can be saved in local binary repository or directly saved to the cloud so they can be easily referenced during deployment.

3.4.1.3 Deployment Challenge #3 – Actual Multi-Cloud Deployment

Cloud provider deployment APIs are all different and non-standard. How do we deploy on top of multiple private/public cloud IaaS to make it fully available with any software stack that is able to run applications or services?

The Multi-Cloud Deployer needs to know about specific cloud provider API. Now that we covered image creation for multiple cloud providers we need a component that will do

the deployment to each cloud provider. Deployment is somewhat trivial and sometimes we need to do more than deployment – we need orchestration. Orchestration can be defined as deployment and coordination of the components to produce a desired or target state. Desired or target state refers to what we have described in the cloud blueprint i.e. x number of components describing Compute, Networking, Storage, Security. So Deployer/Orchestrator will maintain desired state based on the cloud blueprint/manifest as well as additional rules defined via a rules engine that come out of SLA Enforcer Rules Engine pattern that will be described later on.

Here is a brief graphical preview of Multi-Cloud Node (VM) Deployer - Orchestrator Pattern

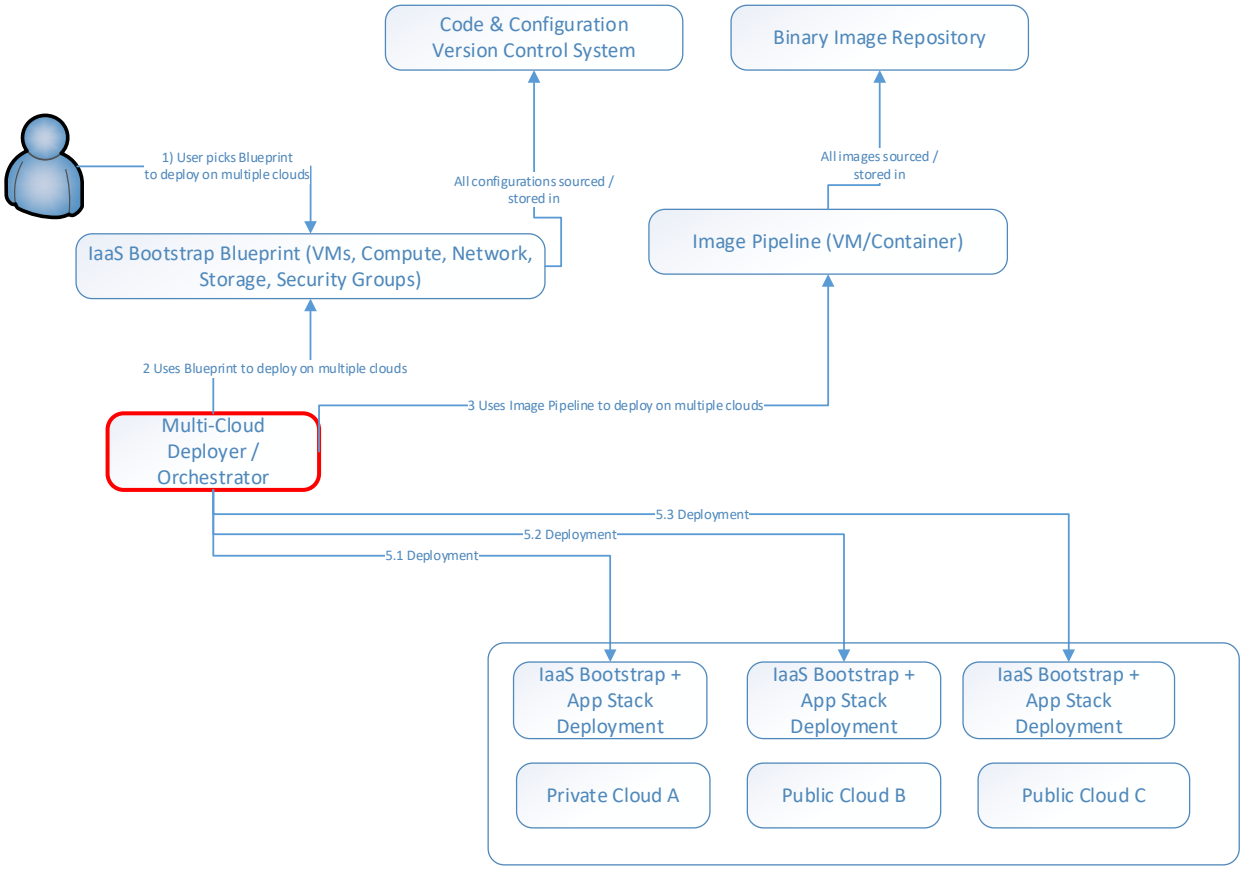


Figure 12 - Multi-Cloud Deployer Pattern

Deployer/Orchestrator uses versioned cloud blueprint/manifest combined with binary node image (produced from image pipeline) to deploy to the target clouds (Public or Private).

3.4.1.4 Deployment Challenge #4 – Fault Tolerance in Multi-Cloud Environment

How do we deploy in the most fault tolerant way in the multi-cloud environment?

Faults occur in the cloud all the time due to software, hardware or communications failure. The fundamental way to deal with failures in the cloud is via Availability Zones. Availability Zones are isolated locations in data centers that generally have to guarantee

fault tolerance via redundant physical racks, its own power provider, network, storage and even telecommunications providers. Based on historical failures in Appendix A you will see that availability zone failure is one of the most frequent failure causes.

So, the solution as we deploy to multiple clouds we need to query the availability zone metadata APIs and distributed the nodes so that not all get deployed to the same availability zones.

More details will be provided in the next chapter In the meantime, here is partial illustration of a deployment where everything is deployed into individual availability zones:

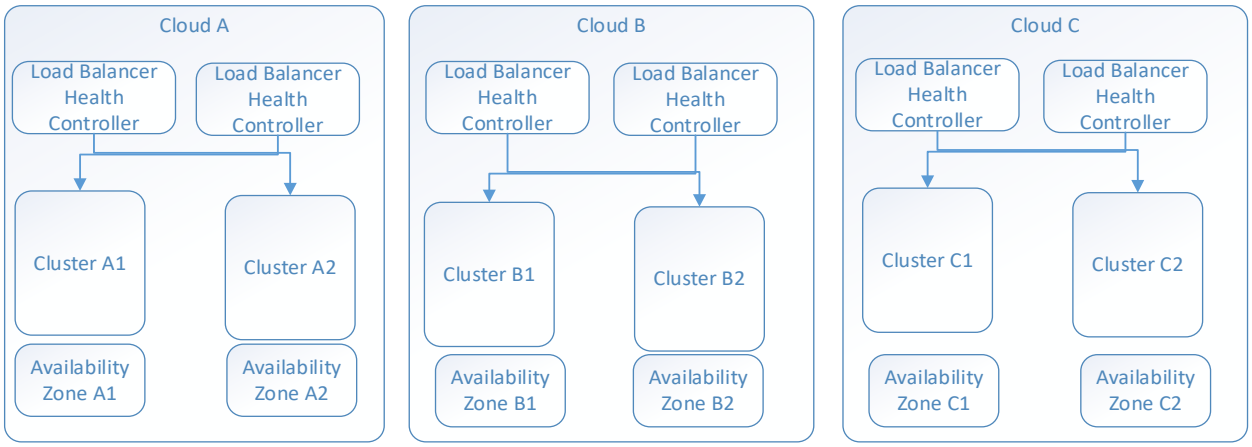


Figure 13 - Fault Tolerance in Multi-Cloud Environment

As you can see in this deployment everything is deployed in individual unique availability zone.

3.4.1.5 Deployment Challenge #5 – Routing in Multi-Cloud Environment

How do we route traffic to multiple cloud providers all at once in fault tolerant and resilient way? Domain Name System comes to mind as simple proven solution, but what happens when a DNS provider suffers outage or is under distributed denial of service attack? We can certainly have another backup DNS provider, but is there a better way? General Multi-Cloud Fault Tolerant Routing Pattern addresses these concerns.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern. In this diagram after primary DNS fails we have failover DNS to rely on, but even if both DNS providers fail we can always rely on direct IP API to continuer processing requests.

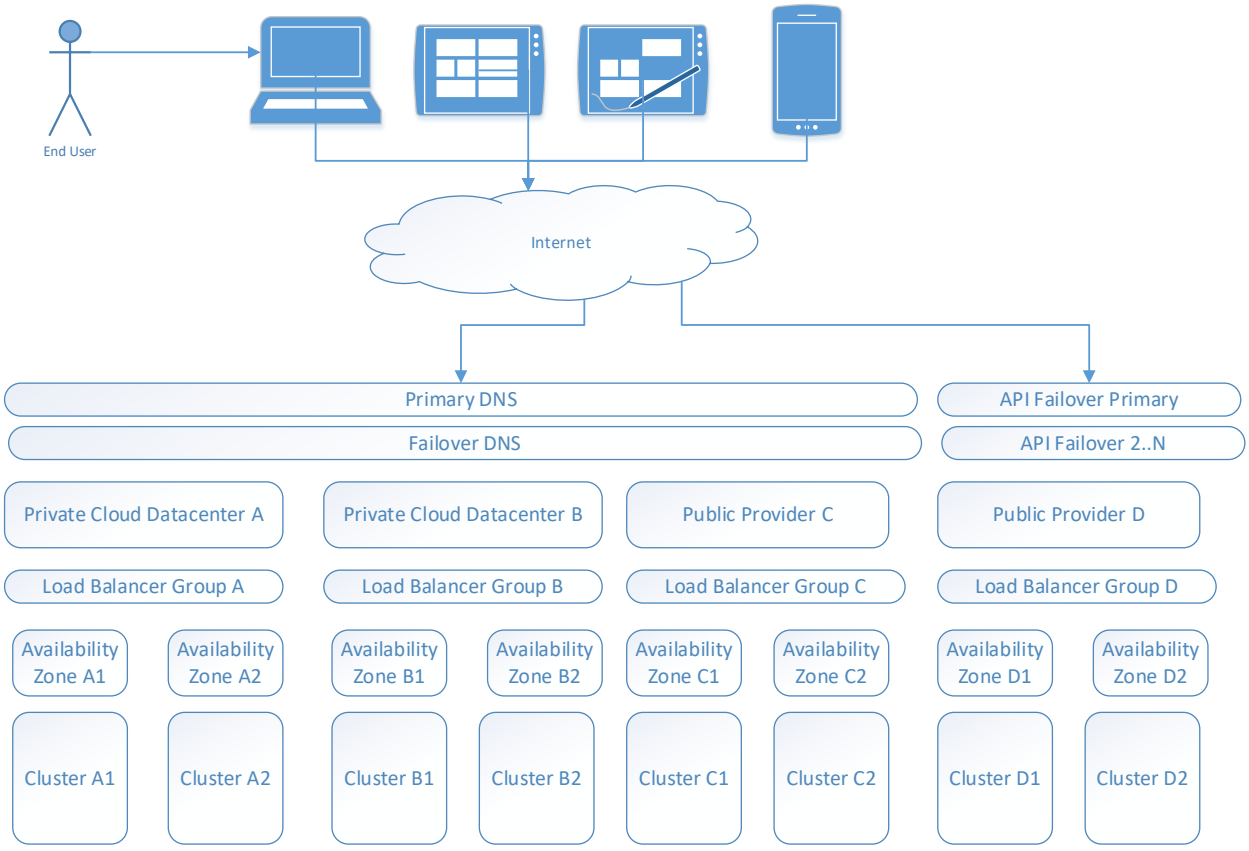


Figure 14 - Routing in Multi-Cloud Environment

Let's dive into a little more detail.

What happens if our DNS failover fails? If we are able to control TCP/IP/HTTP client on the device, we can have an API with well-known public IPs that device can cache that will be available to replace DNS. In this scenario, the device will have to periodically download the list of known IP addresses of known API end points that can be used to bypass DNS in case DNS providers fail or under DDoS attack.

3.4.1.6 Deployment Challenge #6 – Keeping Track of All Components

Finally, where do we store and keep track of information about everything we deployed in the environment with multiple cloud providers? The proposed solution to this problem is Multi-Cloud Service Registry and Discovery pattern which allows us to keep track of everything that we deploy. Think of this as distributed registry and latest source of state that has an API so that any components in multi-cloud control plan can use create/read/update/delete operation. When a new component comes alive it can always use lookup via DNS the Registry and the query for any component or meta-data service that it needs to connect to.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

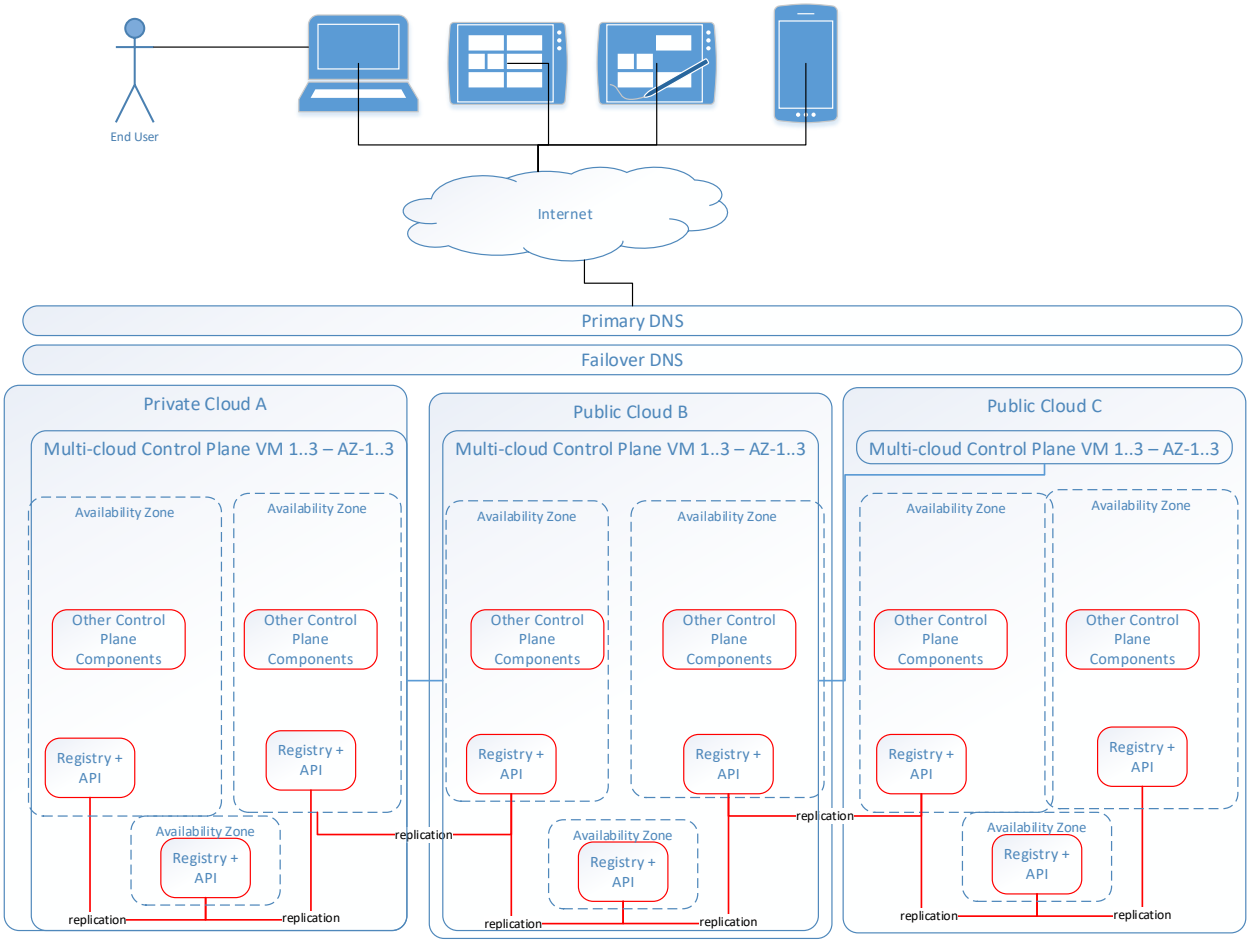


Figure 15 - Multi-Cloud Registry and API Pattern

Multi-Cloud Registry and API needs to be highly distributed, replicated and resilient as it contains latest state of deployment and where to locate services.

3.4.2 Multi-Cloud Management after Initial Deployment and Dealing with Failures

3.4.2.1 Failure Challenge #1 – Telemetry, Logs and Failure Detection in Multi-Cloud Environment

Once we deploy how do we find out the health of the node instances? It is difficult to manage anything that you cannot measure especially in large multi-cloud deployment hence we need Multi-Cloud Telemetry and Log Aggregation Pattern. This pattern helps to aggregate all of the telemetry (CPU, Memory, Storage, IO, Network utilization) from all cloud providers and all of components such Virtual Machines etc. It also aggregates key logs and health information of all of the components (for example if component, system or application is down).

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

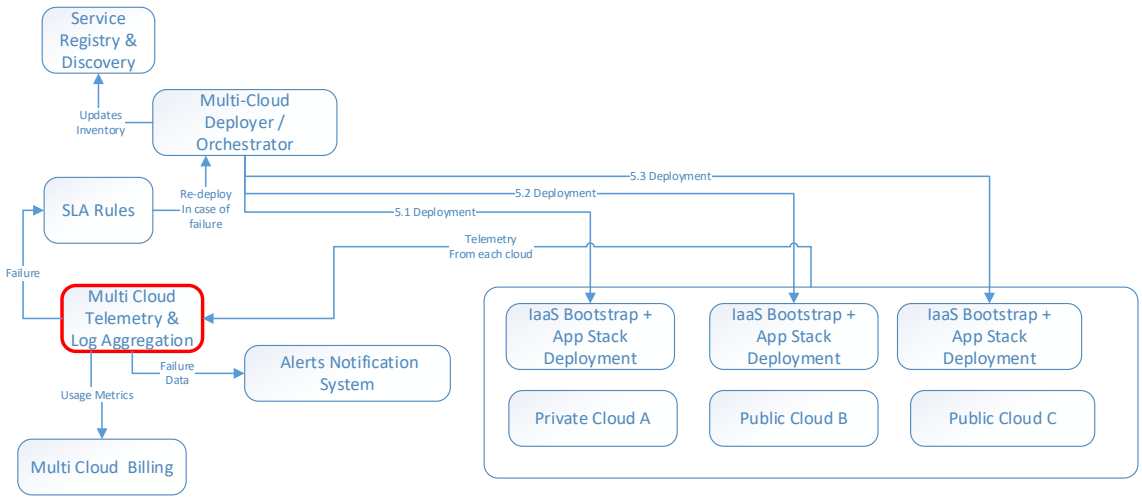


Figure 16 - Telemetry, Logs and Failure Detection in Multi-Cloud Environment

3.4.2.2 Failure Challenge #2 – Multi-Cloud Automated Recovery / Self - Healing

If the node or virtual machine fails or entire availability zone fails - can we recover automatically? What if our entire cloud deployment infrastructure fails – how do we re-deploy? So now that we know things failed or something is at high CPU/Memory/IO utilization how do we heal the failure or breach of SLA so operator does not have to even wake up i.e. at 3 a.m.?

Multi-Cloud SLA Enforcer Rules Engine pattern can automatically respond to failures based on Telemetry and Log Aggregation and trigger rules the bring the cloud deployment to the desired healthy state. Multi-cloud Telemetry and Log Aggregation feeds information to SLA Enforcer Rules Engine to process, apply rules and decide on action to be taken. In case of node failure (health check failed) SLA Enforcer Rules Engine will trigger resurrection of another node using the desired virtual machine image to match the lost node.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

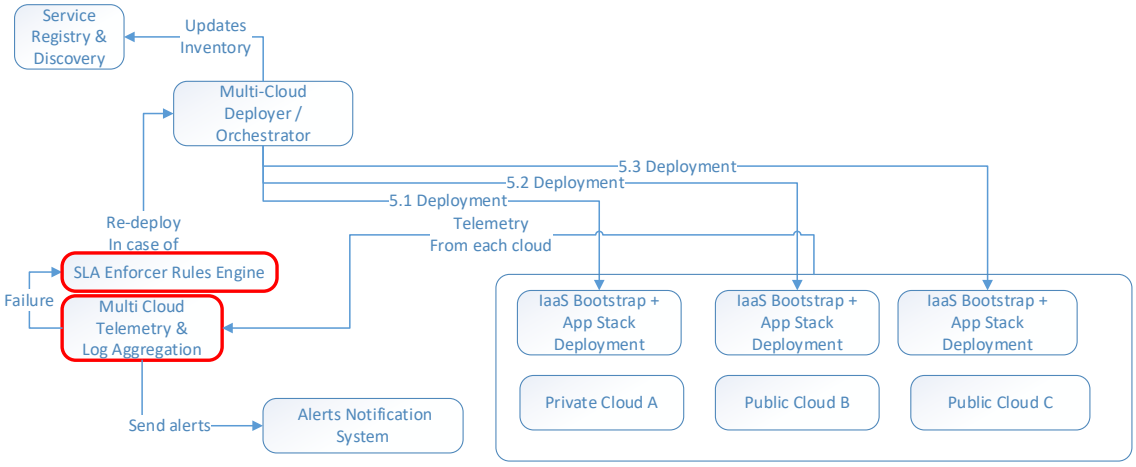


Figure 17 - Multi-Cloud SLA Enforcer Rules Engine

3.4.2.3 Failure Challenge #3 – Data Resiliency in Multi-Cloud Environment

What happens to the data if multiple databases fail? It is necessary to replicate data and state from one cloud provider to another so that in case of an outage we can continue processing on a different availability zone or cloud provider. One of the key problems in distributed systems is how to manage state and data? Specifically, one of the fundamental problems is how to safeguard the data in case database fails and how can we continue operating/processing without any downtime?

Multi-Cloud Data Replication is not as simple as it sounds because if we were to replicate across cloud providers there is significant network latency as well as possible network congestion. Hence the solution needs to include data storage solution that can tolerate high latency connections, network congestion as well as fault tolerance in each of the cloud deployments.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

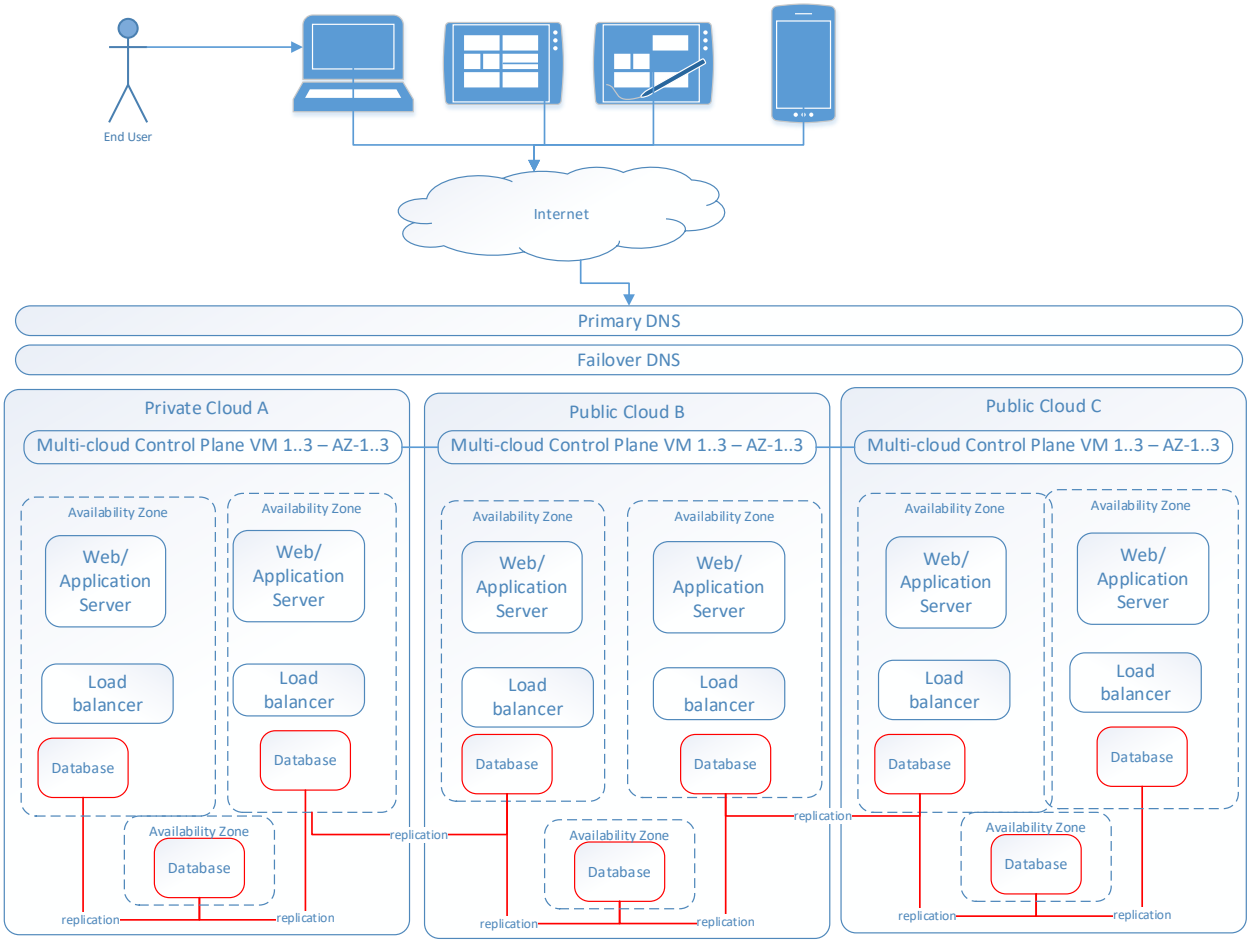


Figure 18 - Multi-Cloud Data Replication

3.4.2.4 Failure Challenge #4 – Fail Over in Case of a Fault

How do we known when fault occurs and fail over gracefully to another set of node so there is not impact to end users? There are multiple faults that can occur at multiple levels of the stack for example Virtual Machine, Storage, Network and Application Services layer. However ultimately you have a number of services that you need to health check all at once for example: Web Server, App Server and Database which impact end user experience.

The Reactive Multi-Cloud Health Check and Load Balancing Pattern consists of a dynamic web page or API HTTP URL end point on a web server that attempt to connect to app server health check, which in turn has a page to connect to the database. If any of component page's in the chain return a code that is NOT HTTP 200 then the component is down and the whole path should be marked as down.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

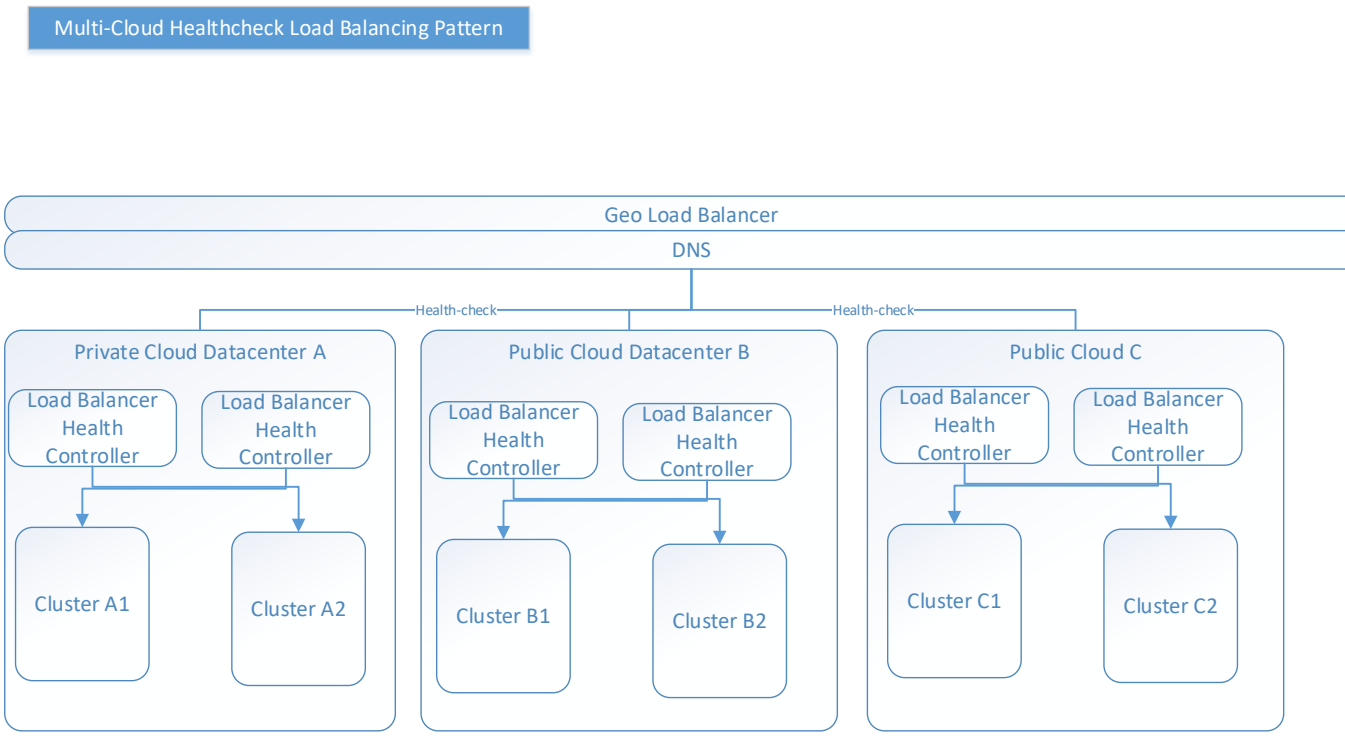


Figure 19 - Reactive Multi-Cloud Health Check and Load Balancing Pattern

3.4.2.5 Failure Challenge #5 – Advanced Failover based on SLA Telemetry

Is there more intelligent way of enforcing Service Level Agreement if we know things are about to fail – before failure occurs? Proactive Multi-Cloud SLA Policy Enforcement

Pattern can help in this case. It builds on top of Multi-Cloud SLA Telemetry and based on set of rules if a cloud provider breaches an SLA we can move workloads to another cloud provider. This pattern allows us to apply rules/triggers against aggregated multi-cloud telemetry data so we can be proactive about managing resources in case of degradation of performance etc. An example of a SLA rule and enforcement could be:

```
If Cloud X Bandwidth > Desired State

        stop routing to this cloud provider

        direct load to another cloud providers
```

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

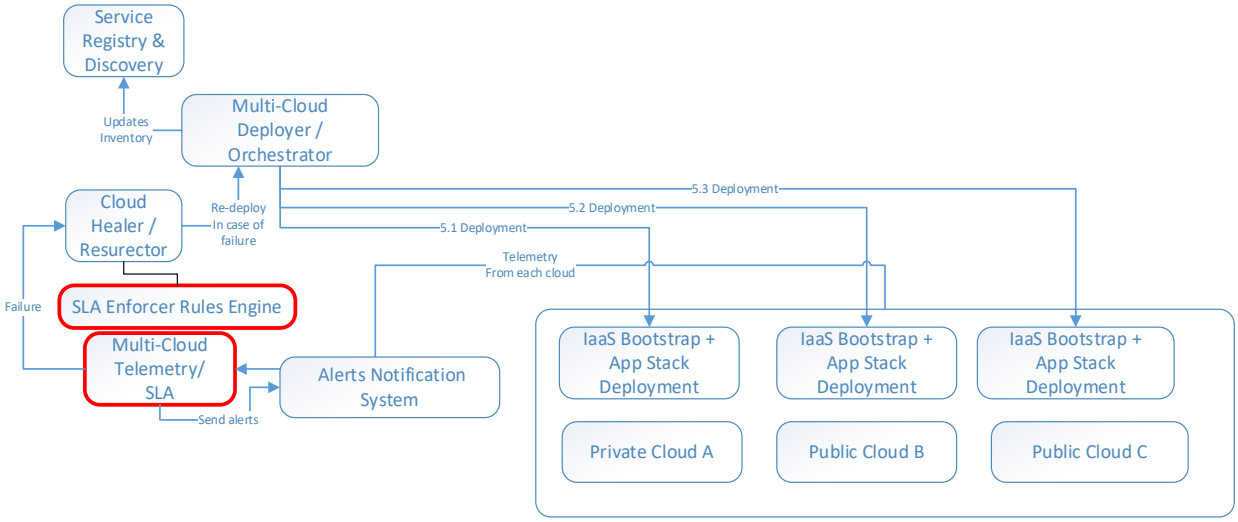


Figure 20 - Advanced Failover Based on SLA Telemetry

3.4.2.6 Failure Challenge #6 – Exhausted Capacity

What if private cloud capacity gets exhausted and we want to deploy to public cloud to increase capacity and migrate workloads?

Once you have close to or have exhausted your private cloud capacity and Public Cloud Bursting Pattern allows to "burst" and use public cloud capacity to augment on premises workloads. The entire software infrastructure stack, application or services and other components needs to be ready to be replicated in public cloud and the need. Multi-cloud control plane pattern enables cloud bursting and can be combined with Multi-Cloud SLA Enforcer to automate rules on when to burst to the cloud. Using SLA Enforcer in this pattern would allow to set a rule to state for example that at 80% utilization of private datacenter - start deploying workloads only to public cloud.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

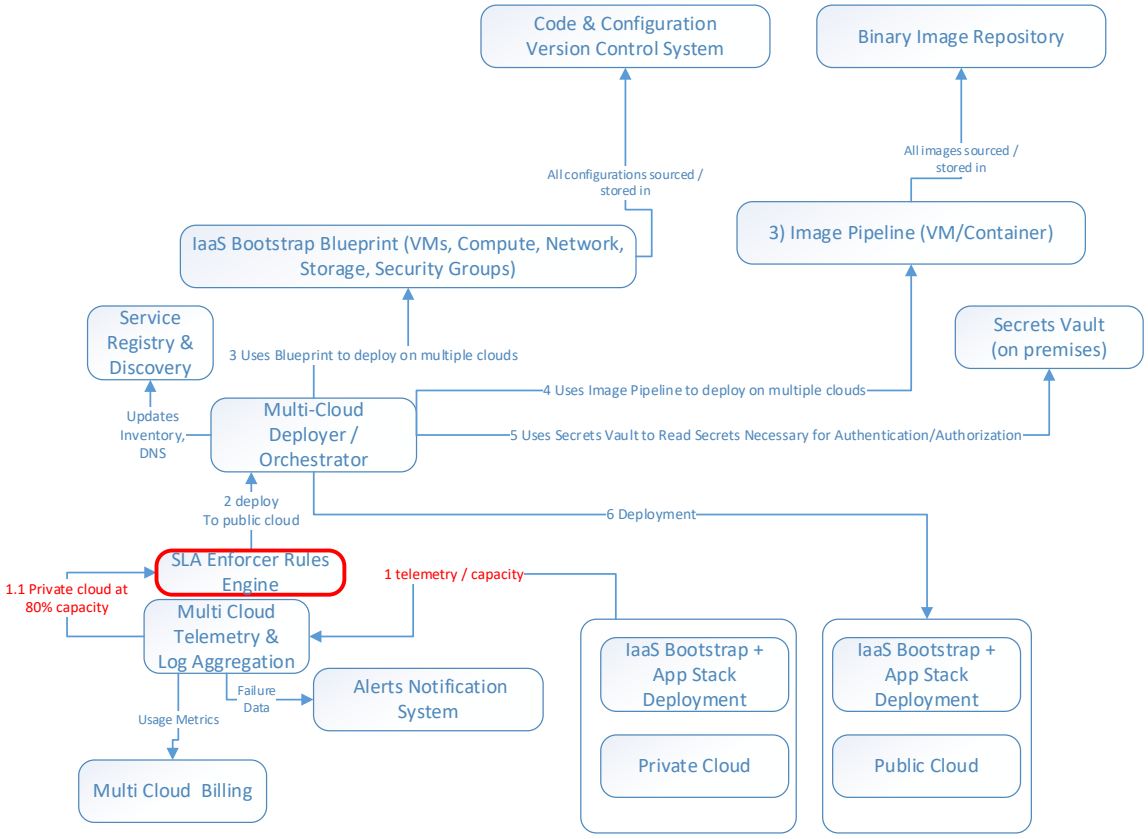


Figure 21 - Exhausted Capacity Failover

3.4.2.7 Failure Challenge #7 – Disaster Recovery

What if private cloud fails how can recover from this disaster using a public cloud?

Many organizations still run in private data centers for various reasons which might include better security and control from physical location point of view. However, what if private data/center/cloud fails? Can we recover from this disaster using a public cloud?

Variation of a Data Resiliency and Public Cloud Bursting Pattern can help in this case. During an outage in on premises data center cloud can be used to augment the private data center to continue operations. Ideally this would be done in automated fashion with advanced load balancing marking the path to the failed data center as down and automatically routing to one or more cloud provider. Continued replication of data from on premises to the cloud is needed to avoid disruption of service. This can be accomplished via Public Cloud for Disaster Recovery pattern. The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

First we would need Continuous Data Replication

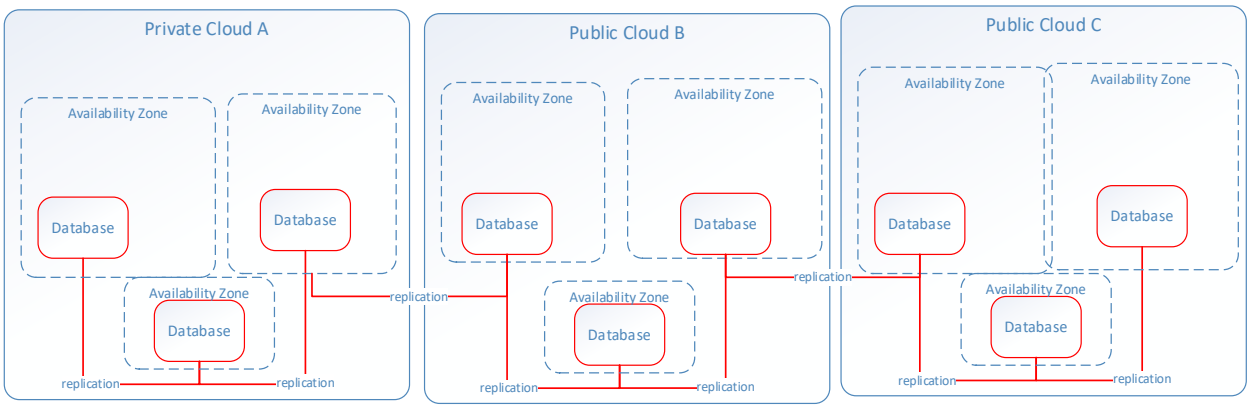


Figure 22 - Continuous Data Replication Needed for Multi-Cloud Disaster Recovery

In case of disaster Telemetry notifies SLA Enforcer that it did not receive a response from the private cloud which in turn processing the rule/action for this event and notifies SLA Enforcer Rules Engine to start deployment to public cloud.

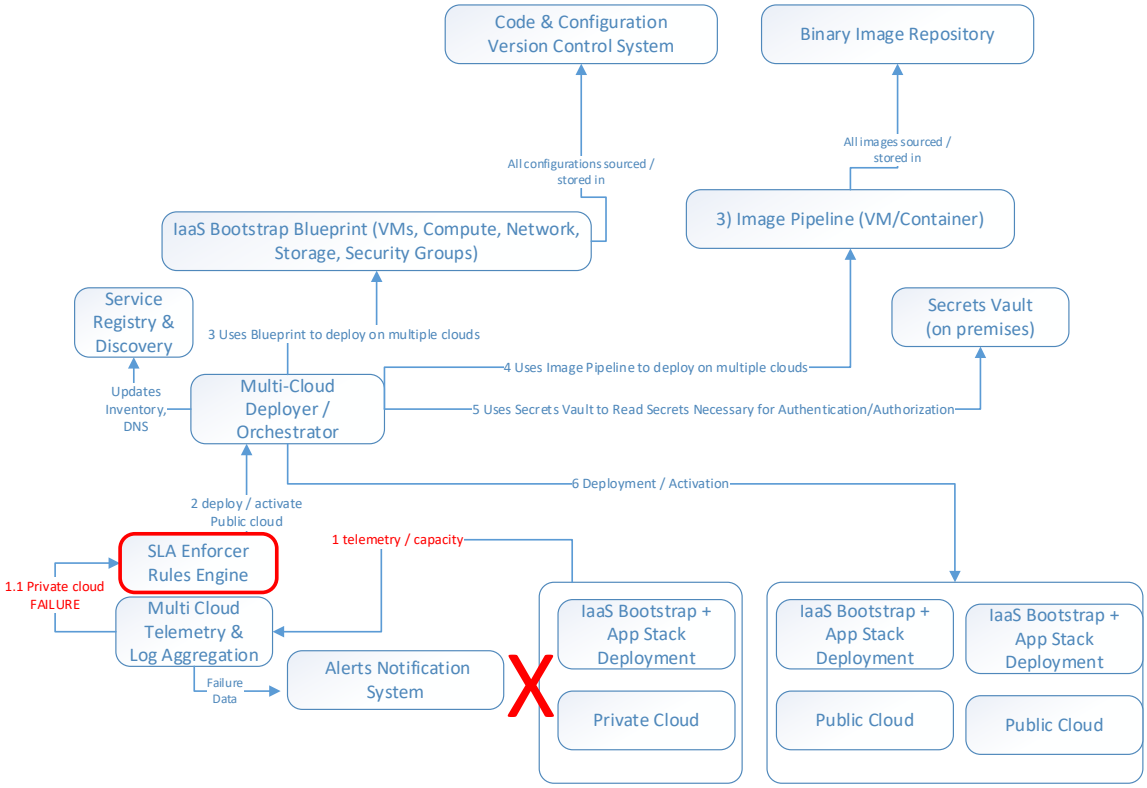


Figure 23 - Multi-Cloud Disaster Recovery Pattern

3.4.3 Cost Efficiency Multi-Cloud Challenges

Cost is a big factor in moving to the cloud and the big attraction is that you don't have to spend on capital expenditures such as upfront real estate, hardware etc. However, cloud costs can increase fast as operating expenditures, hence it is very important to figure out how to gain better efficiency via use of multiple cloud providers.

3.4.3.1 Cost Challenge #1 – Aggregate Billing in Multi-Cloud Environment

How do we aggregate billing and cost from all multi-cloud deployments?

Multi-cloud Aggregate Billing and Chargeback Pattern can help with this challenge. In a multi-cloud deployment scenario one needs to keep track and aggregate billing from multiple cloud providers. The key motivations for the pattern are:

- Aggregate billing helps with chargeback to appropriate cost centers for each application team or business unit running in multi-cloud environment.
- One might want to make sure that none of the cloud providers exceed billing quotas.
- You might want to generate alerts based on cloud utilization.

It also might make sense to process more using a cheaper cloud provider based on billing metrics. Multi-Cloud Aggregate Billing gets pricing/billing APIs data from each cloud instance (if available). In addition, Multi-Cloud Telemetry sends usage aggregate data to supplement the billing data if we can't get it reliably from the cloud provider. Once aggregate cost is calculated – alerts can get send if we breached cost threshold via Alert Notification System.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

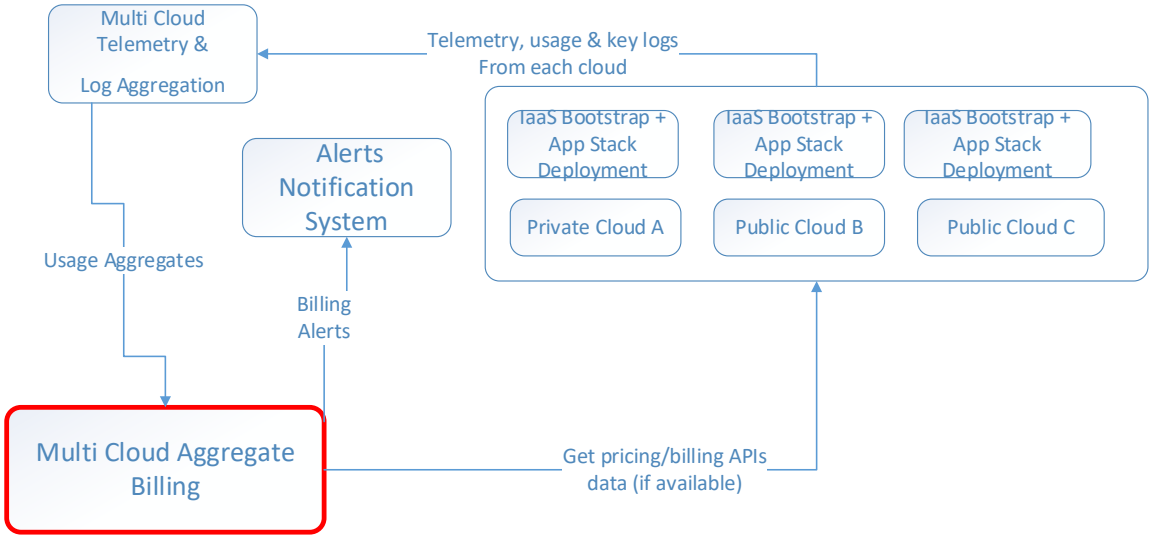


Figure 24 - Aggregate Billing in Multi-Cloud Environment

3.4.3.2 Cost Challenge #2 – Taking Advantage of Cloud Provider Price Discounts Dynamically

How do we make sure that cost is not out of control and we know how much we are spending as well as deploy workloads where it is cheaper?

In some cases, cloud providers are offering discount virtual machine instances due to idling extra capacity allowing customers to save money. Usually cloud provider allows to bid for spare capacity, however the instances can be taken away by an event notice from cloud provider so the applicable use cases/workloads need to be able to scale on demand and shut down as the cloud provider requests. Therefore, one needs to be able provision instances in automated/dynamic nature and be able to tear down them down when the cloud provider needs it back gracefully so that we don't have a negative impact to the cloud application workloads. So how do we dynamically take advantage of discount prices without impacting the transactional SLA?

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

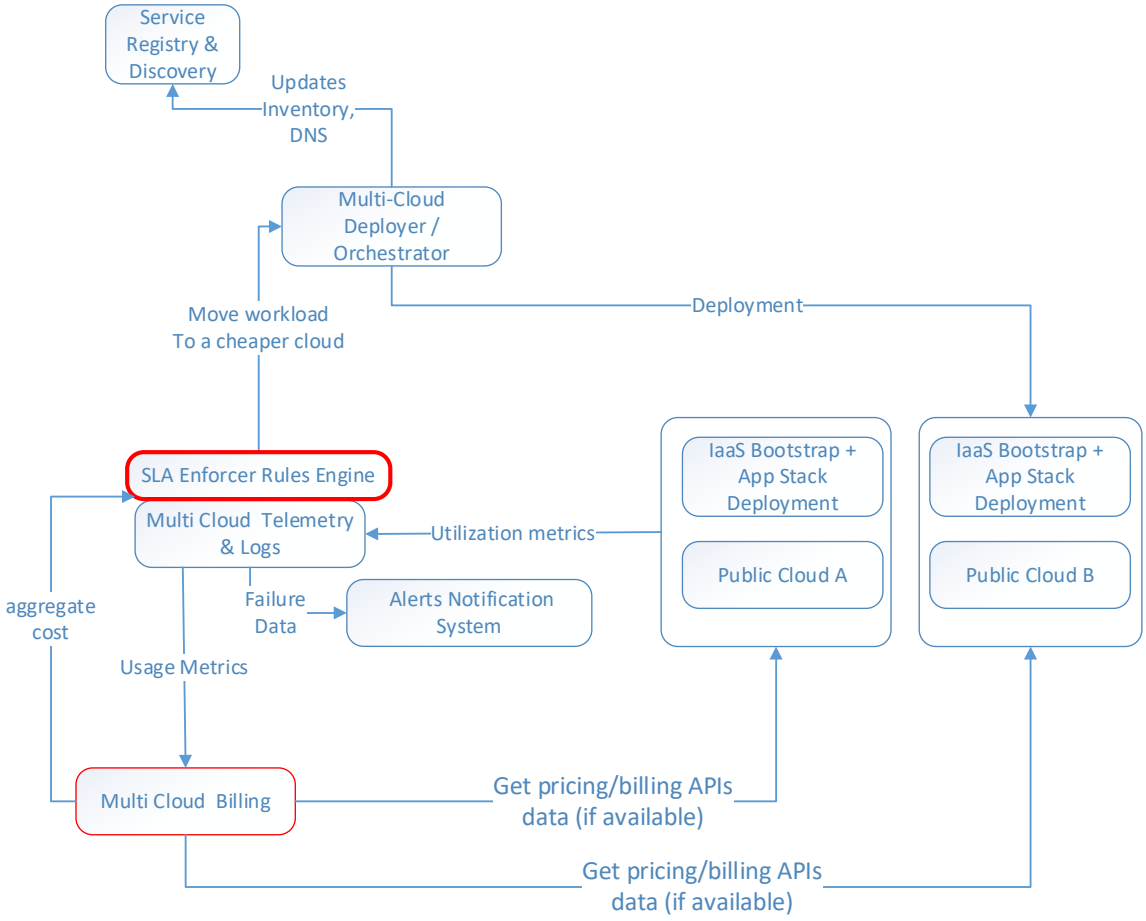


Figure 25 - Taking Advantage of Cloud Provider Price Discounts Dynamically

- Multi Cloud Billing gets lower change in price from cloud provider
- Multi Cloud Billing notifies SLA Enforcer Rules Engine
- SLA Enforcer Rules Engine evaluates the data and find the rule match to Move workload to a cheaper cloud

- SLA Enforcer Rules Engine triggers action for SLA Enforcer Rules Engine Move workload to a cheaper cloud
- SLA Enforcer Rules Engine starts deployment move workloads to a cheaper cloud

3.4.4 Security Challenges in Multi-Cloud Environment

3.4.4.1. Security Challenge #1 – Multi-Cloud Secrets Storage and Retrieval

Where do we securely store and retrieve secrets when we are dealing with multiple cloud providers?

Secrets include authentication information such as user names and passwords, certificates, private keys etc. Every cloud deployment needs secure secrets storage involves user names, passwords, certificates and any other sensitive information that needs to be protected. At some point create/update/delete and even read operations for cloud APIs will require authentication and authorization. How do we this securely in multi-cloud environment? More importantly we need to make sure this pattern works for all cloud providers. Multi-Cloud Secret Storage and Retrieval – Secrets Vault Pattern helps us to solve this problem. It permits secure access via API to secrets and can be deployed on premises as well as with every cloud deployment. API is necessary for automation and interaction in the orchestration flow of multi-cloud control plane framework.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

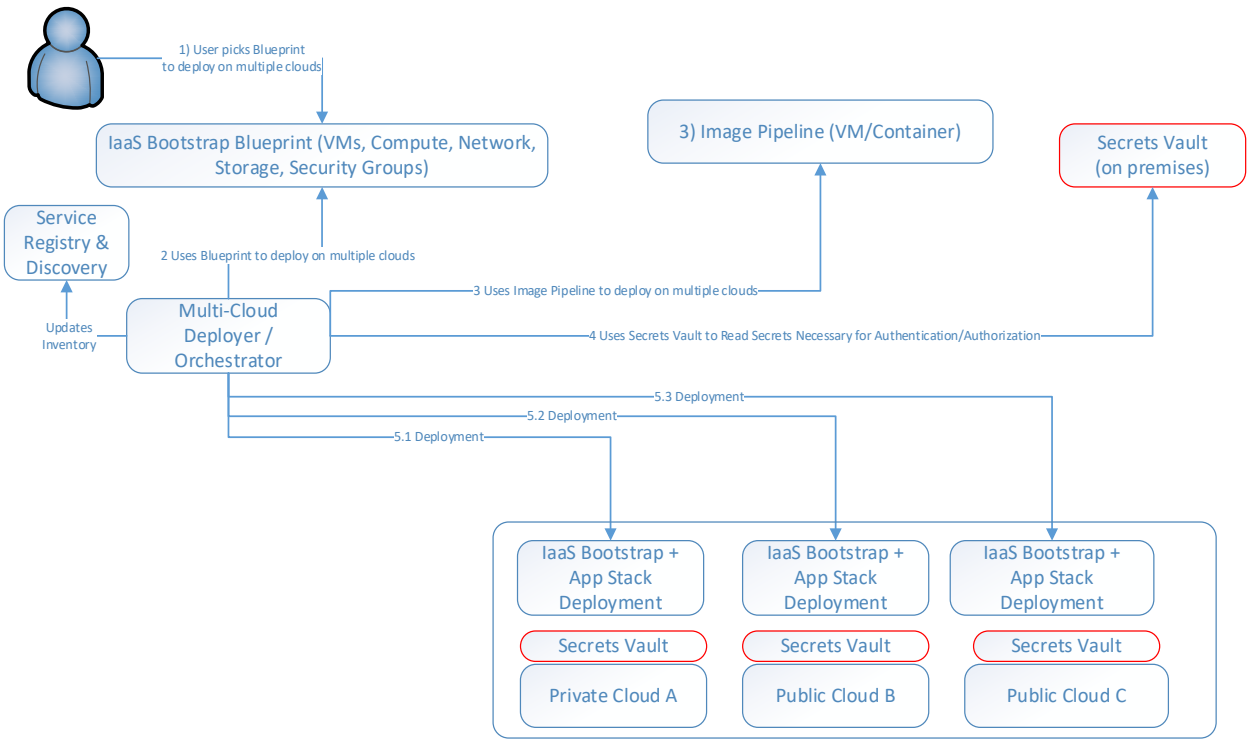


Figure 26 - Multi-Cloud Secrets Storage and Retrieval

3.4.4.2 Security Challenge #2 – Multi-Cloud Security Policy Audit, Compliance and Vulnerability Detection

How do we ensure that deployment does not have any known vulnerabilities and gets patched if there is a vulnerability? How do we continuously audit the cloud and make sure there are no unauthorized or unintended changes?

Multi-Cloud Security Policy Auditor Pattern helps us to assure that our multi-cloud deployment has not be tempered with by malicious actors and patched frequently if there is a vulnerability.

Motivation for this pattern is to be able to patch instances from known CVEs (vulnerabilities) and protect from tempering by malicious users. Multi-Cloud Security

Policy Auditor Pattern helps us to assure that our multi-cloud deployment has not be tempered with by malicious actors and patched frequently if there is a vulnerability.

Cloud Auditor continuously runs and checks all of the setting and configurations on all clouds and validates against last known secure configuration manifests as well as checking for possible vulnerabilities that might have occurred due to a new deployment / application introduced into the cloud. After that findings are logged and alerts are sent out if serious issue have been found.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

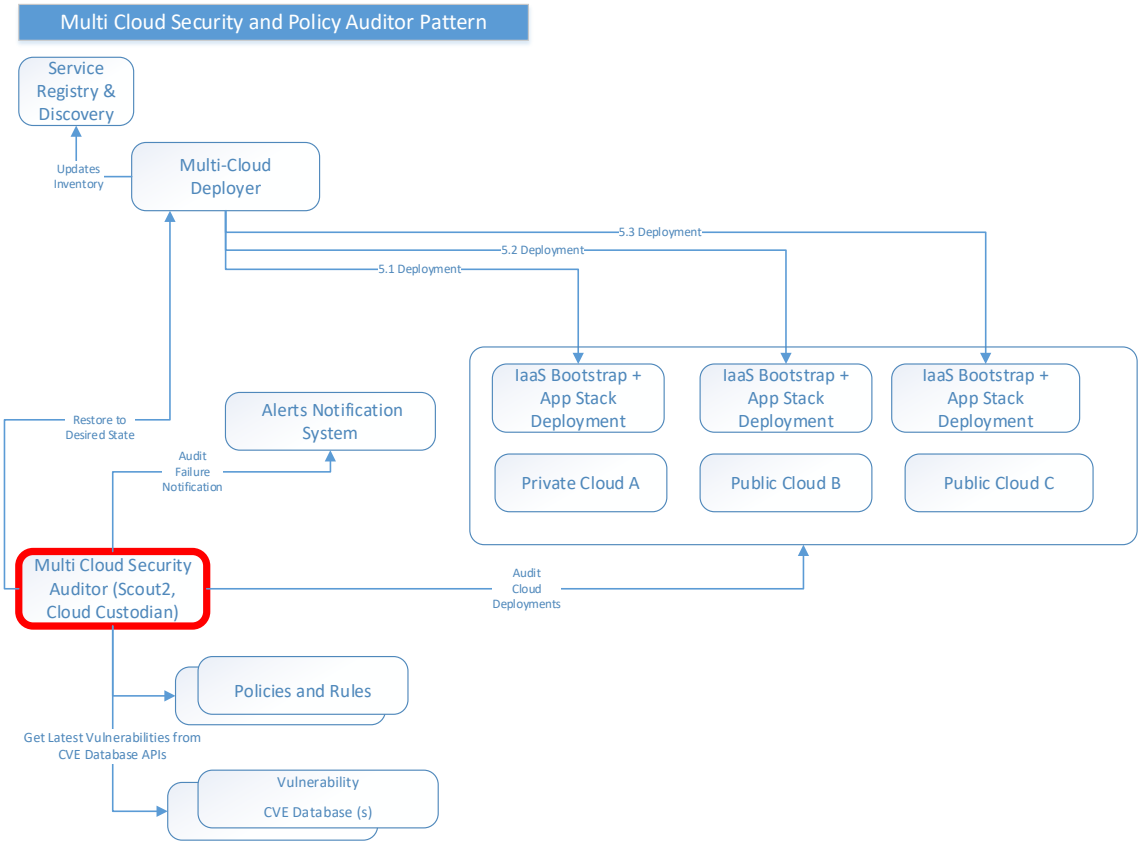


Figure 27 - Multi-Cloud Security Policy Auditor Pattern

3.4.5 General Multi-Cloud Application Deployment Challenges

3.4.5.1. Application Deployment Challenge #1 – Web Application / Service Deployment

Now that we have our Multi-Cloud IaaS and application runtime ready to run applications or services how do we apply this to real application patterns such as Web Application deployment? More importantly - how do we deploy real applications in Multi-Cloud environment so if something fails the user or service does not experience and interruption?

Multi-Cloud Web Application / Service Pattern helps to deploy web application or web services in multi-cloud environment and keep healing the failed components. This pattern will deploy general components you expect in most web applications in fault tolerant manner. In addition, all of the components will be monitored and healed by Multi-cloud control plane without for any involvement of a human operator.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

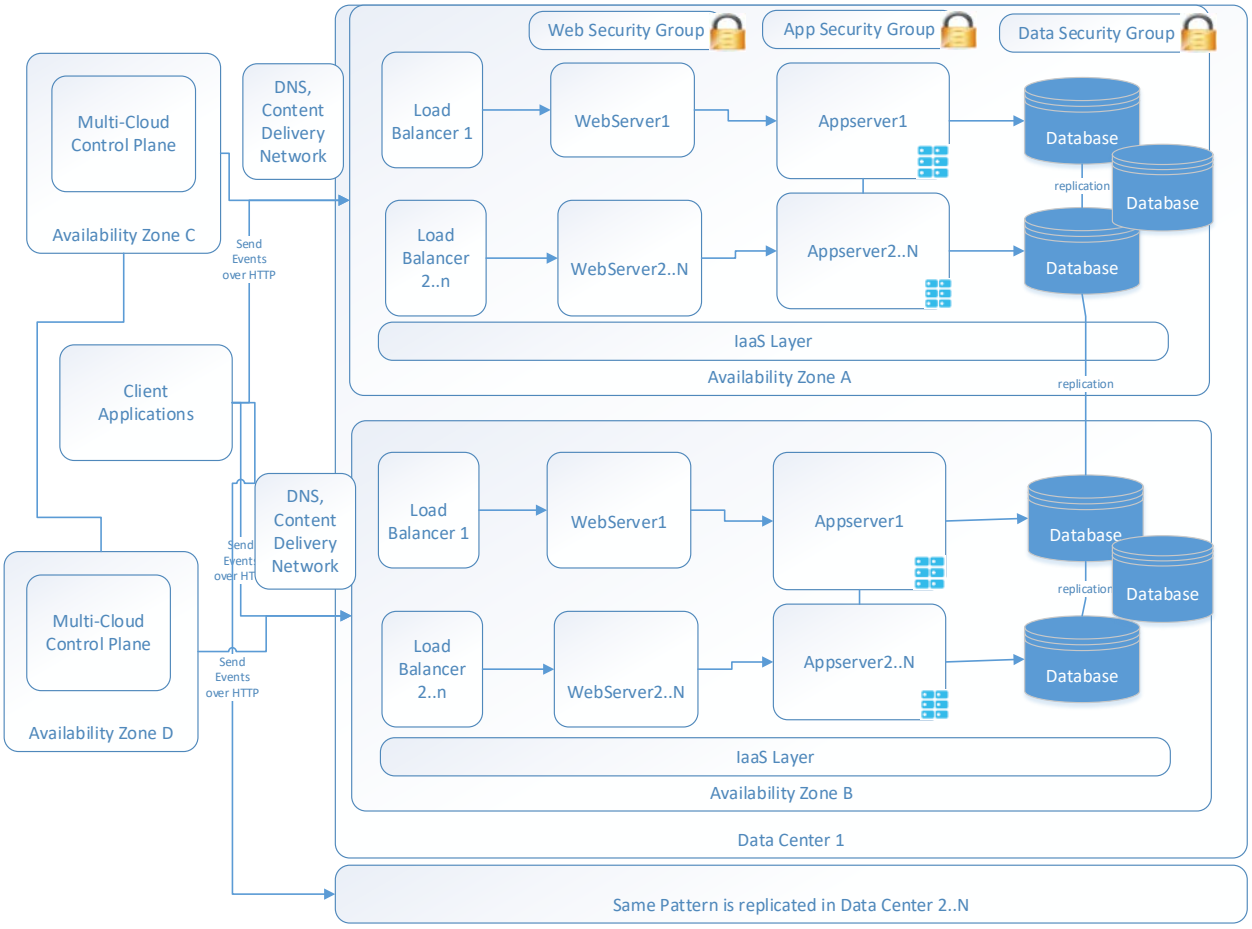


Figure 28 - Multi-Cloud Web Application / Service Pattern

3.4.5.2 Application Deployment Challenge #2 – Multi-Cloud Internet of Things and Big Data Deployment

How do we apply what we have covered to a more complicated stack deployment i.e. Internet of Things or Big Data processing pipelines?

Internet of Things (IoT) can be generally defined as any device or object around us that are connected via Internet network this can be sensors, vehicles, buildings etc. All of these devices are reporting extremely high volume of data events or streams of

information. This pattern can be characterized more specifically by high data volume, high rate of concurrent connections from 100 of millions – billions of devices. We will cover the detailed fault tolerant solution to this problem in Chapter 4.

The detailed solution to this problem will be provided in the chapter 4 with validation in chapter 5. In the meantime, here is the brief graphical preview of the pattern:

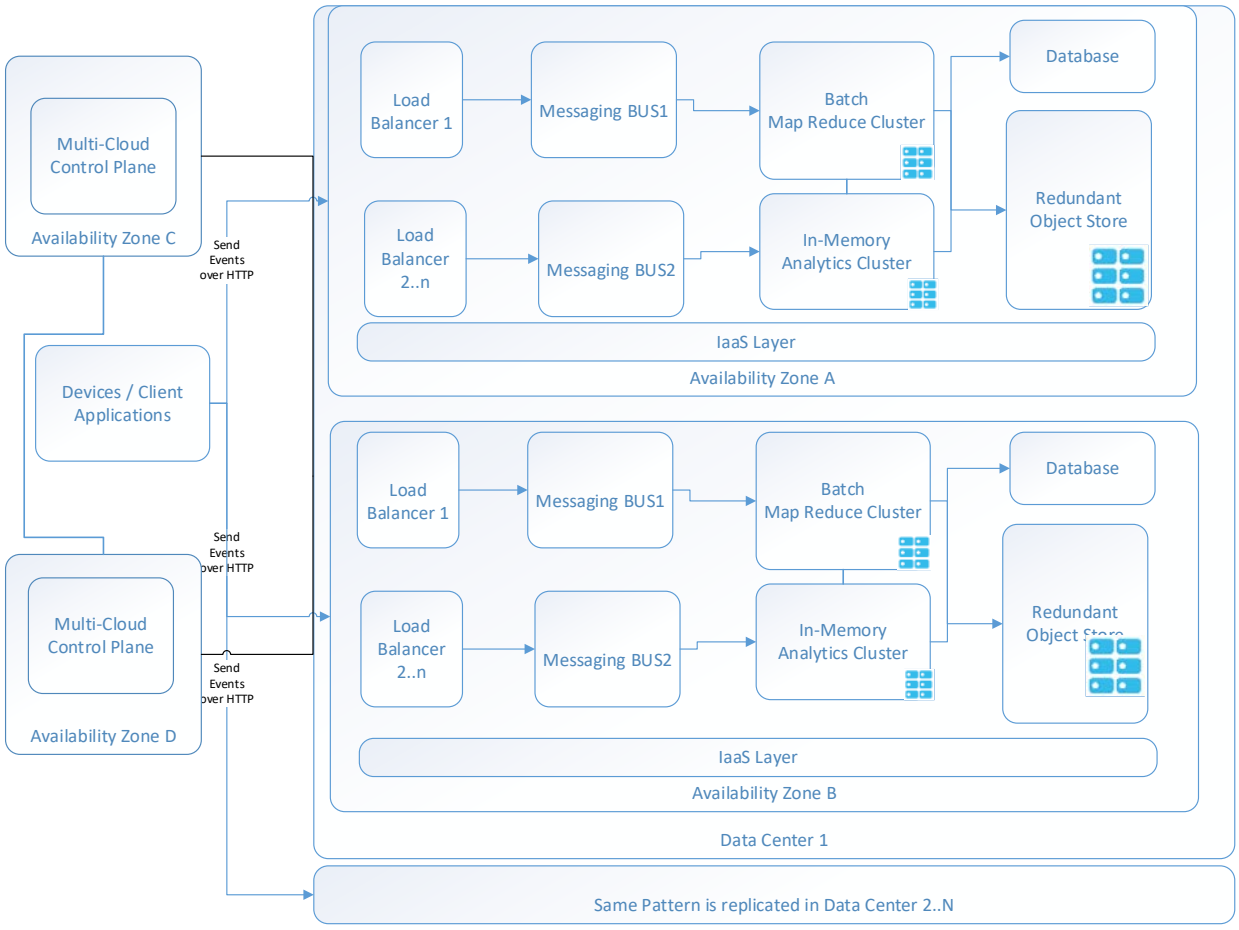


Figure 29 - Multi-Cloud Internet of Things and Big Data Deployment

3.4.5.3 Application Deployment Challenge #3 – Multi-Cloud Deployment and Orchestration with Containers

How do we run Applications or Services Packaged in Containers in Multi-Cloud Environment?

Container is kernel virtualization which focuses on CPU, Memory and Storage isolation. Containers were originally introduced as a concept Solaris as Linux containers i.e. LXC, Docker, Rkt. Once we have IaaS Virtual Machines created then we can deploy containers to subdivide virtual machines. The benefit would be higher utilization and higher multi-tenancy density.

Virtual Machines take from couple of seconds to tens of seconds or even minutes to startup. Is there a faster way to startup and provide virtualization at the same time? Furthermore, the deployment of an application on IaaS are done in virtual machines usually leave the Virtual Machine frequently underutilized. Is there a way to subdivide the Virtual Machine so we can pack more workloads in it vertically a.k.a. vertical scalability?

The core problem is how do you deploy and manage containers in the multi-cloud environment? Container Orchestration Pattern which extends Resource Orchestrator and Resurrection Pattern focusing on deploying, coordinating and resurrecting containers. We will cover the detailed fault tolerant solution to this problem in Chapter 4.

In the meantime, here is the brief graphical preview of the pattern:

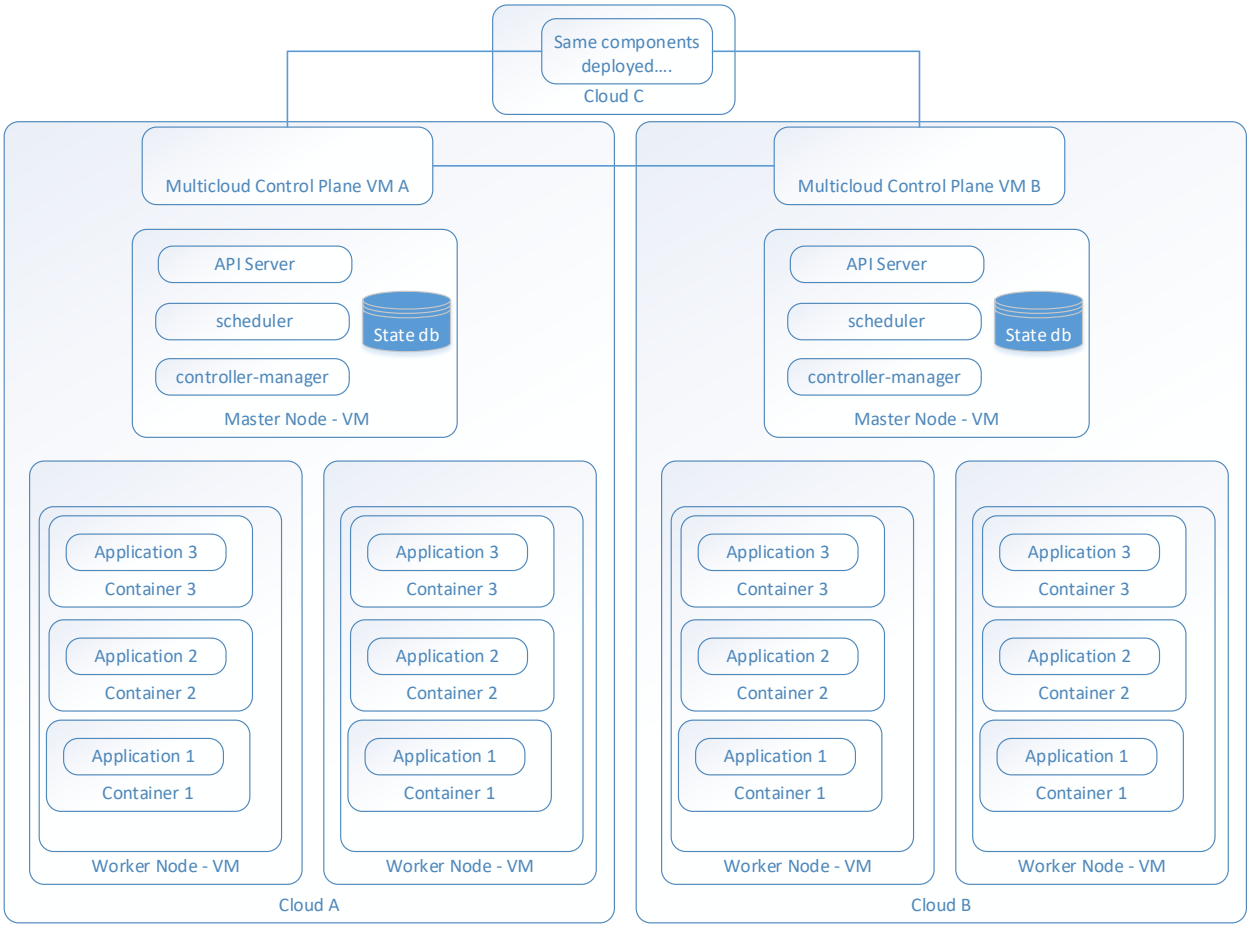


Figure 30 - Multi-Cloud Deployment and Orchestration with Containers

3.5 Summary

In this chapter, we have covered assumptions and objectives for supporting open-source multi-cloud deployment. Following with key challenges while deploying enterprise computing in multi-cloud environment. We have introduced Open-Source Multi-Cloud Deployment Solution Framework that can help us with most of these challenges. Next we introduced solutions to major multi-cloud computing challenges and provided high level solution via an abstract pattern.

Here are how the major challenges / problems we have covered map to the patterns side by side

Multi-Cloud Challenges to the Patterns M 1

Major Problems / Challenges with multi-cloud computing	**Patterns Solution for Each Problem**
Initial Multi-Cloud General Deployment Fault Tolerance Challenges:	**Initial Multi-Cloud Fault Tolerant Deployment Patterns**
• How to deploy with maximum fault tolerance and isolation?	• General Multi Availability Zone Fault Tolerant Pattern
• How do we route to multiple providers all at once with maximum fault tolerance in mind? • If the cloud provider fails what do we do? Can we route the requests and workloads to another cloud provider? • What if Domain Name System fails – how do we continue operating?	• General Multi-Cloud Cloud Fault Tolerant Routing Pattern
• How do we deploy on multiple cloud providers if all of them have different APIs? • More importantly is there a way where you can define your infrastructure requirements once and	• Multi-Cloud Cloud Blueprint Pattern

something will take care of translating this into the API calls for the specific provider?	
• Cloud providers require different virtual machine formats – how do create these images in an automated way?	• Image Build Pipeline Pattern for Multi-Cloud Deployment
• Finally, where do we store and keep track of information about everything we deployed?	• Multi-Cloud Service Registry and Discovery API
Multi-Cloud Management after initial deployment and dealing with failures:	**Multi-Cloud Management after Initial Deployment and Dealing With Failures Patterns**
• Once we deploy how do we find out the health of the instances?	• Reactive Multi-Cloud Health check and Load Balancing Pattern • Multi-Cloud SLA Monitoring Pattern
• If the node virtual machine fail - can we recover them automatically? • If availability zone fails – can we recover automatically? • What if our deployment infrastructure fails – how do we re-deploy? • How do we fail over gracefully when fault occurs?	• SLA Enforcer Rules Engine

• How do we make sure database is always available even if one database instance fails?	• Multi-Cloud Data Replication • Multi-Cloud Disaster Recovery Pattern
• Is there more intelligent way of enforcing Service Level Agreement if we know things are about to fail or there is a degradation in performance – before failure occurs?	• Proactive Multi-Cloud SLA Policy Enforcement Pattern
• What if I want to deploy to public cloud only if private cloud capacity gets exhausted? • What if private cloud fails how can recover from this disaster using a public cloud?	• Public Cloud Bursting Pattern
Cost Efficiency Problems	**Cost Efficiency Patterns**
• How do we actually aggregate billing and cost from all cloud deployments?	• Multi-cloud Aggregate Billing and Chargeback Pattern
• How do we make sure that cost is not out of control and we know how much we are spending as well as deploy workloads where it is cheaper?	• Cost Efficiency Discount Multi-Cloud Pattern
Security Problems	**Security Related Patterns**

• Where do we store and retrieve secrets?	• Secret Storage and Retrieval – Secrets Vault Pattern
• How do we ensure that deployment does not have any known vulnerabilities and gets patched if there is a vulnerability? • How do we continuously audit the cloud and make sure there are no unauthorized or unintended changes?	• Multi-Cloud Auditor Pattern
Application Deployment in Multi-Cloud environment	**Multi-Cloud Application Deployment Patterns**
• Virtual Machines are great, but what if I want to run Applications in Containers – how do I do this in multi-cloud environment with proper fault tolerance in mind?	• Container Orchestration Pattern
• How do we deploy real applications in Multi-Cloud environment so if something fails the user or service does not experience and interruption?	• Multi-Cloud Web Application / Service Pattern
• How to deploy internet of things services in a multi-cloud environment?	• Multi-Cloud Internet of Things Event Stream Ingesting Pattern

In the next chapter let’s do a deep dive into each pattern to provide more details along with reference implementation.

Chapter 4 - Detailed Multi-Cloud Design Patterns and Multi-Cloud Based on Open-Source Technologies

In this chapter we will do a deep dive into each pattern to provide more implantation details. We will start with basic foundational patterns that helps us with initial fault tolerant deployment.

4.1 Multi-Cloud Foundation Patterns focused on Fault Tolerant Deployment Solutions

4.1.1 General Multi Availability Zone Fault Tolerant Pattern

Faults occur in the cloud all the time due to software, hardware or communications failure. The fundamental way to deal with failures in the cloud is via Availability Zones. Availability Zones are isolated locations in data centers that generally have to guarantee fault tolerance via redundant physical racks, its own power provider, network, storage and even telecommunications providers.

How do we deploy in the cloud environment so that one availability zone failure does not result in an outage? The primary solution is isolating faults via availability zones so that all components are deployed in unique availability zones. Let's review the pattern details:

- **Pattern Name and Classification:** General Multi Availability Zone Fault Tolerant Pattern
- **Also Known As:** Multi AZ Deployment Pattern

- **Problem:** Majority cloud provider outages occur at the Availability Zone level. Availability Zone generally can be defined as physically independent location in data center or geographical region that has its own separate hardware rack, independent hypervisor, networking switch, independent power supply and independent network connectivity provider. Deploying all of the virtual machines into single availability zone represents a single point of failure dependency therefore every deployment needs to take in account availability zone.
- **Intent:** To improve fault tolerance even within one cloud provider deployment
- **Motivation (Forces):** You would like to reduce points of cloud provider points of failure and improve resilience of your deployment
- **Applicability:** any multi public/private cloud deployment
- **Structure:** A graphical representation of the pattern.

Sequence diagram of obtaining availability zone information and placement of virtual machines.

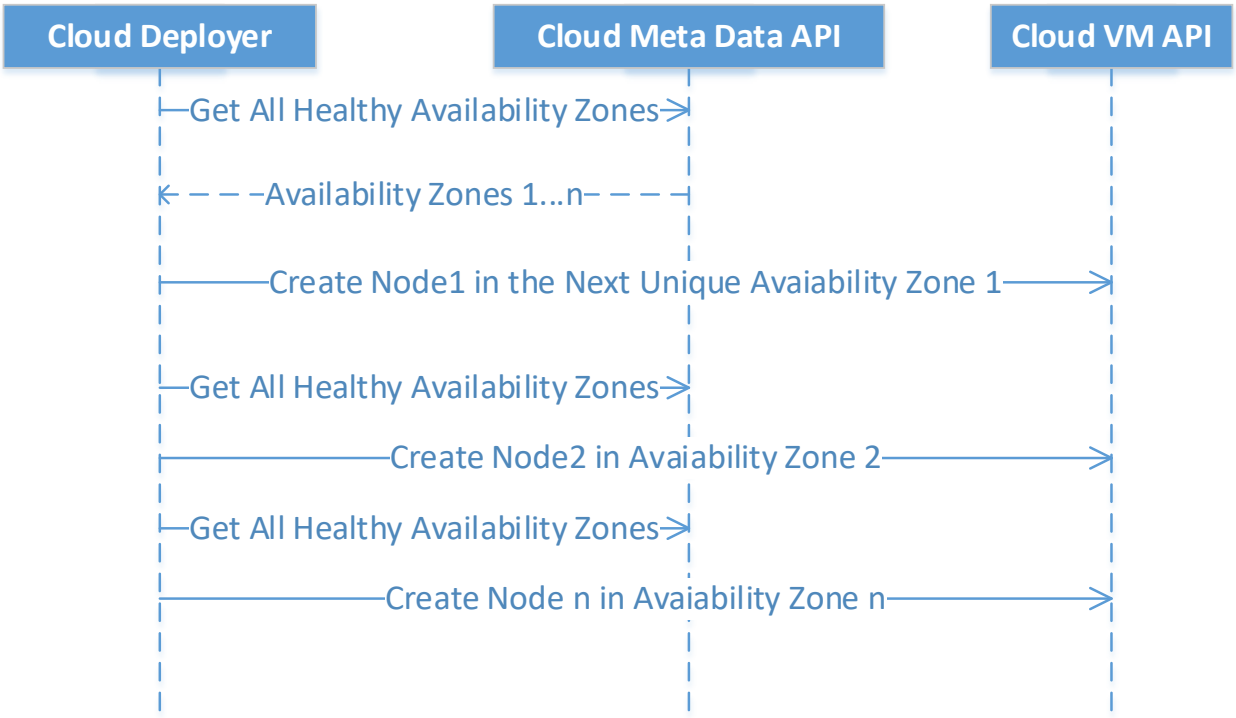


Figure 31 - Sequence Diagram of Obtaining Availability Zone Information and Placement of Virtual Machines

The key concept here is to get the list of all the healthy availability zones before we create any nodes / virtual machines. Another important detail is to pick availability zone that has not been picked before and keep iterating through the list.

So, the simple multi availability distribution algorithm would look something like this:

```
loop for each cloud_provider in the cloud provider list

        loop for each for each virtual machine in the vm list

                availability_Zone_List = cloud_provider.Get_All

                Healthy_Availability_Zones()
```

```
availability_Zone_In_Use = availability_Zone_List.Item

If (last_Used_Availability_Zone != availability_Zone_In_Use)

        //only create in a unique AZ

        Create Node1 in the availability_Zone_In_Use

else

        //we don't want to use previously used availability zone

        print ("skiping AZ" + availability_Zone_In_Use)

last_Used_Availability_Zone = availability_Zone_In_Use

//keep Iterating through the list
```

Keep going down the list till we find AZ that has not been just picked. Eventually some availability zones will have to be re-used, but we will have a good even distribution of nodes in each.

Most major cloud providers support this API call although these are generally not standard – for example in AWS the call via CLI will be

describe-availability-zones

returning JSON array as output

```
{
   "AvailabilityZones": [
      {
```

```
        "State": "available",
        "RegionName": "us-east-1",
        "Messages": [],
        "ZoneName": "us-east-1b"
    },
    {
        "State": "available",
        "RegionName": "us-east-1",
        "Messages": [],
        "ZoneName": "us-east-1c"       }, …    ] }
```


Reference:

http://docs.aws.amazon.com/cli/latest/reference/ec2/describe-availability-zones.html

However even if availability zone is healthy we need to check health of virtual machines and applications running on each one of the nodes. True health check pattern needs to include the highest component in the stack and all of the dependencies such as database:

Multi Availability Zone Deployment with Health Checking

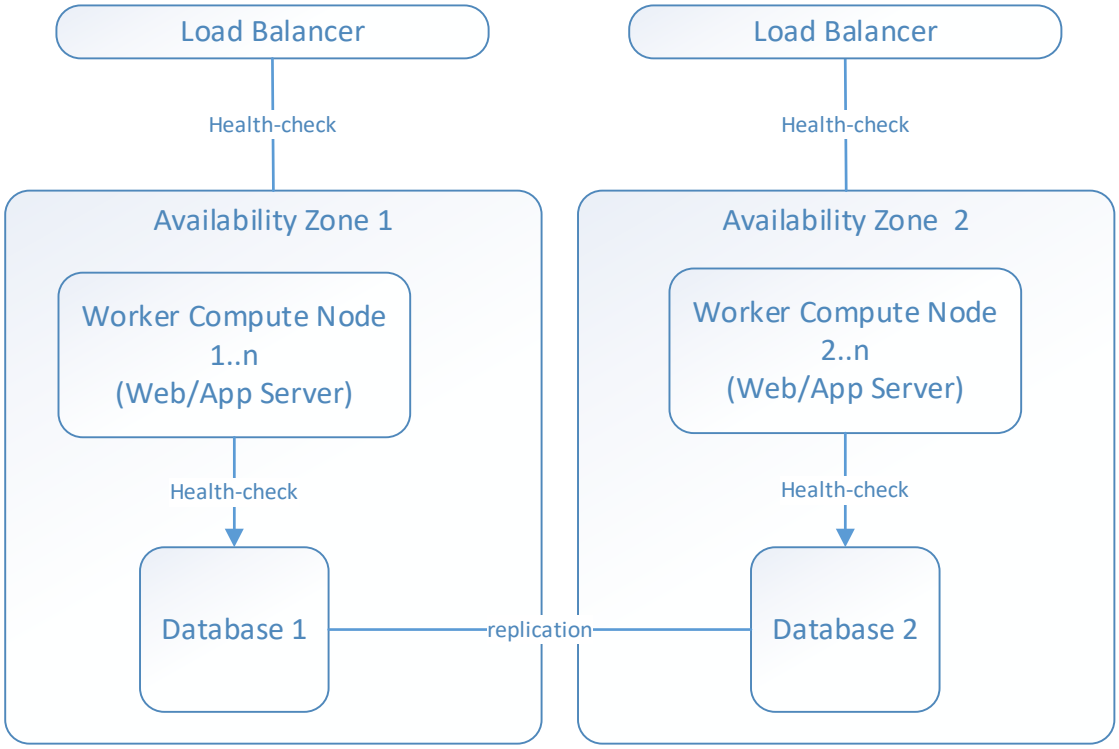


Figure 32 - Multi Availability Zone Deployment with Health Checking

Please note since most application rely on a database for high availability and fault tolerance the minimum we will need is replication so that even if one Availability Zone deployment fails we can keep processing in another.

Load Balancers can be also deployed in same Availability Zones as depicted below.

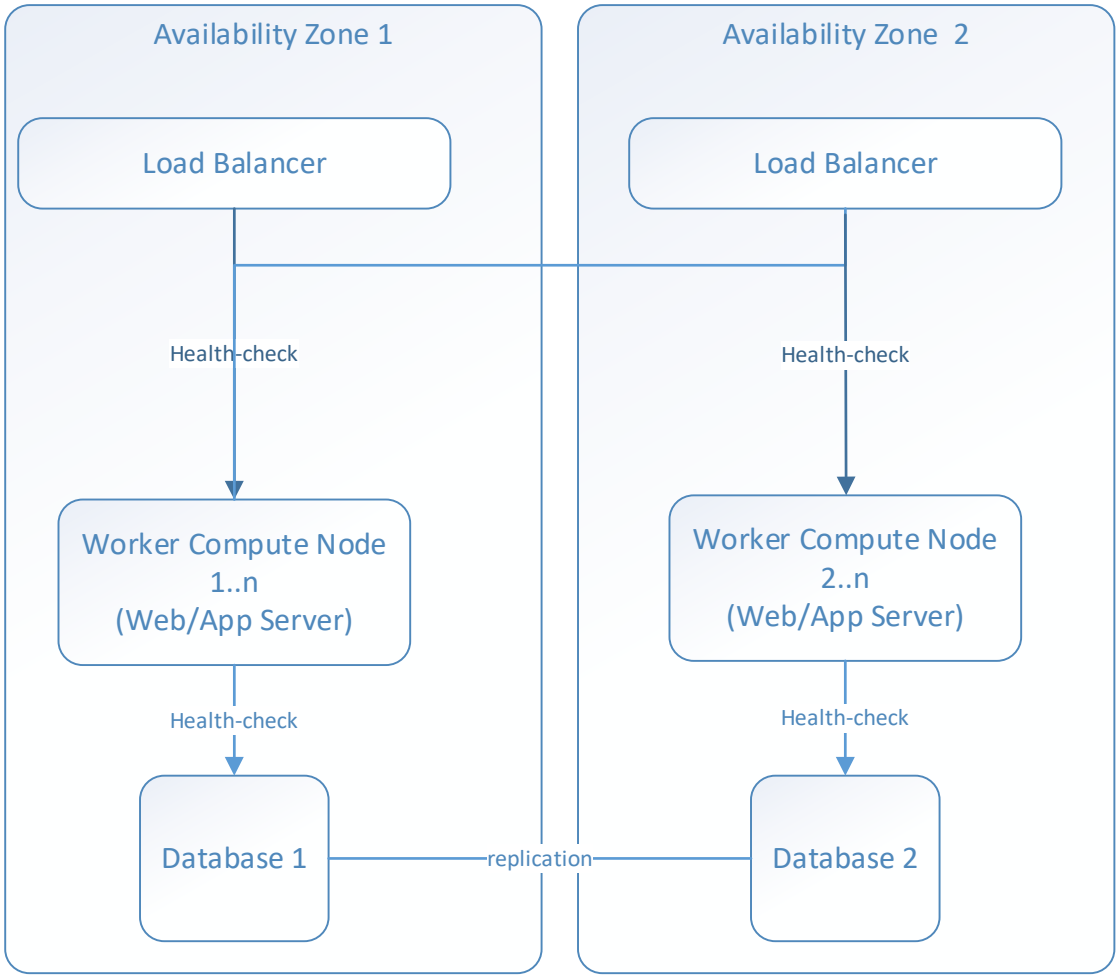


Figure 33 - Multi Availability Zone Deployment with Load Balancers and Health Checking

However, if there is negligible latency across availability zones it's better to keep these in separate availability zones in the geographical region/data-center if there is negligible latency across availability zones.

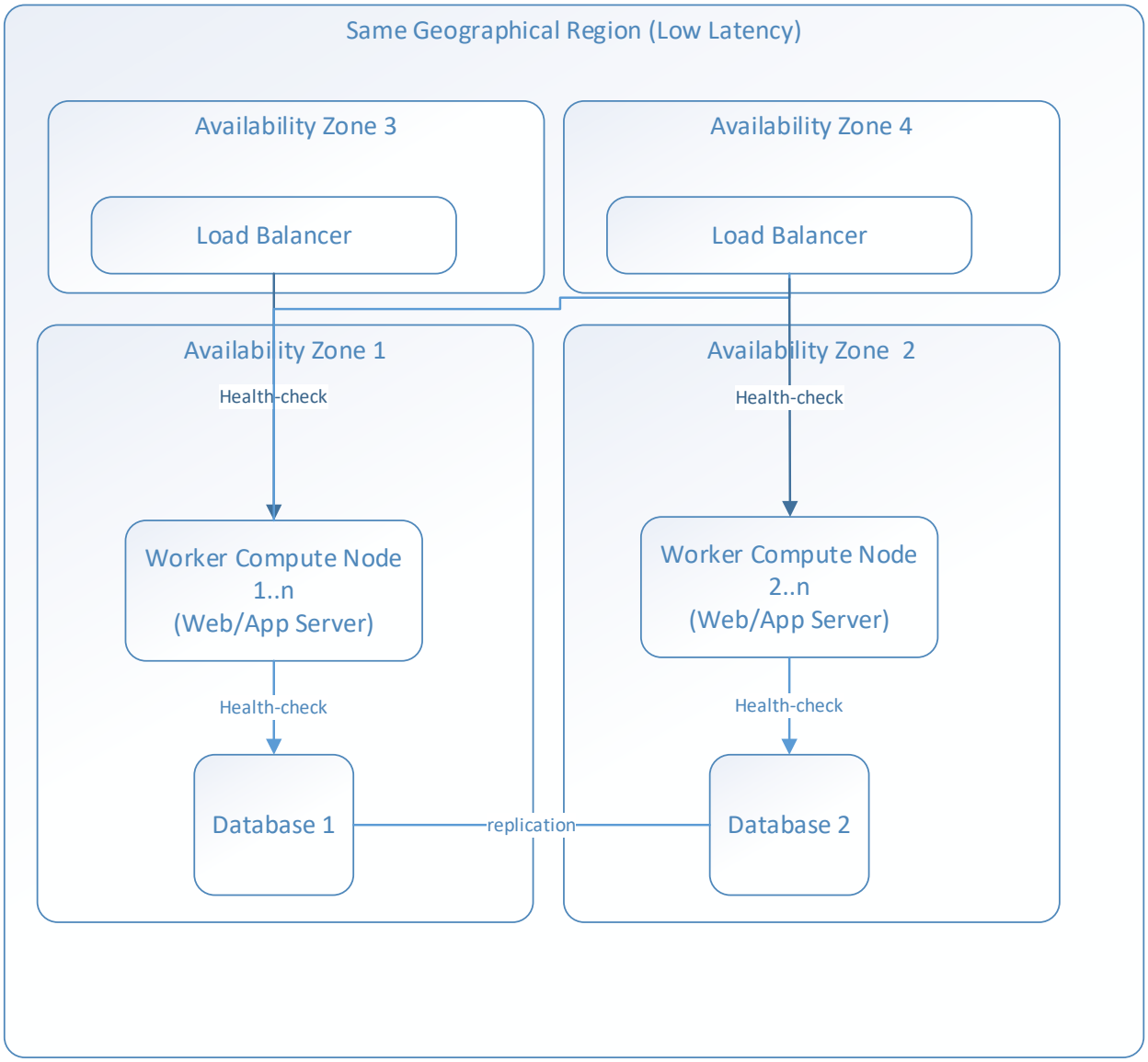


Figure 34 - Same Geographical Region Deployment for Low Latency

Next let's introduce how this pattern fits in the multi-cloud deployment context. As we deploy to multiple clouds we need to query the availability zone metadata APIs and distributed the nodes so that not all get deployed to the same availability zones.

Here is an example of multi-cloud deployment:

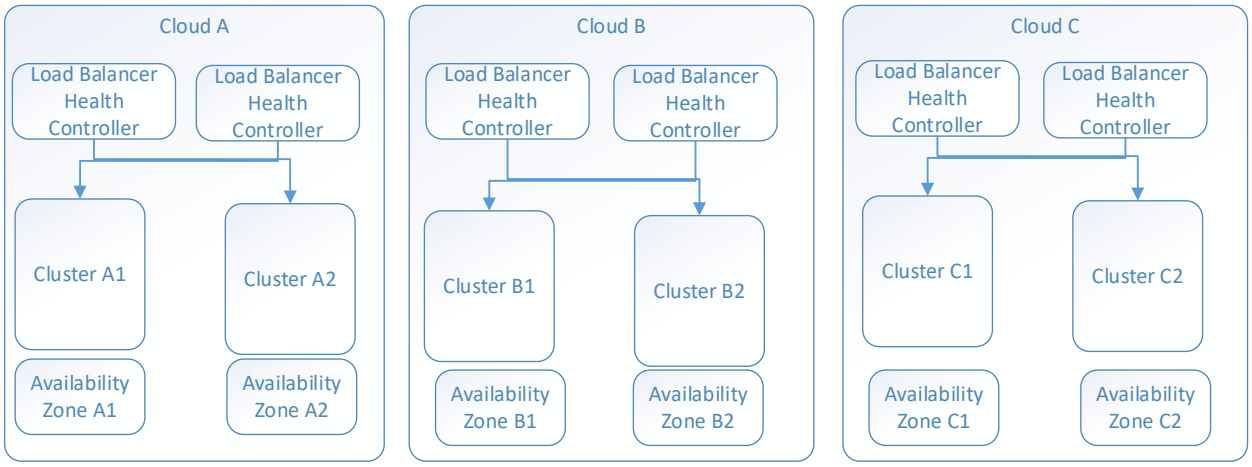


Figure 35 - Multi-Cloud Deployment Example

Multi-Cloud and Multi-Availability Zone Deployment

- **Participants:** Multi-Cloud Deployer / Orchestrator, SLA Enforcer Rules Engine , IaaS Bootstrap Blueprint (VMs, Compute, Network, Storage, Security Groups), Image Pipeline (VM/Container)
- **Consequences:** Virtual Machines, Storage, Networking, Security Groups, and Entire software stack above Virtual Machine is deployed in independent availability zones
- **Implementation:** all major cloud providers (and frameworks) support availability zones: AWS, Google, Azure, OpenStack etc. However, since all the APIs are different we need an abstraction framework to keep it cloud neutral.

  Some AZ aware deployment frameworks that will be covered later on include:

  - Bosh (http://bosh.io/)
  - Terraform (https://www.terraform.io/)

However, these frameworks do not provide out of the box AZ proper fault distribution and require additional specific configuration.

- **Known Uses:** most public cloud or private cloud IaaS support availability zones
- **Related Patterns:** SLA Enforcer Rules Engine . SLA Enforcer Rules Engine which will be covered later on will compliment this pattern in cases when availability zone failure occurs and we need to re-deploy to a different availability zone.

Next let's take a look at how do we distribute traffic and route among different cloud providers with fault tolerance in mind?

4.1.2 General Multi-Cloud Fault Tolerant Routing Pattern

Now that we have covered foundational availability zone pattern how do we apply this to the multi-cloud deployment especially how do we route distribute traffic in a fault tolerant way. Single cloud provider or private data center is not sufficient for fault tolerance and geographical availability. How do we distribute traffic and route among different cloud providers with fault tolerance in mind? Domain Name System is key to discovery and routing. Single DNS provider can be a single point of failure especially if undergoes outage or denial service attack how can we design and manage DNS in multi-cloud environment?

- **Pattern Name and Classification:** General Multi-Cloud Routing Pattern
- **Also Known As:** Hybrid / Multi-Cloud Pattern DNS Routing Pattern
- **Problem:** Single cloud provider or private data center is not sufficient for fault tolerance and geographical availability. How do we distribute traffic and route

among different cloud providers with fault tolerance in mind? Single DNS provider can be a single point of failure especially if undergoes distributed denial service attack (DDoS).

- **Intent:** To improve fault tolerance, geographical availability, reduce vendor lock-in and mitigate distributed denial of service attack.
- **Motivation (Forces):** improve fault tolerance, geo availability, reduce vendor lock-in
- **Applicability:** any multi public/private cloud deployment
- **Structure:** A graphical representation of the pattern.

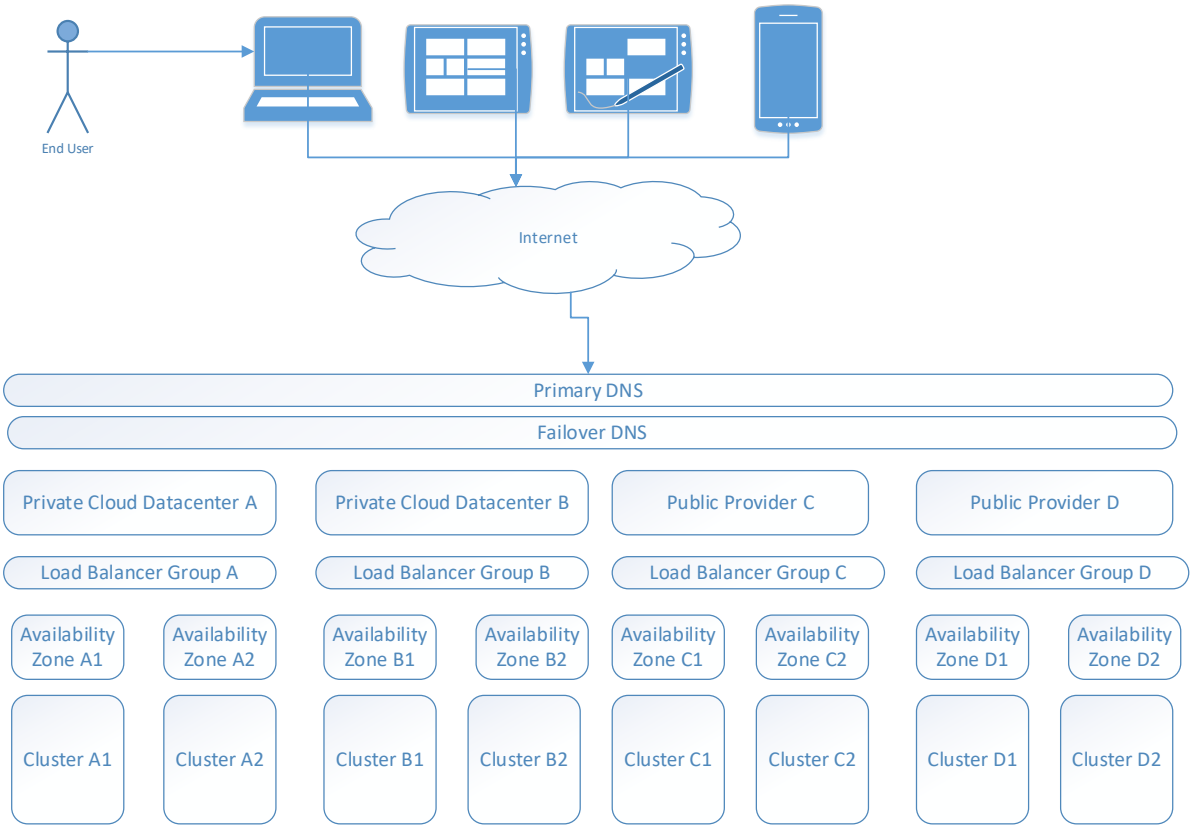


Figure 36 - DNS Only Multi-Cloud Fault Tolerant Routing Pattern

What happens if our DNS failover fails? If we are able to control TCP/IP/HTTP client on the device we can have an API with well-known public IPs that device can cache that will be available to replace DNS. In this scenario, the device will have to periodically download the list of known IP addresses of known API end points that can be used to bypass DNS in case DNS providers fail or under DDoS attack.

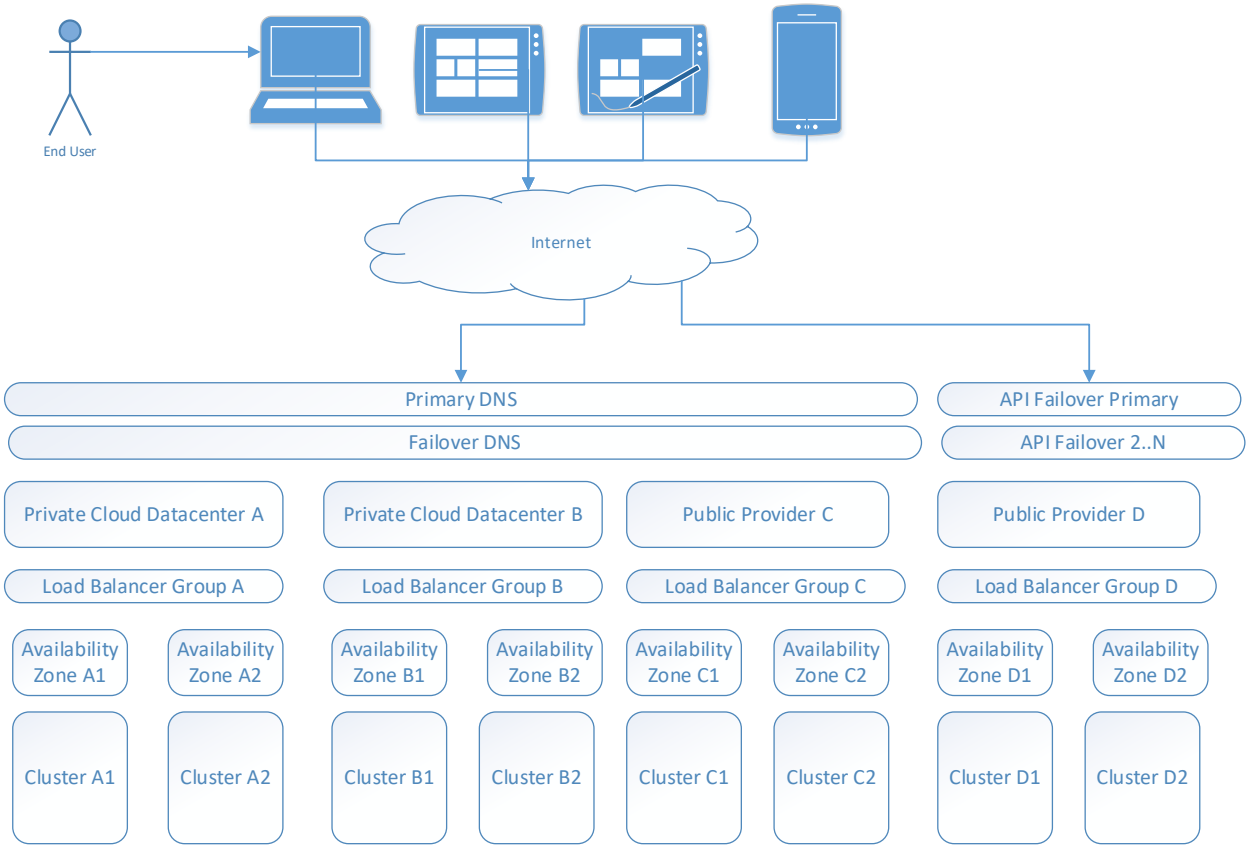


Figure 37 - Multi-Cloud Fault Tolerant Routing Pattern with DNS and API Failover

However, to make the API more Resilient we would want to deploy it across all cloud instances

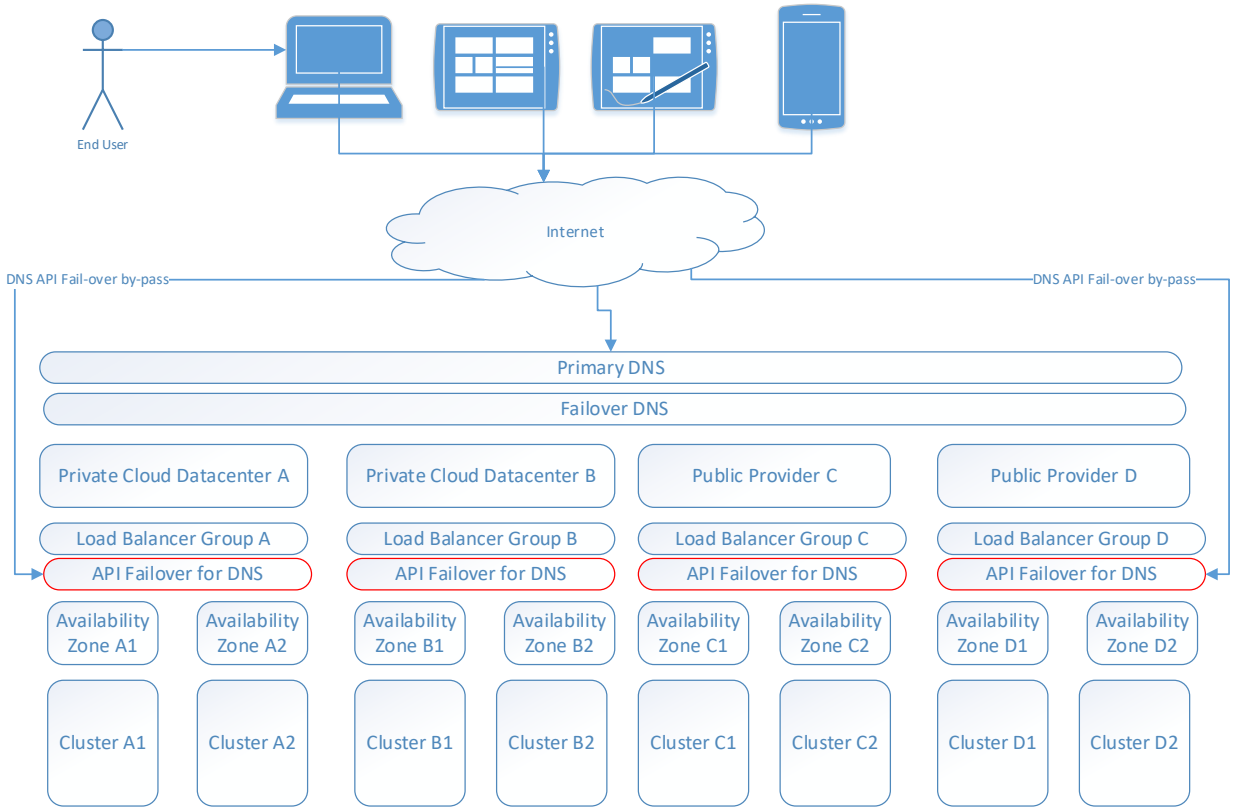


Figure 38 - Multi-Cloud Routing API Failover Deployment

- **Participants:** Internet, Domain Name System, Geographic Load Balancers, Public and Private Data Centers, Availability Zones (isolated physical racks, with its own power provider, network, telecom providers)
- **Consequences:** Applications deployed in multi-cloud environment get the benefit of better fault tolerance, disaster recovery, less vendor lock in and better geographical availability and lower latency.
- **Implementation:** Cedexis, Akamai FastDNS

Multi-Cloud DNS Implementation Example

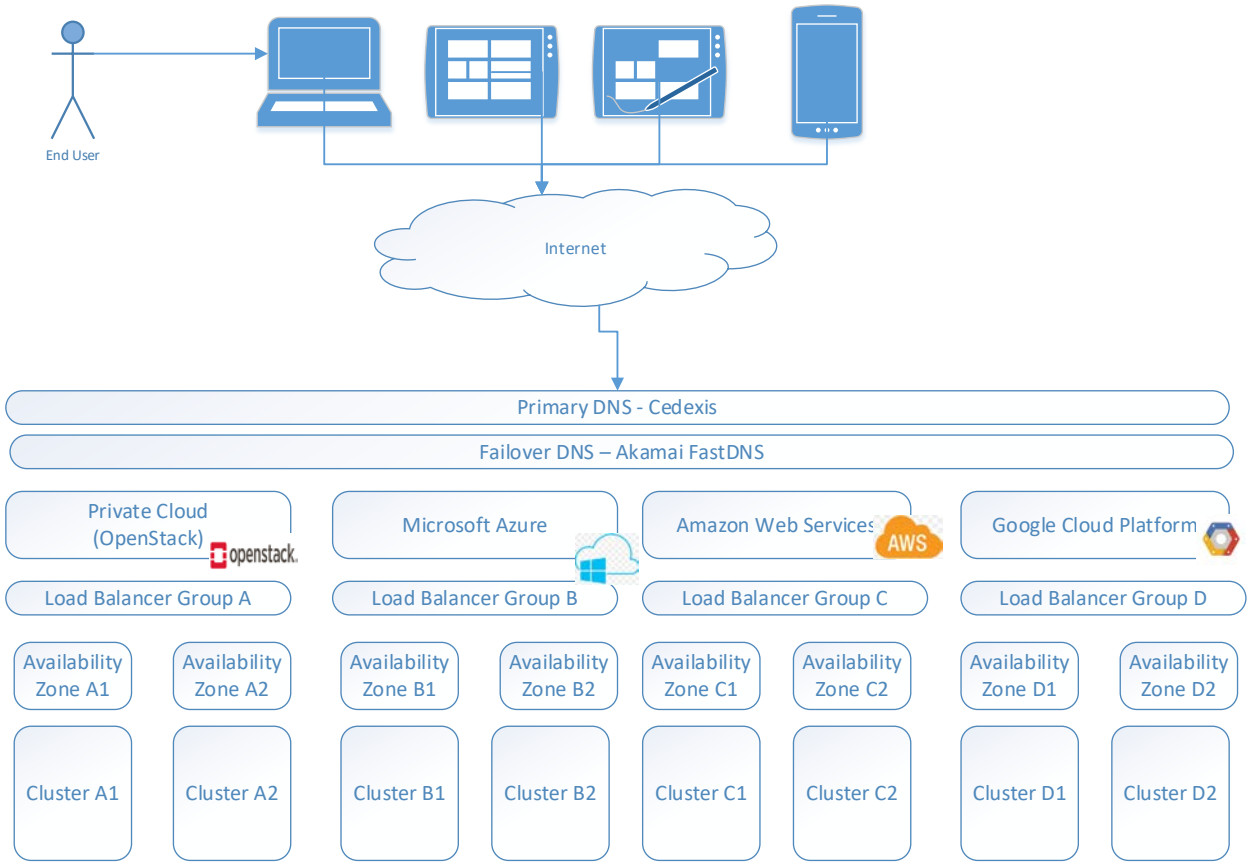


Figure 39 - Multi-Cloud DNS Implementation Example

DNS and API Fail-over Implementation Example

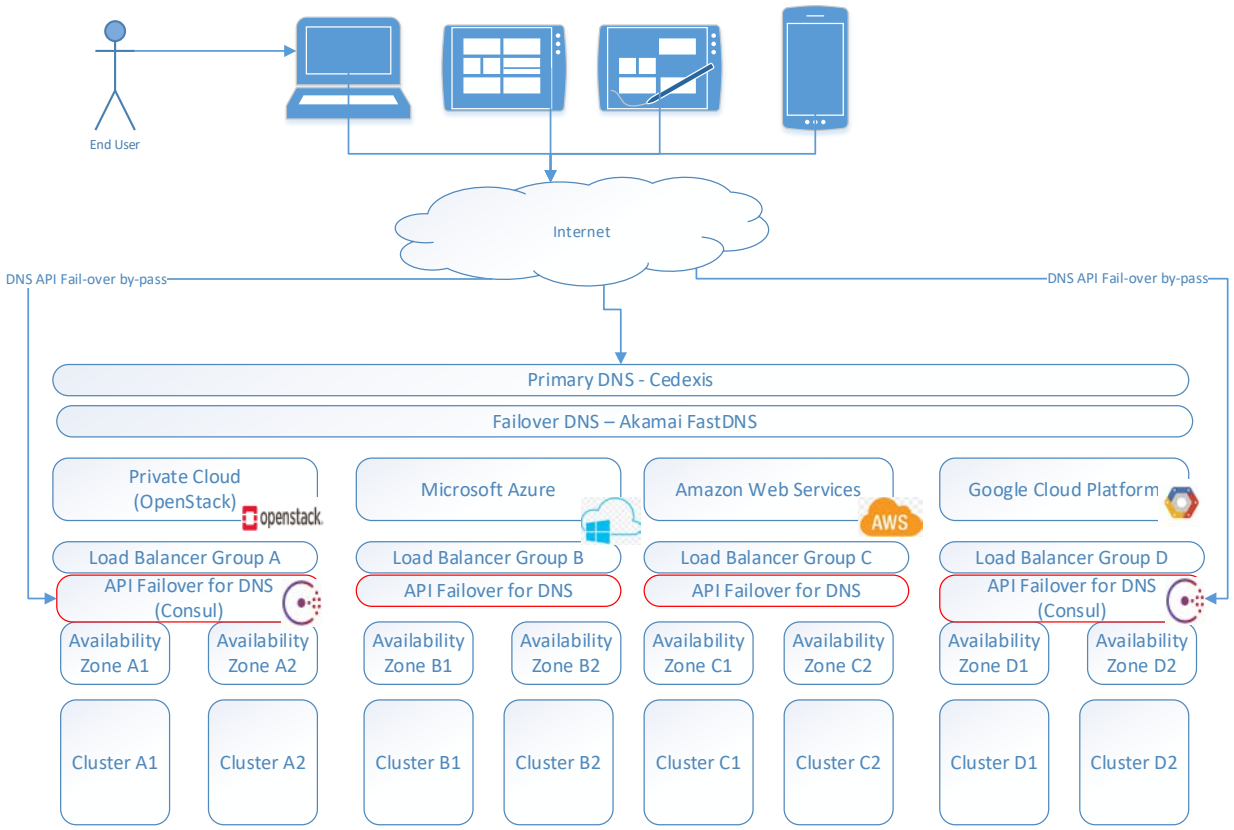


Figure 40 - DNS and API Fail-over Implementation Example

- **Known Uses:** Examples of real usages of the pattern.

  Global DNS Highly Available providers:

  Akamai FastDNS https://www.akamai.com/us/en/solutions/products/cloud-security/fast-dns.jsp

  Cedexis http://www.cedexis.com/products/openmix/

  Nustar UltraDNS https://www.neustar.biz/services/dns-services

  Multi-Cloud API - https://www.consul.io

- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

Now that we have introduced the foundational fault tolerant patterns, let's take a look how we build up multi-cloud deployment with the Multi-Cloud Cloud Blueprint Pattern on top of Infrastructure as a Service cloud provider.

4.1.3 Multi-Cloud Cloud Blueprint Pattern

How do we describe what desired deployment should be on multiple cloud providers if all of them have different non-standard blueprint formats?

How do we define your infrastructure requirements once and something will take care of translating this into cloud specific blueprint for a specific provider? This way you don't need to maintain IaaS definition for multiple cloud providers. In order to automate provisioning of multiple IaaS this pattern helps to describe your cloud infrastructure as a blueprint and use that blueprint to drive orchestrator to target specific IaaS provider and create all the resources necessary for your application and services to run on any cloud provider (Public or Private). Currently there is no uniform API and it is not uniformly implemented in the same way across cloud providers

- **Pattern Name and Classification:** Cloud IaaS Tenancy Blueprint Bootstrap Pattern
- **Also Known As:** Cloud Infrastructure as a Service Blueprint Bootstrap Pattern
- **Problem:** Currently there is no uniform Cloud deployment format or API and it is not uniformly implemented in the same way across cloud providers. Most cloud provider's APIs are different and non-standard. To solve this problem, we need to have a layer of abstraction independent from specific cloud provider API. Ideally you would define your cloud blueprint once and automate any cloud provider specific calls. In order to automate provisioning of multiple IaaS this pattern helps to

describe your cloud infrastructure as a blueprint and use that blueprint to drive orchestrator to target specific IaaS provider and create all the resources necessary for your application and services to run on any cloud provider (Public or Private).

- **Intent:** Describe your IaaS as Cloud Independent Blueprint
- **Motivation (Forces):** Multi-cloud deployment blueprint that can be used on most cloud providers
- **Applicability:** Any multi-cloud deployment including private and public IaaS
- **Structure:** A graphical representation of the pattern.

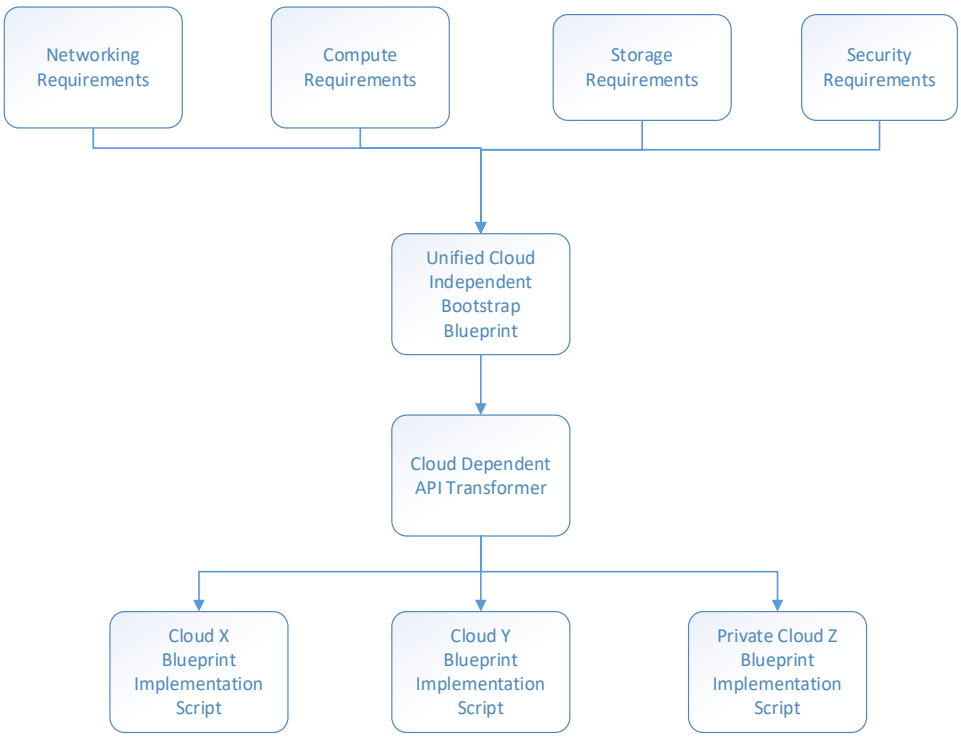


Figure 41 - Multi-Cloud Cloud Blueprint Pattern

- **Participants:**
  Networking Blueprint, Compute Blueprint, Storage Blueprint, Security Blueprint,

Unified Cloud Independent Bootstrap Blueprint, Cloud Dependent API Transformer

Cloud Blueprint Implementation Script

- **Consequences:** Result of this pattern is the particular cloud stack you need is created in multiple cloud providers and fully operational
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Example Deployment Options that are somewhat similar to this pattern with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/cloud-blueprint-bootstrap-options

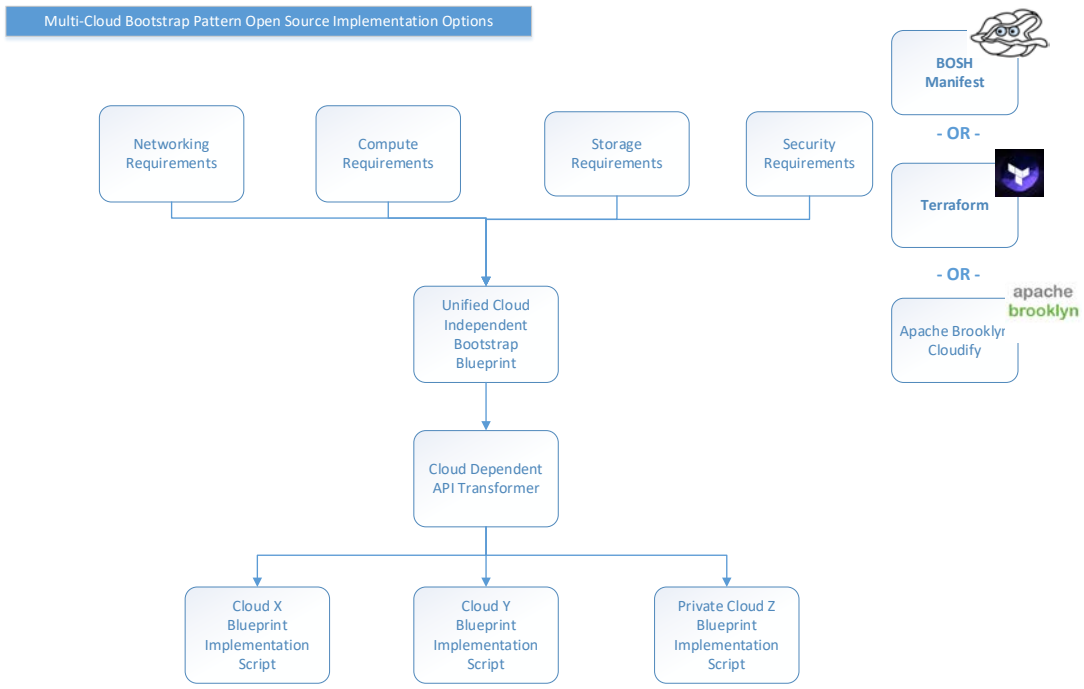


Figure 42 - Multi-Cloud Bootstrap Pattern Open Source Implementation Options

- **Known Uses:**

Not all projects listed here implement the pattern exactly as described, but generally close.

- Apache Brooklyn https://brooklyn.apache.org
- Bosh (http://bosh.io)
- Terraform (https://www.terraform.io/)
- OpenStack Heat templates based on cloud CAMP standard (https://wiki.openstack.org/wiki/Heat)

- **Related Patterns:** Image Build Pipeline pattern

Next let's talk about how do we provide images that can run on any cloud provider?

4.1.4 Image Build Pipeline Pattern for Multi-Cloud Deployment

So now that we have a blue print describing how we creating a deployment on each IaaS cloud provider – what specifically are we going to deploy? Cloud providers require different virtual machine formats – so how do create these images in an automated way? In addition, by having this IaaS specific virtual machine image this will help us to be able to quickly reference this image if a failure occurs and re-build virtual machines. To make it even more efficient the virtual machine image should contain:

- Operating System (i.e. Linux)
- Runtime Environment (i.e. Java Runtime, Tomcat, Apache etc.)
- Application

By having the above created in one image we achieve an immutable infrastructure where components do not need to be changed but rather simply replaced with the entire image that we know is in a good state.

- **Pattern Name and Classification:**

  Image Build Pipeline Pattern for Multi-Cloud Deployment

- **Also Known As:** Cloud Image Pipeline

- **Problem**: How do we provide images that can run on any cloud provider? Image Build Pipeline Pattern is focused on building ready to boot images that contain entire software stack and even can contain application itself. (i.e. OS, Middleware, Application etc.) It is very important to have ready to boot image if you have a spike in load (via user requests, events etc.) and you need to scale out and create new instances quick. During run-time, this pattern relies on Resource Orchestrator Pattern. The build pipeline also might be tailored to produce multiple images targeting multiple providers and formats in order to gain the most efficiency and exploit any cloud provider format or API differences. There is no uniform API and it is not uniformly implemented in the same way across cloud providers.

- **Applicability:** Virtual Machine Images, Container Images (i.e. LXC, Docker) as well as Non-Virtualized Images

- **Intent:** Build images (virtual machine or containers) that can run on any cloud or virtualized infrastructure

- **Motivation (Forces):** you would like to use one process to target multiple IaaS layers
- **Applicability:** multi-cloud / hybrid deployment
- **Structure:** A graphical representation of the pattern.

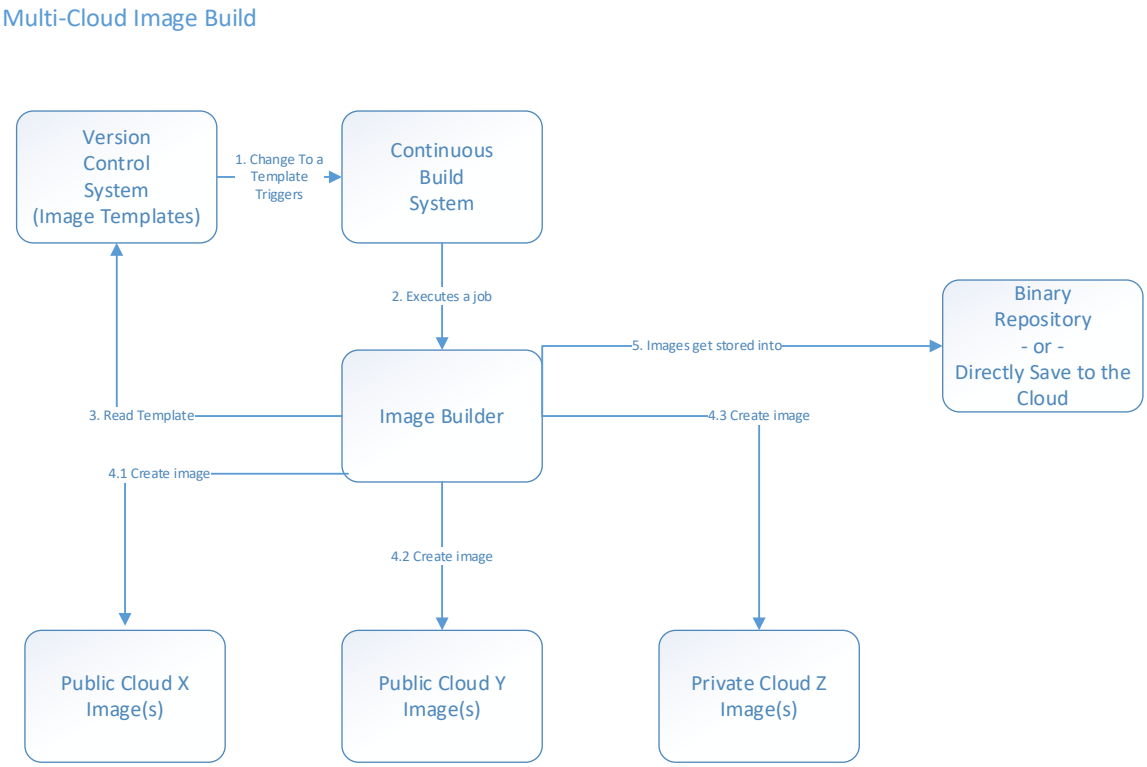


Figure 43 - Multi-Cloud Cloud Image Build Pattern

- **Participants:** Image Builder, Public/Private Cloud Providers, Binary Repository, Version Control System
- **Consequences:** Using templates multiple images get produced to be used in any public cloud or on premises/private cloud
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Example Deployment Options that are somewhat similar to this

pattern with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/image-pipeline

Next let's take a look how an example implementation of the pattern would look like with Open Source Software and images appropriate for Multi-Cloud deployment:

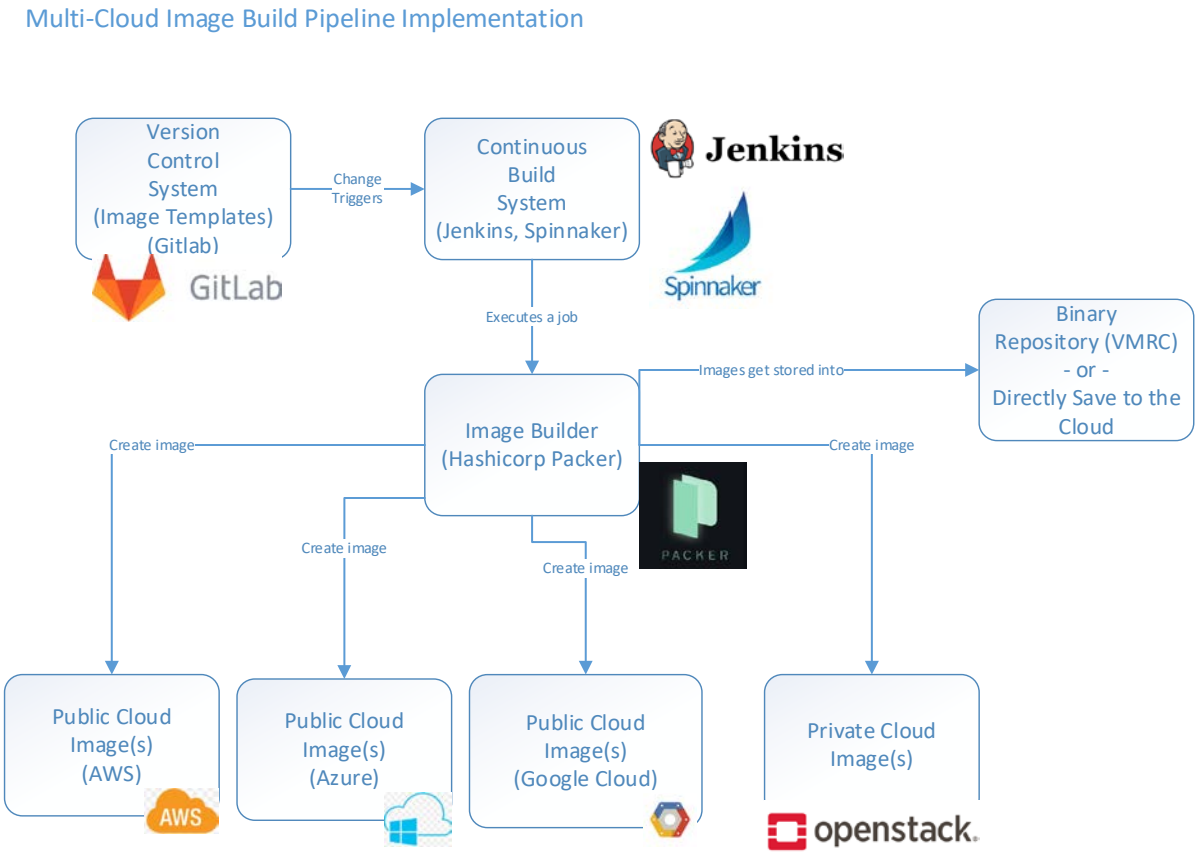


Figure 44 - Multi-Cloud Cloud Image Build Pattern Implementation with Open Source Software

- **Known Uses:** Spinnaker http://www.spinnaker.io, Packer http://www.packer.io
- **Related Patterns:** CI/CD (Continuous Integration/Continuous Build) pipeline

So now that we have blueprint and images to deploy – what does the actual deployment?

4.1.5 Multi-Cloud Node Deployer - Orchestrator Pattern

Now that we covered image creation for multiple cloud providers we need a component that will do the deployment to each cloud provider. Deployment is somewhat trivial and sometimes we need to do more than deployment – we need orchestration. Orchestration can be defined as deployment and coordination of the components to produce a desired or target state. Desired or target state refers to what we have described in the cloud blueprint i.e. x number of components describing Compute, Networking, Storage, Security. So Deployer/Orchestrator will maintain desired state based on the cloud blueprint/manifest as well as additional rules defined via a rules engine that come out of SLA Enforcer Rules Engine pattern that will be described later on.

- **Pattern Name and Classification:** Multi-Cloud Node VM Deployer - Orchestrator Pattern
- **Also Known As:** Multi-Cloud VM Orchestrator Pattern
- **Problem:** You need to deploy on top of multiple private/public cloud IaaS to make it fully available with any software stack that is able to run applications or services. The deployer needs to know about specific cloud provider API.
- **Intent:** Resource Deployer - Orchestrator Pattern is akin to the main builder of all of architecture that was defined in patterns before. The orchestrator is able to initially deploy full stack Virtual Machine images and spin them up as instances.
- **Motivation (Forces):** multi-cloud deployment of the same exactly software stack
- **Applicability:** private/public cloud IaaS multi-cloud deployment
- **Structure:** A graphical representation of the pattern.

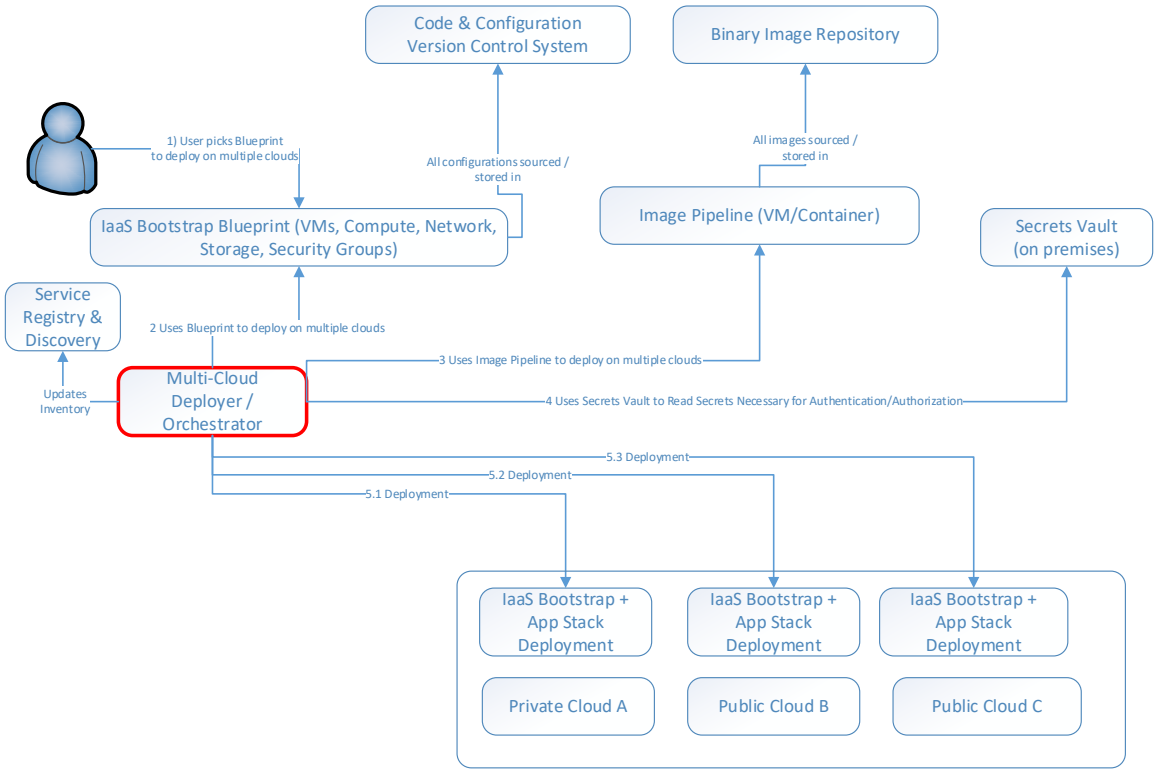


Figure 45 - Multi-Cloud Node Deployer - Orchestrator Pattern

As mentioned before the key to fault tolerant deployment is availability zone information and placement of virtual machines in each unique healthy availability zone

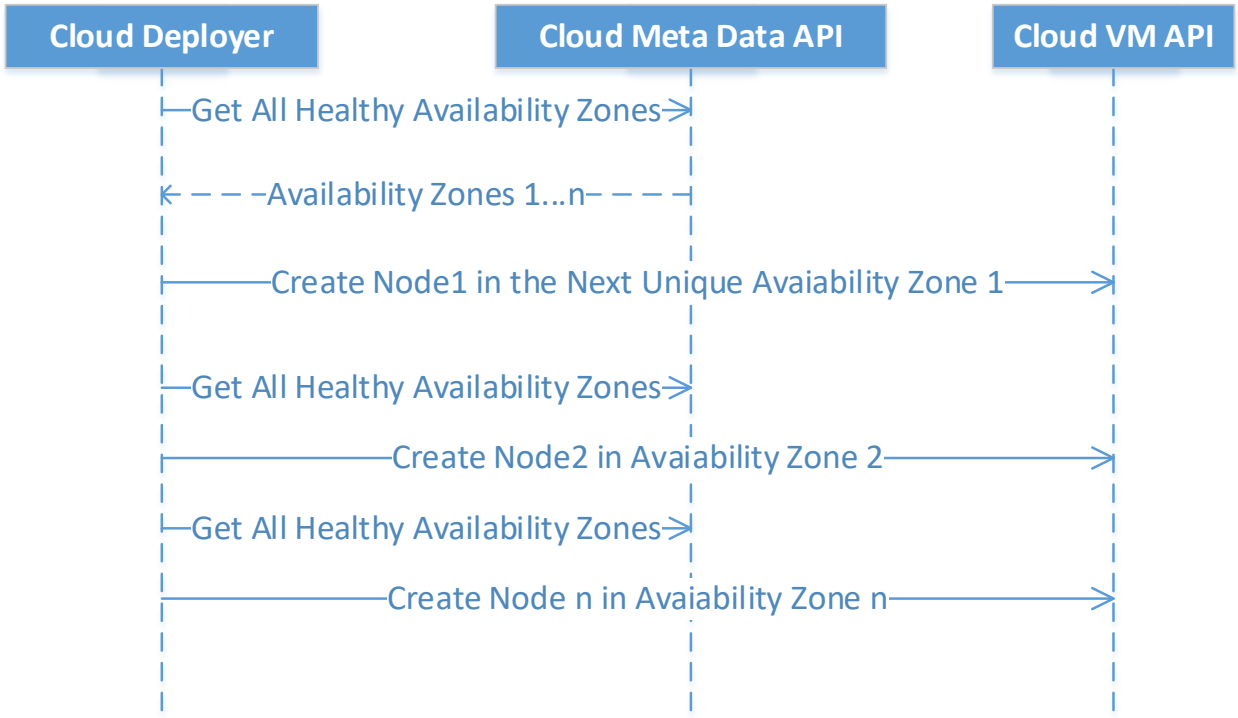


Figure 46 - Multi-Cloud Node Deployer - Orchestrator Pattern with Multi Availability Support

- **Participants:** Multi-Cloud Deployer / Orchestrator, SLA Enforcer Rules Engine , IaaS Bootstrap Blueprint (VMs, Compute, Network, Storage, Security Groups), Image Pipeline (VM/Container)
- **Consequences:** Virtual Machines, Storage, Networking, Security Groups, and Entire software stack above Virtual Machine is deployed
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.

  Example Deployment Options that are somewhat similar to this pattern with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/cloud-blueprint-bootstrap-options

- **Known Uses:** Terraform, Bosh (bosh.io), Apache Brooklyn (https://brooklyn.apache.org), Cloudify (http://cloudify.co)
- **Related Patterns:** SLA Enforcer Rules Engine

So now that we have deployed on multiple cloud providers – where do we store and keep track of information about everything we deployed? Where is our inventory of what's running where? How do we find X? Where do we register newly deployed components?

4.1.6 Multi-Cloud Service Registry and Discovery API

Multi-Cloud Service Registry and Discovery API will enable us to store and keep track of information about everything we deploy. It is our inventory of what's running where. It will help us to find/discover what's running where. It will help us register newly deployed components. All these use cases can be solved via Multi-Cloud Service Registry and Discovery API Pattern.

- **Pattern Name and Classification:** Multi-Cloud Service Registry and Discovery API
- **Also Known As:** Distributed Service Registry and Discovery
- **Problem**: How do we keep track of everything that we have deployed in multi-cloud environment? Multi-Cloud Service Registry and Discovery pattern allows us to

keep track of everything that we deploy. Think of this as distributed registry and latest source of state that has an API so that any components in multi-cloud control plan can use create/read/update/delete operation. When a new component comes alive it can always use lookup via DNS the Registry and the query for any component or meta-data service that it needs to connect to.

- **Applicability:** Any Distributed or Multi-Cloud deployment.
- **Intent:** Distributed Service Registry, Inventory and Discovery
- **Motivation (Forces):** you would like to keep track of your cloud and keep up to date your distributed system inventory
- **Applicability:** multi-cloud / hybrid deployment
- **Structure:** A graphical representation of the pattern.

Now that we covered the fundamentals let’s take a look at how this pattern fits in the entire multi-cloud control plane framework:

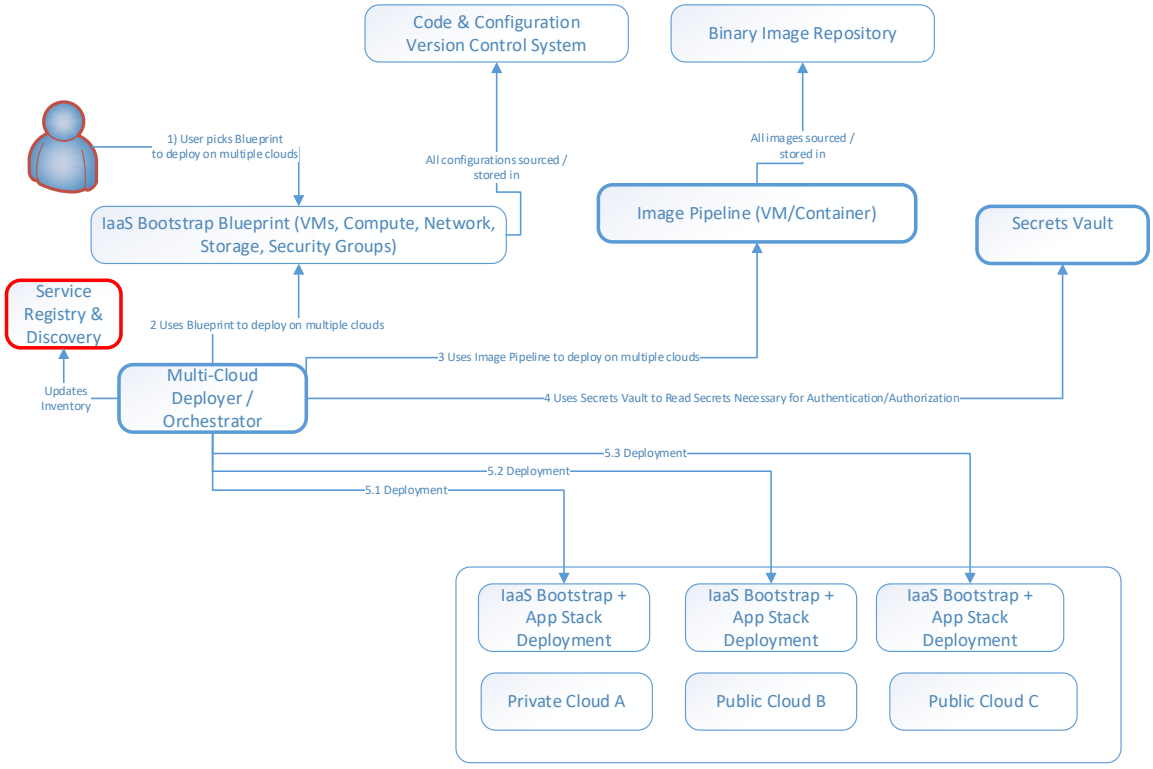


Figure 47 - Multi-Cloud Service Registry and Discovery API with Multi-Cloud Control Plane

- **Participants:** Service Registry, HTTP API Server
- **Consequences:** Registry is deployed and is accessible via the HTTP API Server
- **Implementation:** Here is the multi-cloud deployment of the pattern

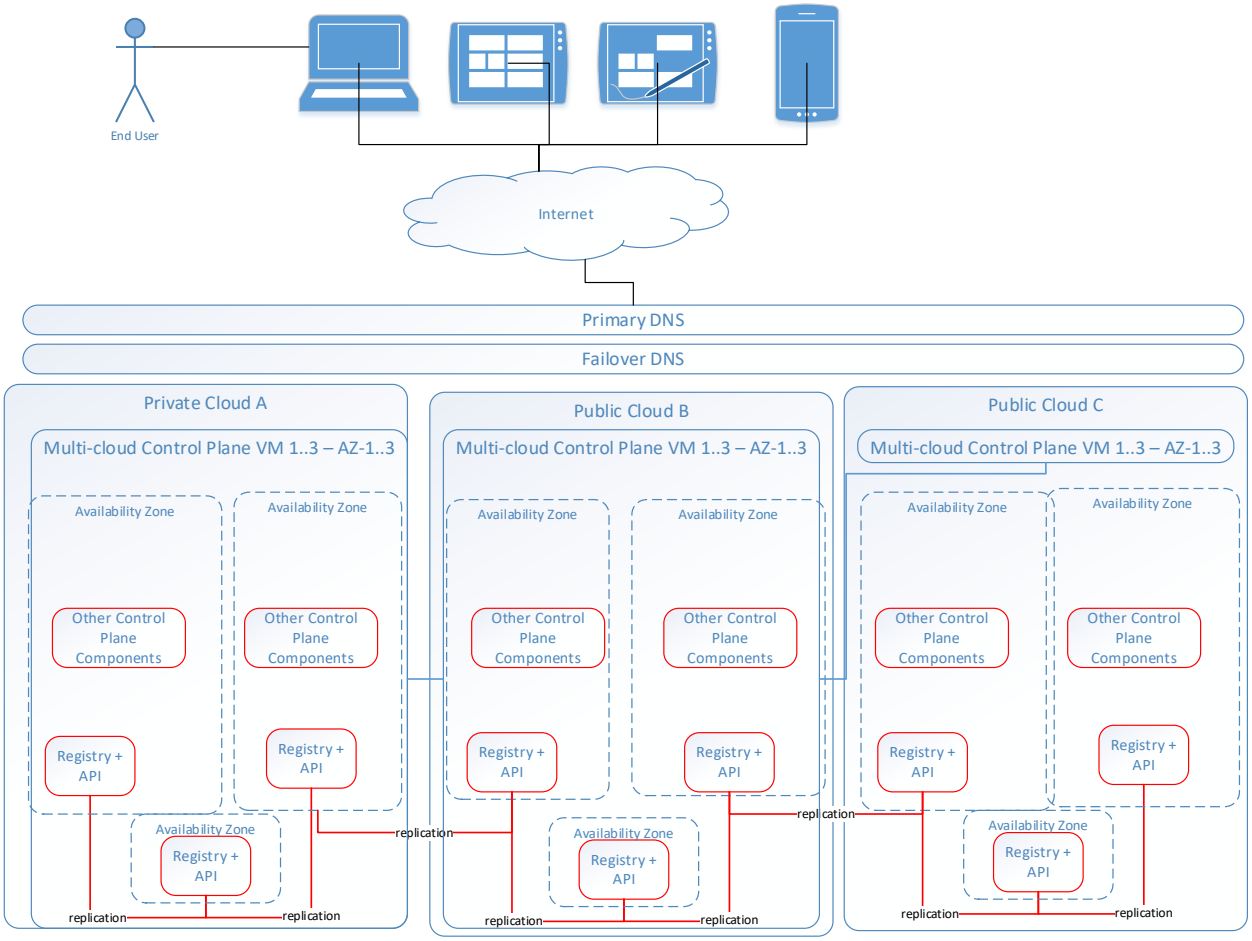


Figure 48 - Multi-cloud Registry and API High Availability Deployment

Multi-cloud Registry and API needs to be highly distributed, replicated and resilient as it contains latest state of deployment and where to locate services.

Specific Technology Mapping to Consul

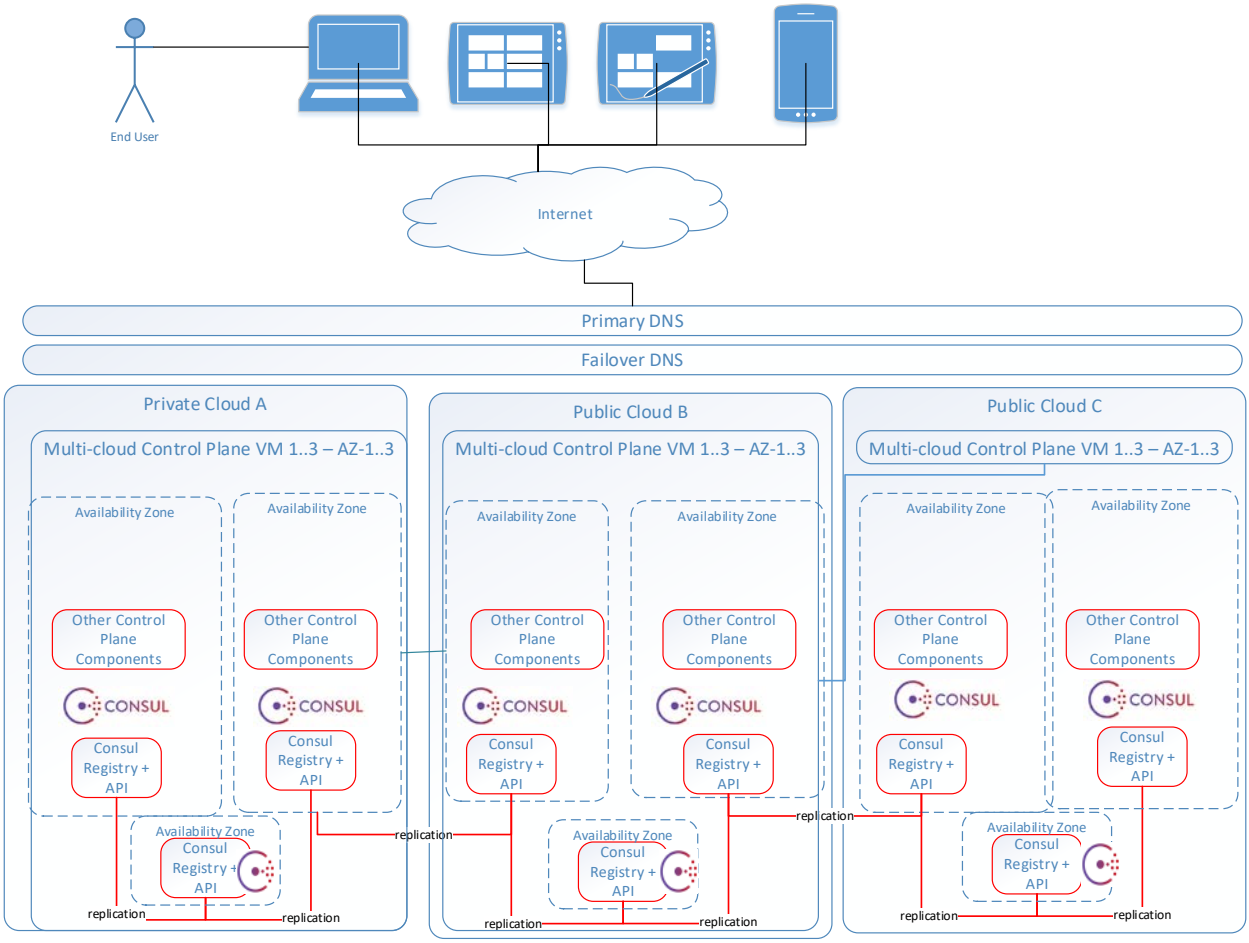


Figure 49 - Multi-cloud Registry and API Implementation Example

Example Deployment Options that are somewhat similar to this pattern with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/service-registry-discovery

- **Known Uses:** Consul http://www.consul.io, Apache Zookeeper https://zookeeper.apache.org
- **Related Patterns:** Multi-Cloud Data Replication

So now that we have deployed everything what do we do when things fail? This is what's sometimes referred to as "Day 2" problems. (“Day 2” as in the day after initial deploy) Ideally we should not need humans involved in dealing with failures after initial deployment.

4.2 Multi-Cloud Management after Initial Deployment and Dealing with Failures in Automated Way

Now that we have our static deployment complete how do we Multi-Cloud Management after Initial Deployment and Dealing with Failures in Automated Way? The key to answer this question is telemetry of our systems. Let's start first by figuring out once we deploy how do we find out the health of the instances?

4.2.1 Multi-Cloud Telemetry and Log Aggregation Pattern

Multi-Cloud Telemetry and Log Aggregation Pattern helps to aggregate all of the telemetry (CPU, Memory, Storage, IO, Network utilization) and key logs from all cloud providers and all of components such Virtual Machines etc. It also aggregates health information of all of the components (if component/system is up/down).

- **Pattern Name and Classification:** Multi-Cloud Telemetry and Log Aggregation Pattern
- **Also Known As:** Multi-cloud SLA Monitoring Pattern
- **Problem:** It is difficult to manage anything that you cannot measure hence we need Multi-cloud SLA Monitoring Pattern that aggregates all of the telemetry data (CPU, Memory, Storage, IO, Network utilization) from all cloud providers as well as Health (up/down) of all of the components.
- **Intent:** Multi-Cloud Telemetry and Log Aggregation Pattern allows to monitor and aggregate multiple cloud provider IaaS data about health and utilization of

components i.e. Virtual Machine CPU, Memory, Storage and Network as well as aggregate of key logs.

- **Motivation (Forces):** Multi-Cloud Telemetry and Log Aggregation Pattern is akin to the central nervous it involves metrics collection agents to be installed on all nodes, process metrics and send aggregate metrics to a central aggregator. Cloud Provider experience outages, but more very frequently clients might experience degradation of quality of service. This could be as simple as Noisy Neighbor problem where you might be sharing a hypervisor (that runs virtual machines) with a neighbor that consumes a great deal of resources. Multi-Cloud Telemetry and Log Aggregation Pattern can be used to monitor the cloud provider performance. It will be more useful in addition to Multi-Cloud Load Balancing Pattern.
- **Applicability:** multi cloud / hybrid deployments
- **Structure:** A graphical representation of the pattern.

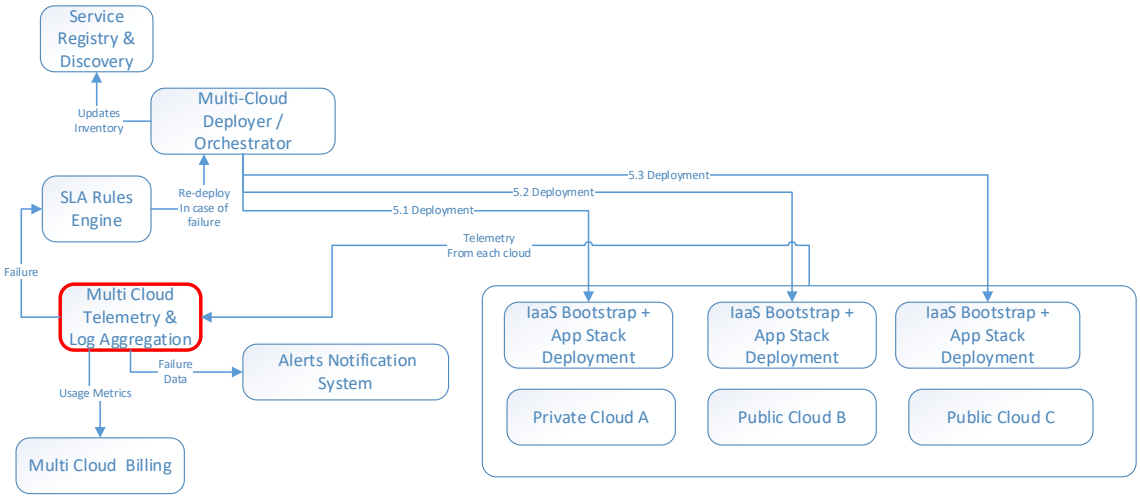


Figure 50 - Multi-Cloud Telemetry and Log Aggregation Pattern

Additionally this pattern can be extended to figure out the latency to a cloud provider via remote distributed probes mimicking the end user experience as depicted here:

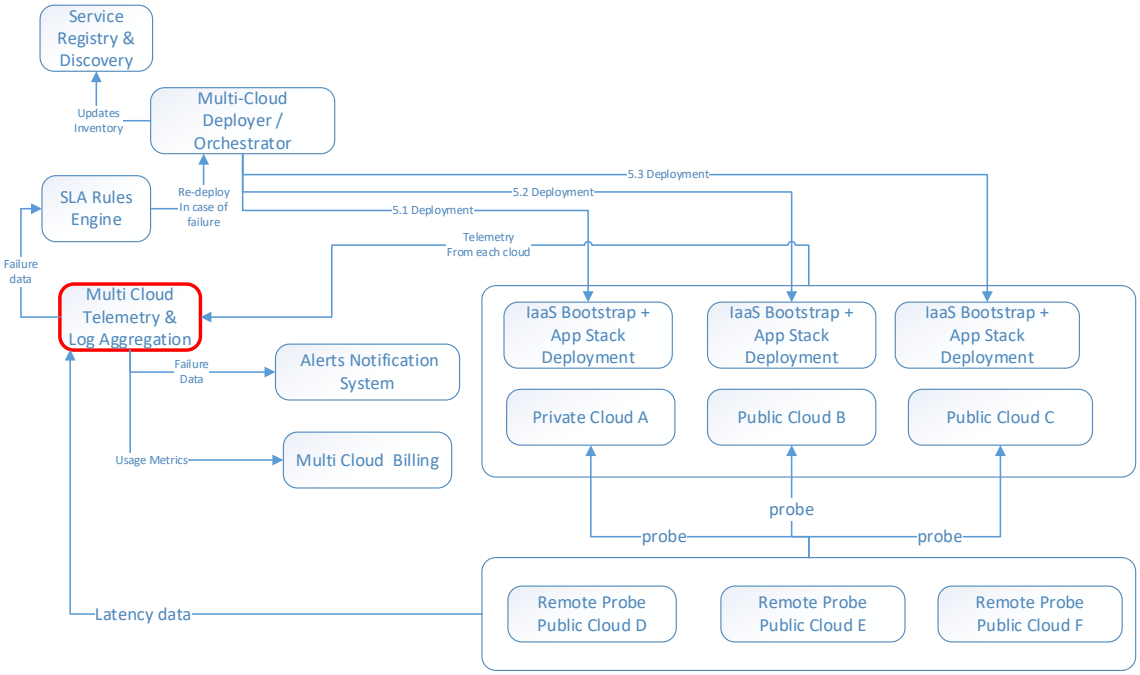


Figure 51 - Multi-Cloud Telemetry and Log Aggregation Pattern with Remote Probes

Remote probes are geographically distributed components (deployed outside cloud provider it is trying to monitory) and used to simulate end user experience with network latency. These components can be as simple as headless browser that runs on schedule doing HTTP get on the web page or service end point. Examples are provided in the implementation section and accompanying github repository.

- **Participants:** each virtual machine has an agent that reports telemetry information as well as forwards logs to centralized system
- **Consequences:** telemetry and logs get aggregated and operator gets to set rules based on incoming data

- **Implementation:**

  Example Deployment Options that are somewhat similar to this pattern with open source software are scripted and provided in the companion github repository for this dissertation:

  https://github.com/compscied/multi-cloud/tree/master/telemetry-multi-cloud

- **Known Uses:** Telemetry open source choices: Icinga https://www.icinga.com, Prometheous http://prometheous.io

  Logging open source choices: EFK Stack: FluentD (http://www.fluentd.org) – log forwarding, Elasticsearch – indexing / Kibana – log mining/visualizations (https://www.elastic.co), Alternatively one can use Apache Solr – indexing (http://lucene.apache.org/solr/) / FluentD / Banana (fork of Kibana)

  Related Patterns: SLA and Health Check Patterns

Next let's take a look at what we can do with the data gathered by this pattern.

4.2.2 SLA Enforcer Rules Engine

So now that we know things failed or something is at high CPU/Memory/IO utilization how do we heal the failure or breach of SLA so operator does not have to even wake up i.e. at 3 a.m.? SLA Enforcer Rules Engine pattern can automatically respond to failures and bring the cloud deployment to the desired healthy state. SLA Monitoring Pattern feeds information to SLA Enforcer Rules Engine to process, apply rules and decide on action to be taken.

Here is an example of a rule for when condition of CPU utilization more than 80% invoke action - create new node:

```
When
   <Condition is true> CPU_Utilization_Percentage > 80
Then
   <Take desired Action> Invoke_Deployer_to_Create_New_Node()
```

- **Pattern Name and Classification:** SLA Enforcer Rules Engine Pattern
- **Also Known As:** Multi-Cloud SLA Rules Processor
- **Problem:** You have deployed X number of components on multiple private/public cloud IaaS and you need to keep resurrect / heal the instances that fail. Additionally, something might not be in failed state, but all of the symptoms are there that something is about to fail or over utilized. For example, CPU or Memory is growing and currently at 80% - in this case we can spin up a new instance to horizontally scale and reduce the load on the current components.
- **Intent:** This pattern usually needs Multi-Cloud Telemetry and Log Aggregation Pattern in order to understand when instance dies to be able to resurrect it.
- **Motivation (Forces):** multi-cloud deployment healing of failed components
- **Applicability:** private/public cloud IaaS multi-cloud deployment
- **Structure:** A graphical representation of the pattern.

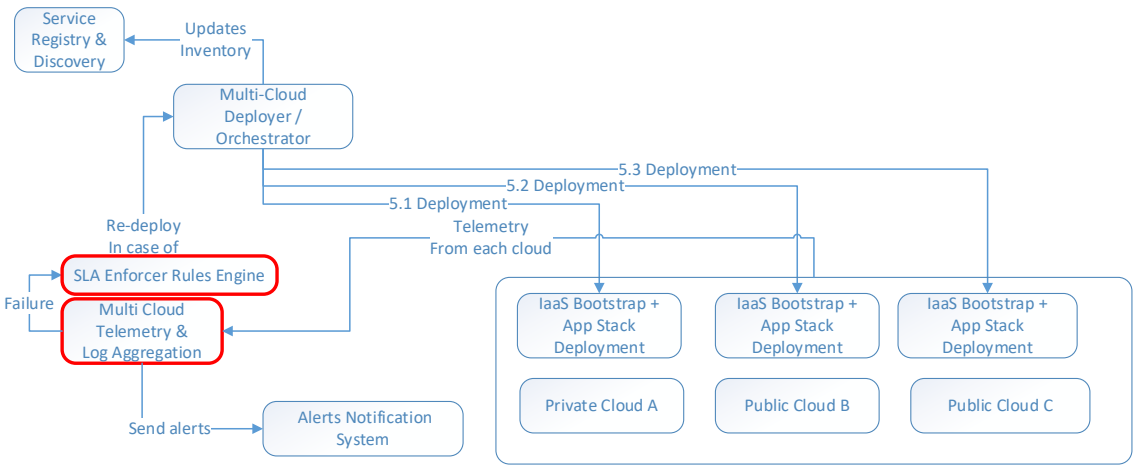


Figure 52 - Multi-Cloud Telemetry and SLA Enforcer Rules Engine

- **Participants:** SLA Enforcer Rules Engine, Telemetry, Multi-Cloud Deployer, Service Registry
- **Consequences:** In case of node failure (health check failed) SLA Enforcer Rules Engine will decide to resurrect another node using the desired virtual machine image to match the lost node. In case of breach of SLA or a particular rule met action is applied.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.

Here is simple algorithm SLA Enforcer Rules Engine implements for keeping the desired state up to date

```
desired_state = deployment_blue_print
current_state = telemetry.get_current_state()
If desired_state != (not equals) current_state
    start_healing()
    for each item in desired_state
            compare/diff(current_state, desired_state)
            // can compare line by line
            get_Healthy_Availability_Zones_List
            resurrect/heal(desired_state.instance,
    get_Healthy_Availability_Zones_List.Item)
```

For example, if we had X (4 instances) virtual machines (desired state) in cloud Y and one of them failed resulting in 3 instances the SLA Enforcer Rules Engine will spin up new 4th instance to bring back the count to the desired state.

Example Deployment Options that are somewhat similar to this pattern with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/cloud-blueprint-bootstrap-options

SLA Rules and Setup Instructions can be found here:

https://github.com/compscied/multi-cloud/tree/master/cloud-healer-sla-rules

Rules:

```
//CPU Utilization higher than 80% - create new node
When
   <Condition is true> CPU_Utilization_Percentage > 80
Then
    <Take desired Action> Invoke_Deployer_to_Create_New_Node()
//RAM Utilization higher than 80% - create new node
When
   <Condition is true> RAM_Utilization_Percentage > 80
Then
    <Take desired Action> Invoke_Deployer_to_Create_New_Node()
```

- **Known Uses:** Bosh (http://bosh.io)
- **Related Patterns:** Health Checking, Multi Cloud Telemetry pattern

Next let's address one of the key problems in distributed systems is how to manage state and data?

4.2.3 Multi-Cloud Data Replication

One of the key problems in distributed systems is how to manage state and data? Specifically, one of the fundamental problems is how to safeguard the data in case database fails and how can we continue operating/processing without any downtime? Multi-Cloud Data Replication is not as simple as it sounds because if we were to replicate across cloud providers there is significant network latency as well as possible network congestion. Hence the solution needs to include data storage solution that can tolerate high latency connections, network congestion as well as fault tolerance in each of the cloud deployments. Let's dive deeper into the pattern.

- **Pattern Name and Classification:** Multi-Cloud Data Replication
- **Also Known As:** Cross Cloud Provider Data and State Replication Pattern
- **Problem:** It is necessary to replicate data and state from one cloud provider to another so that in case of an outage we can continue processing on a different availability zone or cloud provider.
- **Intent:** Replicate data or state across multiple cloud provider.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** multi-cloud, ability to read and continue transaction processing from any cloud provider in any failure scenario
- **Structure:**

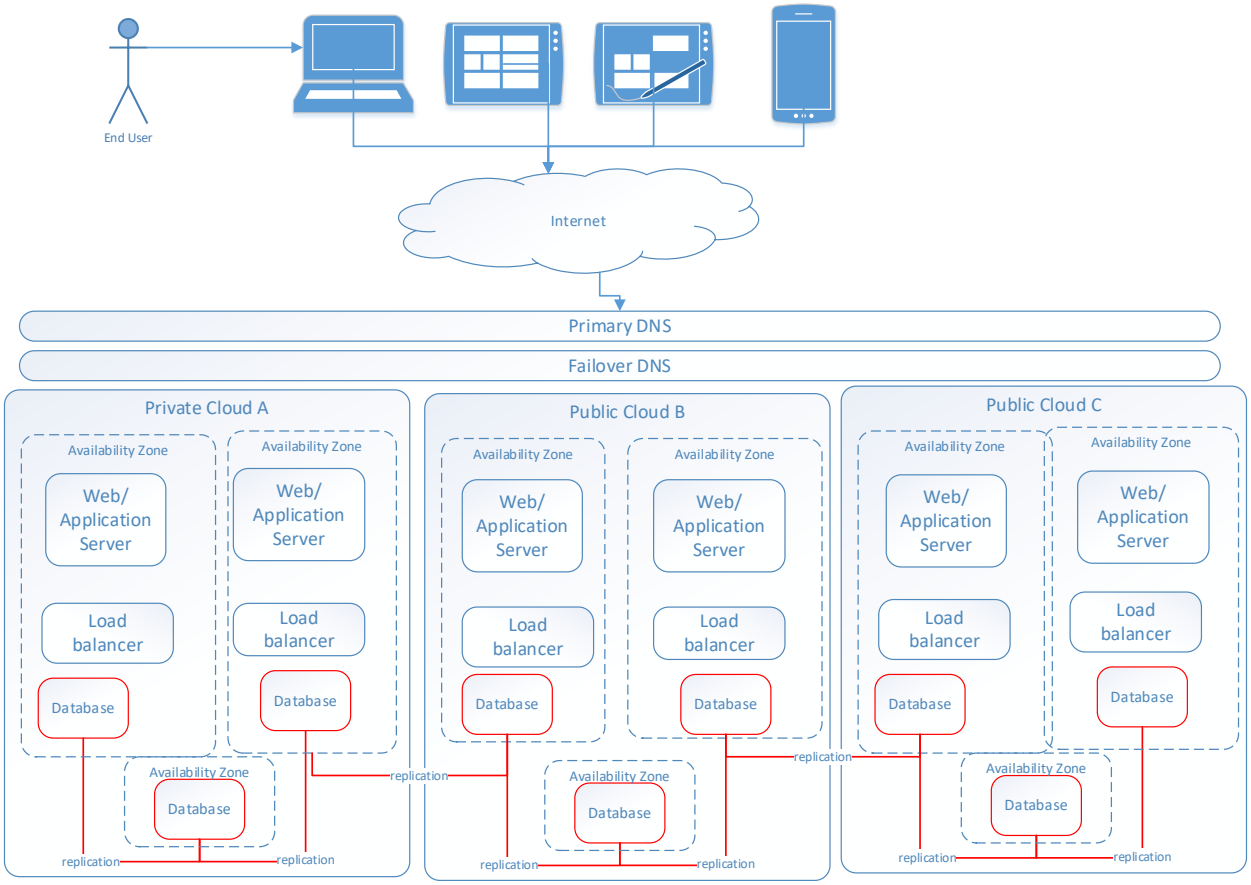


Figure 53 - Multi-Cloud Data Replication Deployment

Each cloud will have a leader that is based on Paxos or Raft consensus algorithm which were described in chapter 2 as foundational components of reliable and resilient distributed systems. If the leader crashes or becomes unresponsive other members will vote and elect a new leader – election of the leader will require at least one more than half of available votes.

- **Participants:** database with ability to replicate with high network latency.
- **Consequences:** data is replicated to all clouds, ability to read and continue transaction processing from any cloud provider in any failure scenario

- **Implementation:** We need database with ability to replicate with high latency network connectivity. Based on tests and benchmarks the following databases fit this requirement.

  https://github.com/compscied/multi-cloud/tree/master/databases-multi-cloud

- **Known Uses:** Apache Cassandra http://cassandra.apache.org/, CockroachDB https://www.cockroachlabs.com, Apache Geode http://geode.apache.org/
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

Next let's explore how do we fail over and manage load distribution?

4.2.4 Reactive Multi-Cloud Health Check and Load Balancing Pattern

You have a number of services that you need to health check all at once for example: Web Server, App Server and Database. The pattern consists of a dynamic web page or API end point on a web server that attempt to connect to app server health check, which in turn has a page to connect to the database. If any of component page's in the chain return a code that is NOT HTTP 200 then the component is down and the whole path should be marked as down.

- **Pattern Name and Classification:** Reactive Multi-Cloud Health check and Load Balancing Pattern
- **Also Known As:** Simple Health check Pattern

- **Problem:** You have a number of services that you need to check all at once for example: Web Server, App Server and Database. The pattern consists of a dynamic web page on a web server that attempt to connect to app server health check, which in turn has a page to connect to the database. If any of component page's in the chain return a code that is NOT HTTP 200 then the component is down and the whole path should be marked as down.

  The pattern is necessary if we have a simple load balancer which can be in front of the chain and if it receives non-HTTP 200 code it will send the load via another path. This pattern does not have to be HTTP centric but could use other protocols that might be built on top of TCP.

- **Intent:** Find out the health of particular path in the deployment
- **Motivation (Forces):** multi-cloud or any distributed deployment
- **Applicability:** multi-cloud, distributed deployment
- **Structure:**

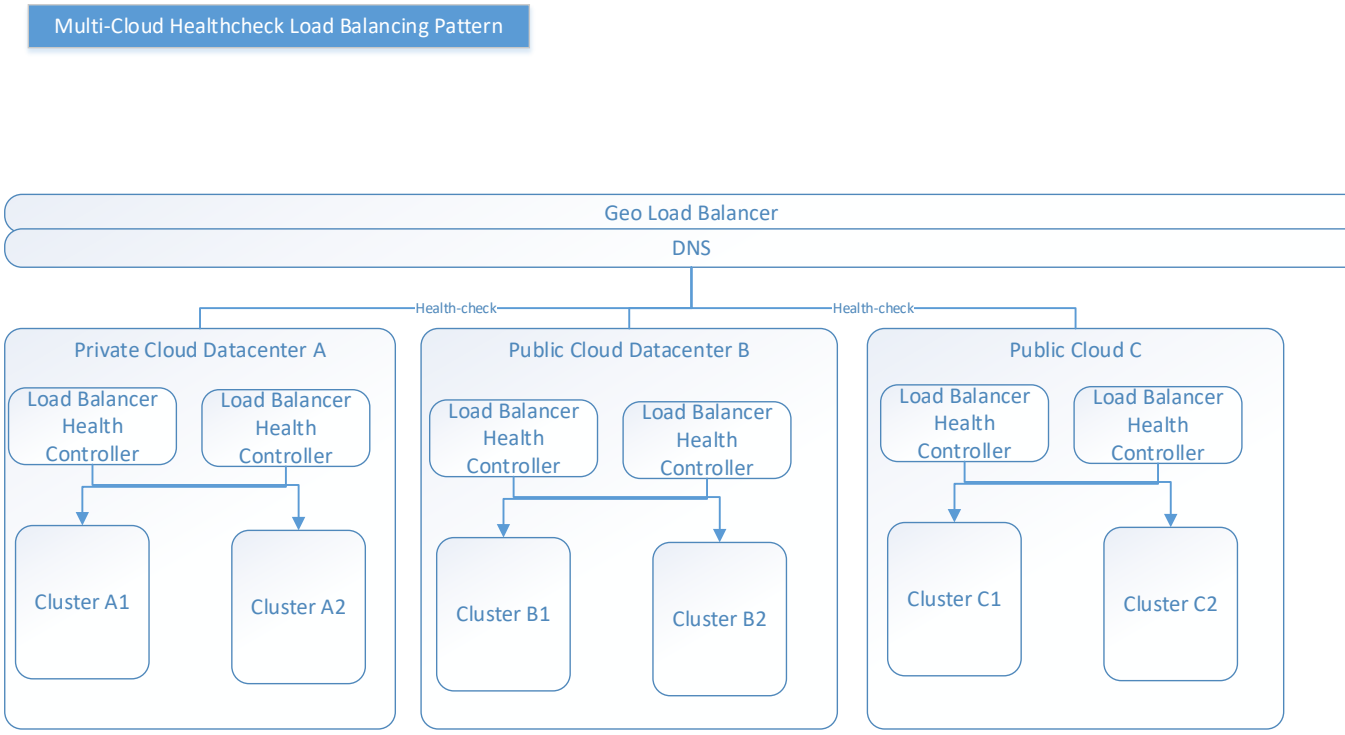


Figure 54 - Reactive Multi-Cloud Health Check and Load Balancing Pattern

- **Participants:** any component that supports http protocol
- **Consequences:** if you get response code which is anything other than HTTP 200 the path is marked as failed
- **Implementation:** https://github.com/keen/pingpong, https://github.com/flok99/httping

4.2.5 Proactive Multi-Cloud SLA Policy Enforcement Pattern

Is there more intelligent way of enforcing Service Level Agreement if we know things are about to fail – before failure occurs?

Proactive Multi-Cloud SLA Policy Enforcement Pattern can help in this case. It builds on top of Multi-Cloud SLA Telemetry and based on set of rules if a cloud provider breaches an SLA we can move workloads to another cloud provider. This pattern allows us to apply rules/triggers against aggregated multi-cloud telemetry data so we can be proactive about managing resources in case of degradation of performance etc.

- **Pattern Name and Classification:** Proactive Multi-Cloud Service Level Agreement Enforcement Pattern
- **Also Known As:** Multi-Cloud SLA Management Pattern
- **Problem:** How do we enforce quality of services across multiple cloud providers? Multi-cloud SLA Monitoring Pattern aggregates all of the telemetry (CPU, Memory, Storage, IO, Network utilization) from all cloud providers. Proactive Multi-Cloud SLA Policy Enforcement Pattern builds on top of Multi-Cloud SLA Telemetry and based on set of rules if a cloud provider breaches an SLA we can move workloads to another cloud provider.
- **Intent:** Apply rules/triggers against aggregated multi-cloud telemetry data so we can be proactive about managing resources in case of degradation of performance etc.
- **Motivation (Forces):** Proactive management of SLA and quality of service in multi-cloud environment
- **Applicability:** multi cloud / hybrid deployments
- **Structure:** A graphical representation of the pattern.

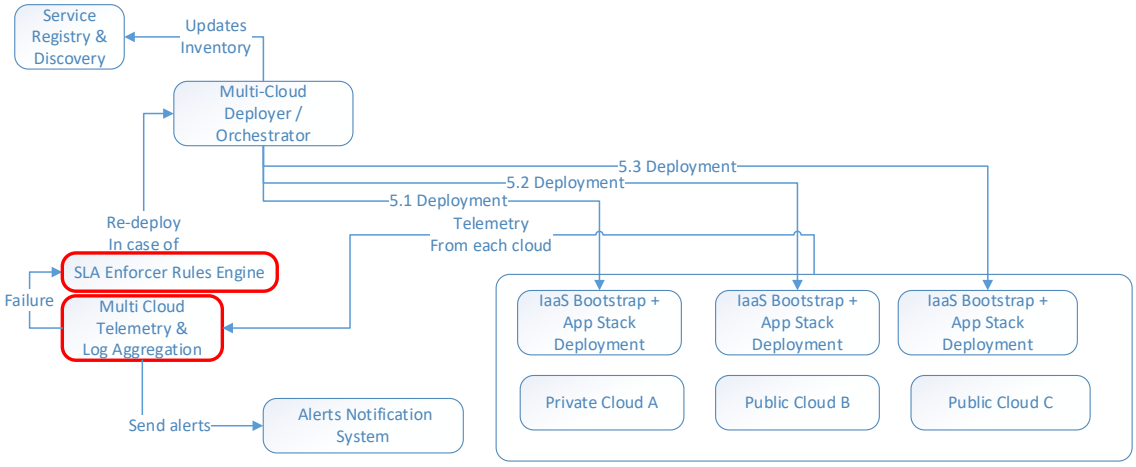


Figure 55 - Proactive Multi-Cloud Health Check and Load Balancing Pattern

Next let's see how this pattern fits within entire Multi-Cloud Framework

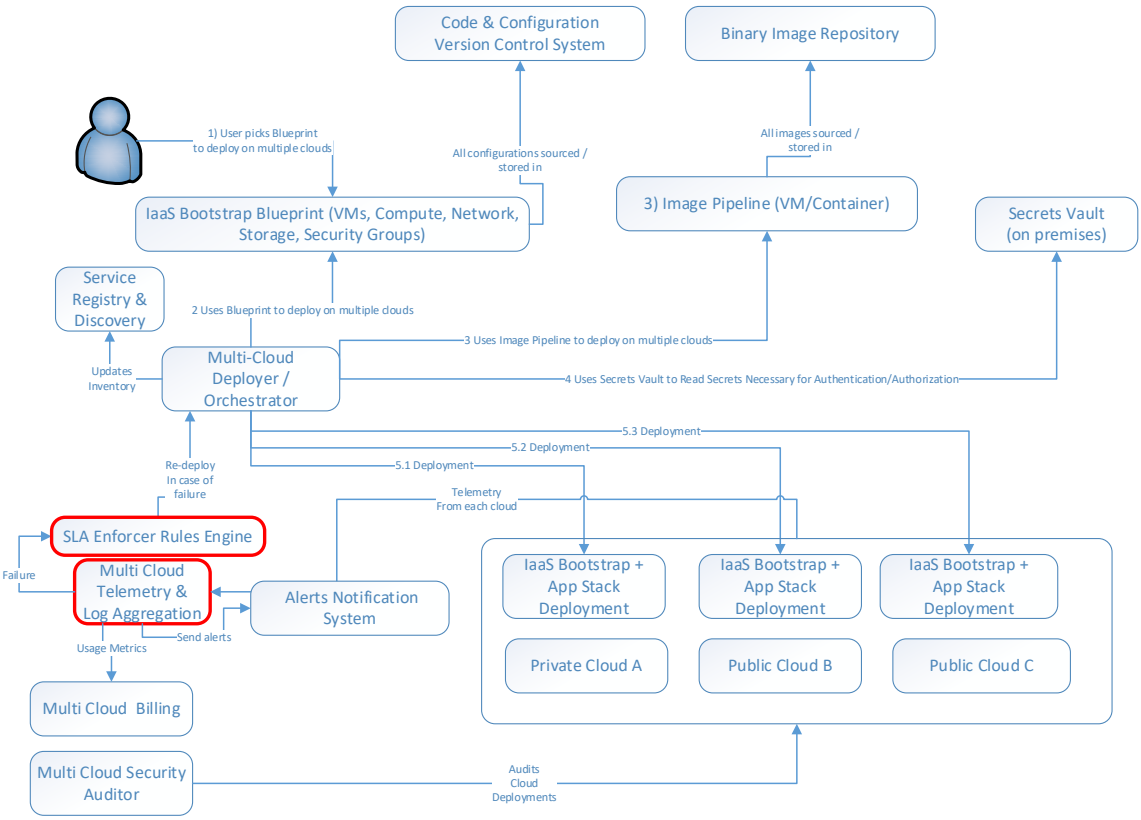


Figure 56 - Proactive Multi-Cloud Health Check and Load Balancing Pattern with Multi-Cloud Control Plane

- **Participants:** each virtual machine has an agent that reports telemetry information as well as forwards logs to centralized system. SLA Enforcer is essentially a rules

engine that processes the aggregated rules and triggers the desired action as an outcome.

- **Consequences:** telemetry and logs get aggregated and operator gets to set rules based on incoming data. Policy Enforcer will evaluate the telemetry data against the rules and trigger an action for SLA Enforcer Rules Engine.
- **Implementation:**

Policy Enforcer will continuously get telemetry information and apply rules for example here is some simple algorithms:

```
If Cloud X Bandwidth > Desired State (90%)
    stop routing to this cloud provider
    direct load to another cloud providers
```

Adaptive Intelligence algorithm can also be used to predict the load based on previous history. This can also be done by pre-scheduling instance to run more instance at pre-defined times.

For example, when stock market opens by 9:30 AM EST spin up 99 compute instances to meet increasing load when trading starts. Ideally we should start this earlier so there is enough time for machines to boot, warm up cache for data etc. so in

this example we will start 33 instances in each cloud provider at 9:15 AM EST so by 9:30 AM everything is routable and ready to go:

```
If Time == 9:15 AM EST
{
    deploy 33 instances Cloud X
    deploy 33 instances Cloud Y
    deploy 33 instances Cloud Z
}
```

Example Deployment Options that are somewhat similar to this pattern with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/sla-rules-engine

There are three primary options illustrated:

- JSON Rules - the simplest option in JavaScript
- Easy Rules - Java – simple to setup and get started with
- Drools - Java - more complicated to setup and use, but has nice interface and integration with Eclipse

- **Known Uses:**

- Commercial: AWS CloudWatch, VM Turbo, ScienceLogic
- Open source: none known at this point

- **Related Patterns:** Multi-Cloud SLA Monitoring, SLA and Health Check Patterns

Next let's explore a pattern that helps to solve a problem when private cloud capacity gets exhausted.

4.2.6 Public Cloud Bursting Pattern

Once you have close to or have exhausted your private cloud capacity and Public Cloud Bursting Pattern allows to "burst" and use public cloud capacity to augment on premises workloads. The entire software infrastructure stack, application or services and other components needs to be ready to be replicated in public cloud and the need. Multi-cloud control plane pattern enables cloud bursting and can be combined with Multi-Cloud SLA Enforcer to automate rules on when to burst to the cloud. Using SLA Enforcer in this pattern would allow to set a rule to state for example that at 80% utilization of private datacenter - start deploying workloads only to public cloud.

- **Pattern Name and Classification:** Public Cloud Bursting Pattern
- **Also Known As:** Multi-Cloud Bursting Pattern.
- **Problem:** In some cases organizations, might approach or exhaust on premises data center capacity and this pattern allows to "burst" and use public cloud capacity to augment on premises workloads.

- **Intent:** Public Cloud Bursting Pattern allows to start new workloads or move existing workloads into the Public Cloud. This pattern is enabled by all of the components in the composite of pattern that we call Multi-Cloud Control plane.
- **Motivation (Forces):** On premises capacity is exhausted and you would like to add capacity from public cloud
- **Applicability:** private / public IaaS deployment
- **Structure:**

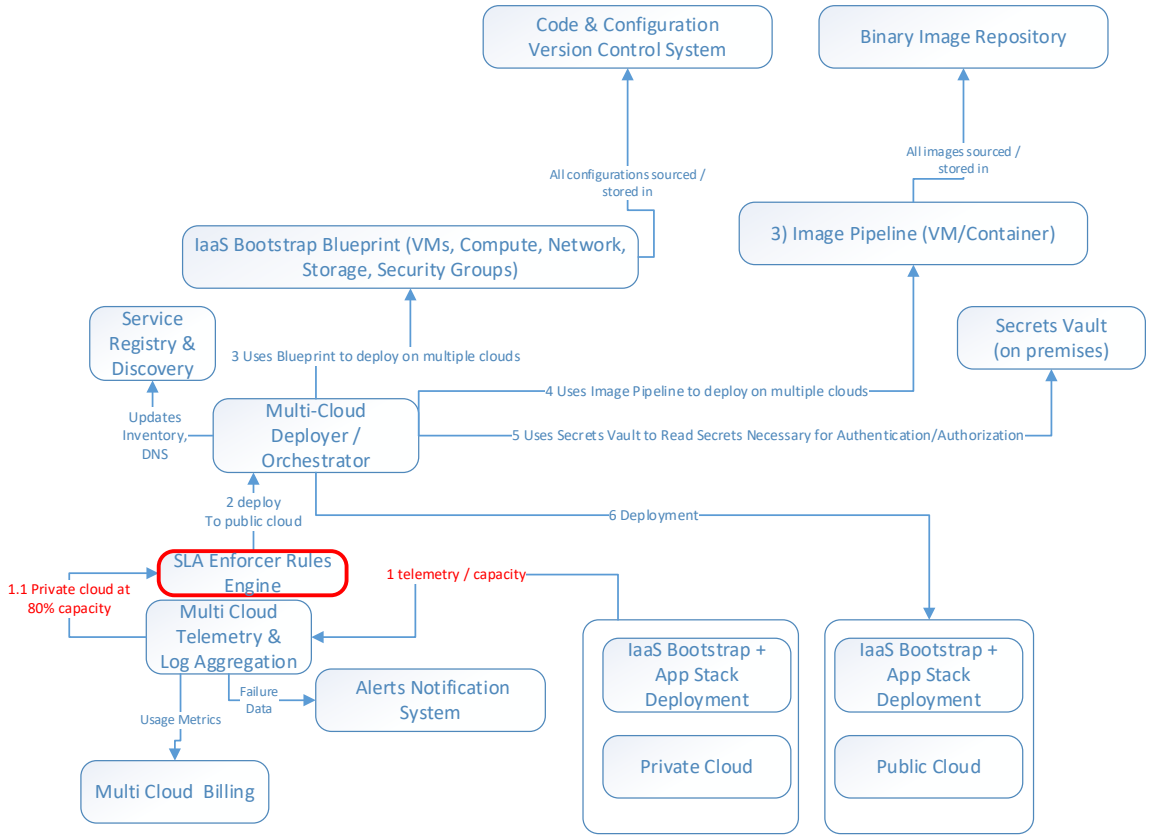


Figure 57 - Public Cloud Bursting Pattern

- **Participants:** Private Cloud, Public Cloud, Multi-Cloud Control Plane, SLA Enforcer
- **Collaboration:**

1) **Multi-Cloud Telemetry** gets and aggregates the capacity data from private cloud and passes it to the **Multi-SLA Enforcer Rules Engine**

2) **SLA Enforcer** determines that capacity is at 80% which meets the alert/trigger to start deployment in public cloud to augment the capacity and invokes public cloud deployment via **Multi-Cloud Deployer**

3) **Multi-Cloud Deployer** pattern uses Blueprint and Base images are built via **Image Pipeline** pattern and stored in binary repository to deploy on public cloud

3) All security sensitive information is sourced from the **Secrets Vault** pattern

4) Multi-Cloud Deployer registers running services in Service Registry and updates any DNS records to add routes to public cloud

- **Consequences:** public cloud deployment has been created and augmenting capacity
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Presents vendor independent logical pattern with specific technology implementation
- **Known Uses:** mostly proprietary solutions
- **Related Patterns:** Using Cloud for Disaster Recovery

Next let's take a look at Multi-Cloud Disaster Recovery Pattern.

4.2.7 Multi-Cloud Disaster Recovery Pattern

Many organizations still run in private data centers for various reasons which might include better security and control from physical location point of view. However, what if private data/center/cloud fails? Can we recover from this disaster using a public cloud?

During an outage in on premises data center cloud can be used to augment the private data center to continue operations. Ideally this would be done in automated fashion with advanced load balancing marking the path to the failed data center as down and automatically routing to one or more cloud provider. Continued replication of data from on premises to the cloud is needed to avoid disruption of service.

- **Pattern Name and Classification:** Using Public Cloud for Disaster Recovery
- **Also Known As:** Cloud DR
- **Problem**: An outage in private decenter requires a failover location that is geographically dispersed.
- **Intent:** Creating disaster recovery in public cloud without the need of creating and maintaining an actual data center.
- **Motivation (Forces):** During an outage in on premises data center cloud can be used to augment the private data center to continue operations. Ideally this would be done in automated fashion with advanced load balancing marking the path to the failed data center as down and automatically routing to one or more cloud provider. Continued replication of data from on premises to the cloud is needed to avoid disruption of service.

- **Applicability:** Situations in which this pattern is usable; the context for the pattern.
- **Structure:**

  This pattern at a minimum requires active data replication from on premises to the cloud

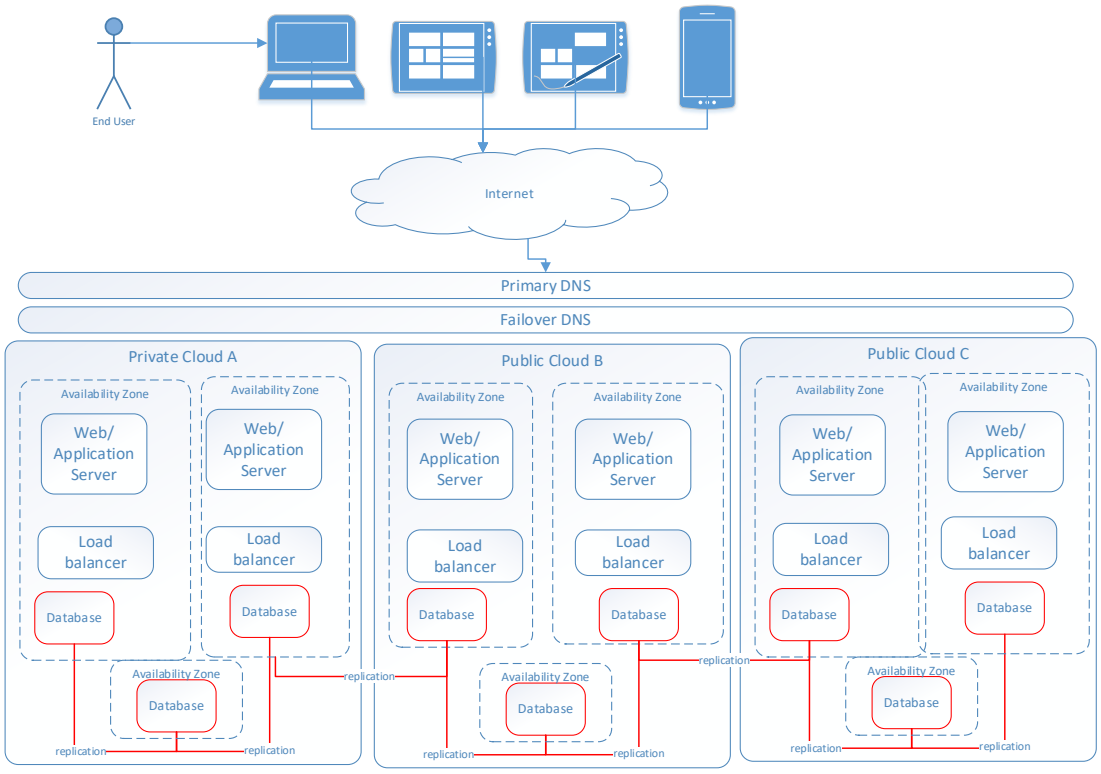


Figure 58 - Multi-Cloud Disaster Recovery Pattern

Public Cloud can be continuously updated with latest deployments using the Multi-Cloud Control Plane Pattern.

Here is the entire flow with Multi-Cloud Control Plane

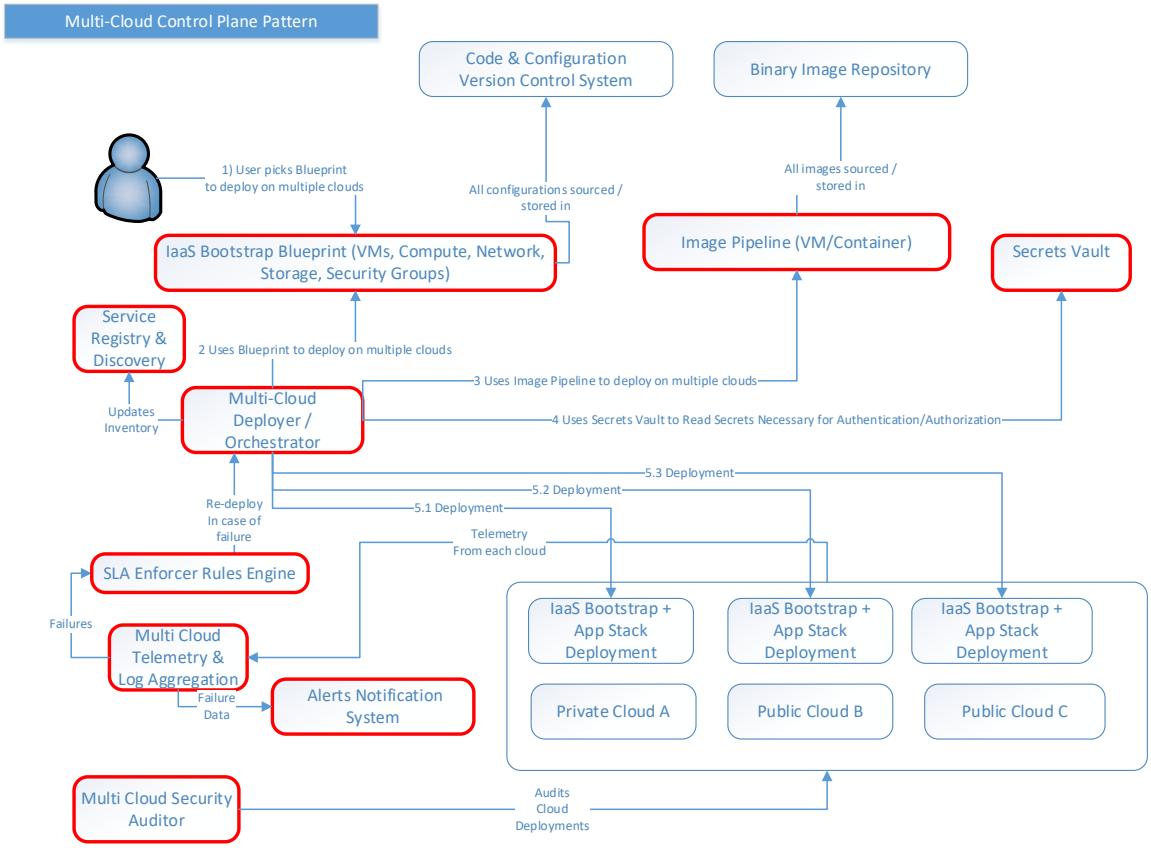


Figure 59 - Multi-Cloud Disaster Recovery Pattern with Multi-Cloud Control Plane

In case of disaster Telemetry notifies SLA Enforcer that it did not receive a response from the private cloud which in turn processing the rule/action for this event and notifies Multi-Cloud Deployer/Orchestrator to start deployment to public cloud or activate existing deployment.

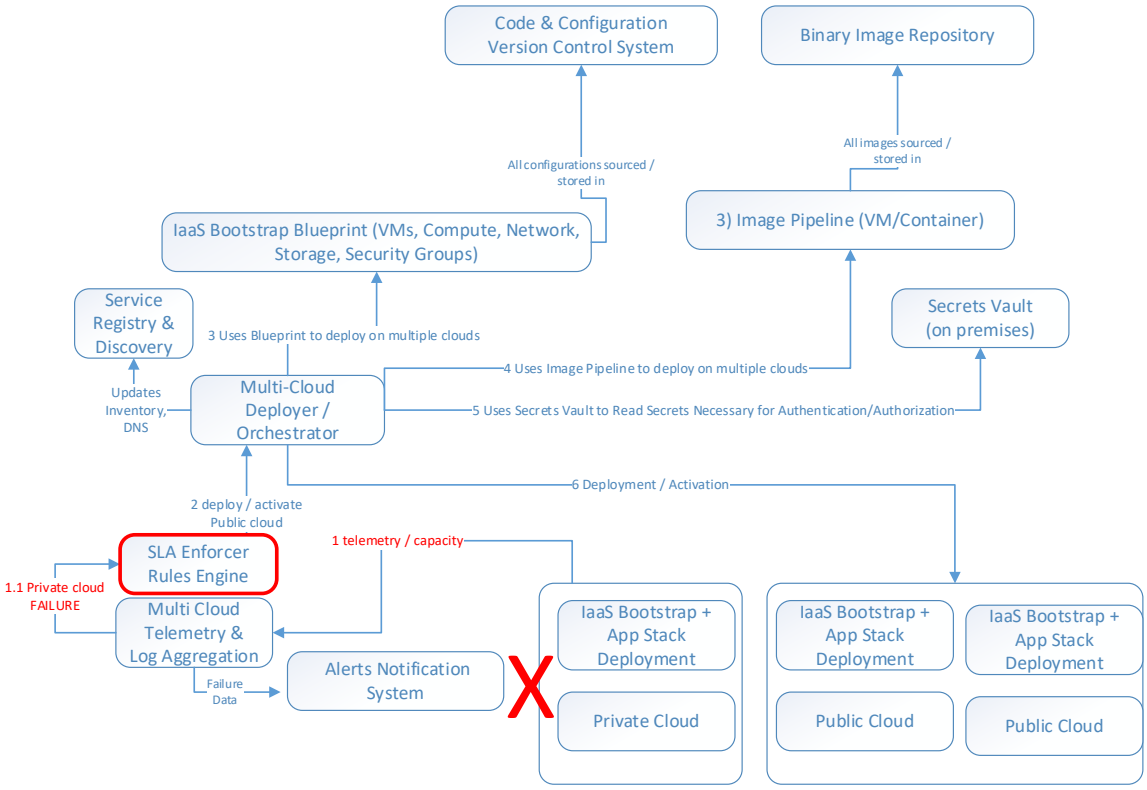


Figure 60 - Multi-Cloud Disaster Recovery Pattern Detailed

- **Participants:** A listing of the Software and Infrastructure Components used in the pattern and their roles in the design.
- **Consequences:** Public Cloud is continuously updated with latest data and applications so if the failover is necessary public cloud is available.

- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Presents vendor independent logical pattern with specific technology implementation
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

4.3 Cost Efficiency Patterns

How do we make sure that cost is not out of control and we know how much we are spending as well as deploy workloads where it is cheaper? Next let's take a look at Multi-Cloud Cost Efficiency Patterns.

First let's take a look at Multi-cloud Aggregate Billing and Chargeback Pattern.

4.3.1 Multi-cloud Aggregate Billing and Chargeback Pattern

How do we aggregate billing and cost from different multi-cloud deployments? In a multi-cloud deployment scenario one needs to keep track and aggregate billing from multiple cloud providers. The key motivations for the pattern are:

- Aggregate billing helps with chargeback to appropriate cost centers
- One might want to make sure that none of the cloud providers exceed billing quotas
- You might want to generate alerts based on cloud utilization

    - It also might make sense to process more using a cheaper cloud provider based on billing metrics

- **Pattern Name and Classification:** Multi-cloud Aggregate Billing and Chargeback Pattern
- **Also Known As:** Other names for the pattern.
- **Problem:** In a multi-cloud deployment scenario one needs to keep track and aggregate billing from multiple cloud providers.
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Motivation (Forces):**
    - Aggregate billing helps with chargeback to appropriate cost centers
    - One might want to make sure that none of the cloud providers exceed billing quotas
    - You might want to generate alerts based on cloud utilization
    - It also might make sense to process more using a cheaper cloud provider based on billing metrics
- **Applicability:** Anytime you need to aggregate utilization for billing and chargeback from more than one cloud
- **Structure:** A graphical representation of the pattern. (Component diagrams and Interaction diagrams may be used for this purpose.)

Multi-Cloud Aggregate Billing gets pricing/billing APIs data from each cloud instance (if available). In addition, Multi-Cloud Telemetry sends usage aggregate

data to supplement the billing data if we can't get it reliably from the cloud provider. Once aggregate cost is calculated – alerts can get send if we breached cost threshold via Alert Notification System.

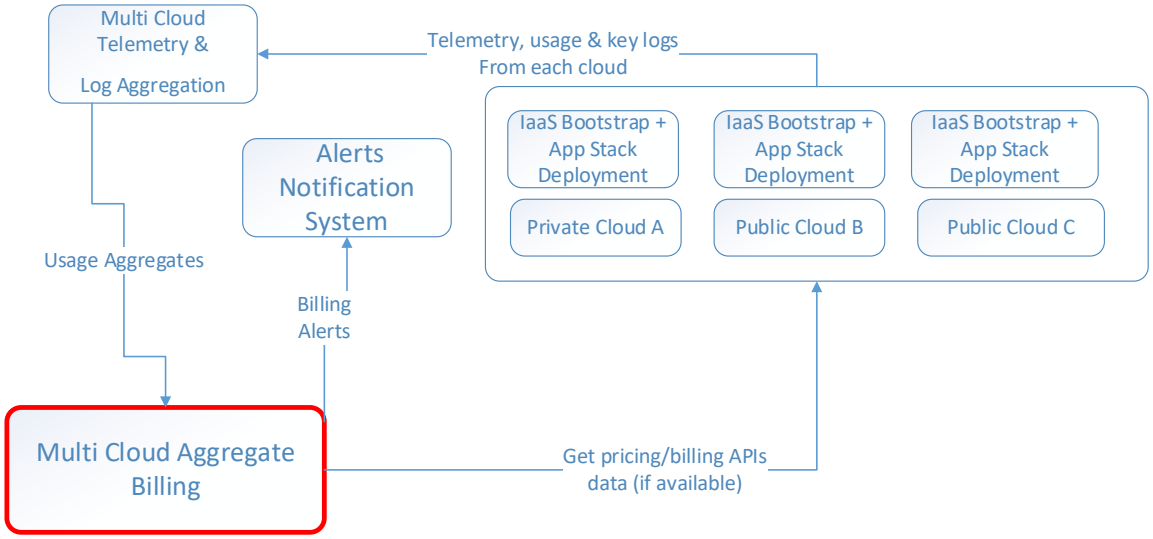


Figure 61 - Multi-cloud Aggregate Billing and Chargeback Pattern

However, better optimization of this pattern would be to use SLA Enforcer Rules Engine to generate billing alerts based on the billing aggregate.

Multi-Cloud Aggregate Billing gets pricing/billing APIs data from each cloud instance (if available). In addition Multi-Cloud Telemetry sends usage aggregate data to supplement the billing data if we can't get it reliably from the cloud provider.

Once aggregate cost is calculated the cost is send to the SLA Enforcer Rules Engine.

SLA Enforcer Rules Engine can send an alert if we breached cost threshold based on Rules set by the operator via Alert Notification System.

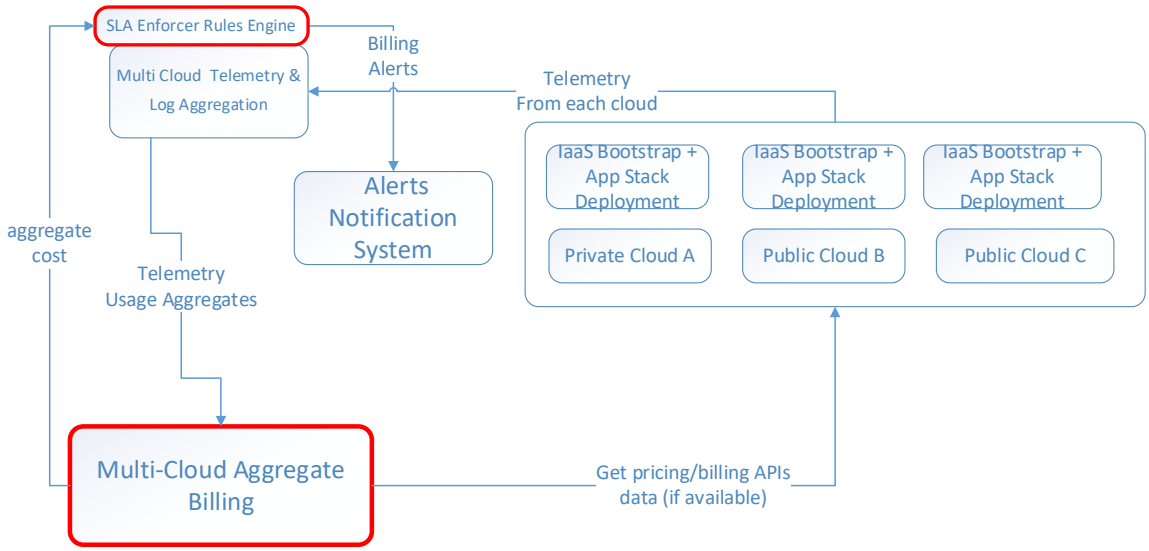


Figure 62 - Multi-cloud Aggregate Billing and Chargeback Pattern with SLA Enforcer Rules Engine

- **Participants:**

  Multi-Cloud Aggregate Billing, Multi-Cloud Telemetry, SLA Enforcer Rules Engine, Alert Notification System

- **Consequences:** we have aggregate cost from multiple cloud instances.

- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.

  Open Source Implementation Options for Multi-Cloud Billing Aggregation are available at the companion repository for this dissertation:

  https://github.com/compscied/multi-cloud/tree/master/multi-cloud-billing

- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

The operating cost might be cheaper in public cloud especially if you look at AWS Spot VM Instances (https://aws.amazon.com/ec2/spot/) or Google's Preemtible VM instances (https://cloud.google.com/compute/docs/instances/preemptible) that are offered at a significant discount to a standard VM pricing. So how do we make sure that cost is not out of control and we know how much we are spending as well as deploy workloads where it is cheaper?

4.3.2 Cost Efficiency Discount Multi-Cloud Pattern

In some cases cloud providers are offering discount virtual machine instances due to idling extra capacity allowing customers to save money. Usually cloud provider allows to bid for spare capacity, however the instances can be taken away by an event notice from cloud provider so the applicable use cases/workloads need to be able to scale on demand and shut down as the cloud provider requests. Therefore, one needs to be able

provision instances in automated/dynamic nature and be able to tear down them down when the cloud provider needs it back gracefully so that we don't have a negative impact to the cloud application workloads. So how do we dynamically take advantage of discount prices without impacting the transactional SLA?

- **Pattern Name and Classification:** Cost Efficiency Discount Multi-Cloud Pattern
- **Also Known As:** Cost Cloud Bursting Pattern
- **Problem:** In some cases cloud providers are offering discount virtual machine instances due to idling or capacity allowing customers to save money. How do we take advantage of discount prices without impacting the transactional SLA?
- **Intent:** save cloud compute operating costs
- **Motivation (Forces):** save money by using discounted instances, reduce operating costs by 50-90% comparatively to regular priced compute, improve application throughput for discounted price
- **Applicability:** usually cloud provider allows to bid for spare capacity, however the instances can be taken away by an event notice from cloud provider so the applicable use cases/workloads need to be able to scale on demand and shut down as the cloud provider requests
- **Structure:**

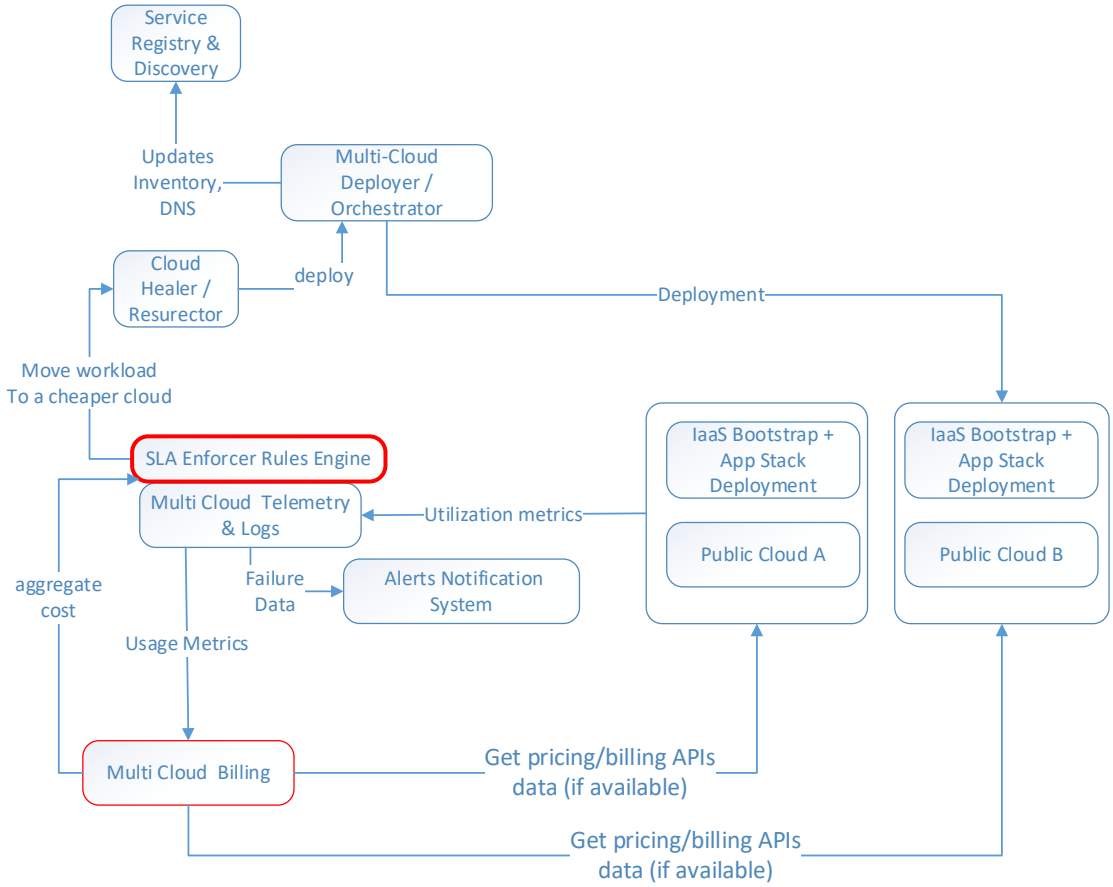


Figure 63 - Cost Efficiency Discount Multi-Cloud Pattern

- Multi Cloud Billing gets lower change in price from cloud provider
- Multi Cloud Billing notifies SLA Enforcer Rules Engine
- SLA Enforcer Rules Engine evaluates the data and find the rule match to Move workload to a cheaper cloud
- SLA Enforcer Rules Engine triggers action for SLA Enforcer Rules Engine Move workload to a cheaper cloud
- SLA Enforcer Rules Engine starts deployment move workloads to a cheaper cloud

- **Participants:** all of the components described in the multi-cloud control plane pattern

- **Consequences:**

Reduction in operating costs by 50-90% comparatively to regular priced compute. However, when the cloud provider issues notification to reclaim the instance the multi-cloud control plane needs to be able to react to it and gracefully migrate workloads etc.

Considerations and Limitations:

- Limitation is usually based on cloud providers reserving the right to shut down the instance at any point. Some cloud providers give advance notice but some do not.

- Application or service running on these instances need to be able to shutdown at any moment with notification from cloud provider implementing an interface for shutdown notification.

- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Presents vendor independent logical pattern with specific technology implementation.

  This pattern can be implemented as extension of Multi-Cloud SLA Enforcement pattern with SLA Rules Targeting Cost variables and thresholds.

  Open Source Implementation Options for this pattern are available at the companion repository for this dissertation:

  https://github.com/compscied/multi-cloud/tree/master/multi-cloud-billing

  https://github.com/compscied/multi-cloud/tree/master/cloud-healer-sla-rules

- **Known Uses:**

  Some examples of these are:

Spot instances in AWS:

https://aws.amazon.com/ec2/spot/

and Preemptible instances in Google Cloud Platform:

https://cloud.google.com/compute/docs/instances/preemptible#preemption_process

The primary difference between AWS and Google discount instances is that in AWS you have to participate in auction to get the instances. In case of Google you get preemptive instances at a fixed discount price.

- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

4.4 Security Related Patterns

Every cloud solution needs to factor in security. Next let's take a look at the minimum we need to provide security patterns that are generally not provided by many cloud providers.

First let's begin with where do we store and retrieve secrets?

4.4.1 Multi-Cloud Secret Storage and Retrieval – Secrets Vault Pattern

Every cloud deployment needs secure secrets storage involves user names, passwords, certificates and any other sensitive information that needs to be protected. At some point create/update/delete and even read operations for cloud APIs will require authentication and authorization. How do we this securely in multi-cloud environment? More importantly we need to make sure this pattern works for all cloud providers.

- **Pattern Name and Classification:** Multi-Cloud Secrets Vault Pattern
- **Also Known As:** Secret Storage and Retrieval
- **Problem:** Every cloud deployment needs secure secrets storage involves user names, passwords, certificates and any other sensitive information that needs to be protected.
- **Intent:** Secure storage and retrieval of secrets
- **Motivation (Forces):** At some point create/update/delete and even read operations for cloud APIs will require authentication and authorization. How do we this securely in multi cloud environment?
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.

- **Structure:** A graphical representation of the pattern.

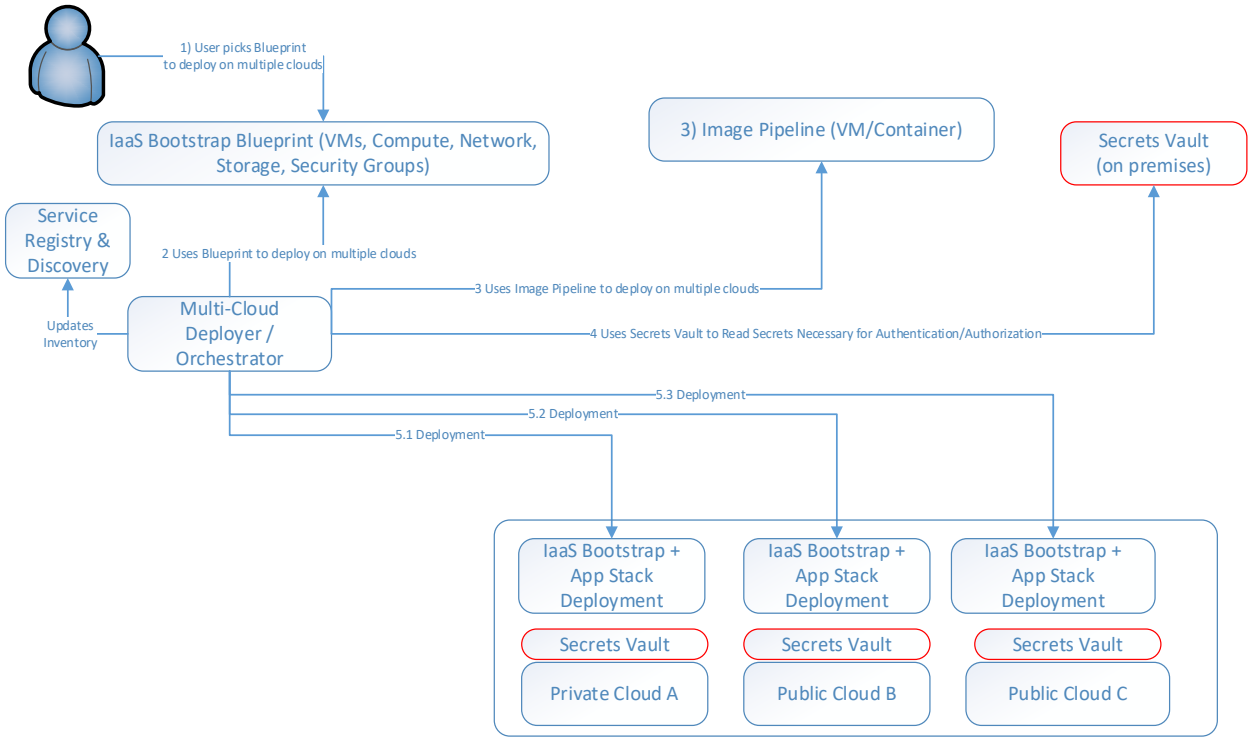


Figure 64 - Multi-Cloud Secret Storage and Retrieval – Secrets Vault Pattern

- **Participants:** Secrets Vault; can be used with Hardware HSM (Hardware Secure Module if offered by cloud provider – example: https://aws.amazon.com/cloudhsm/)
- **Consequences:** Ability to securely retrieve secrets i.e. passwords, certificates.
- **Implementation:** The pattern permits secure access via API to secrets and can be deployed on premises as well as with every cloud deployment. API is necessary for automation and interaction in the orchestration flow of multi-cloud control plane framework. The open source implementation options are provided in this repository:
- https://github.com/compscied/multi-cloud/tree/master/secret-management
- **Known Uses:** Open Source: HashiCorp Vault https://www.vaultproject.io
- **Related Patterns:** Other patterns that have some relationship with the pattern;

Next let's cover how we can assure that our deployment has not be tempered with by malicious actors and patched frequently if there is a vulnerability.

4.4.2 Multi-Cloud Auditor Pattern

How do we ensure that deployment does not have any known vulnerabilities and gets patched if there is a vulnerability? How do we continuously audit the cloud and make sure there are no unauthorized or unintended changes?

- **Pattern Name and Classification:** Multi-Cloud Auditor Pattern
- **Also Known As:** Multi-Cloud Security Policy Auditor Pattern
- **Problem:** How do we ensure that all of our cloud deployments are secure and have not been tempered with? Multi-Cloud Security Policy Auditor Pattern helps us to assure that our multi-cloud deployment has not be tempered with by malicious actors and patched frequently if there is a vulnerability.
- **Intent:** Make sure every multi-cloud deployment is secure and without known vulnerabilities as well as it has not been tampered by a malicious actor.
- **Motivation (Forces):** Motivation for this pattern is to be able to patch instances from known CVEs (vulnerabilities) and protect from tempering by malicious users. Multi-Cloud Security Policy Auditor Pattern helps us to assure that our multi-cloud deployment has not be tempered with by malicious actors and patched frequently if there is a vulnerability.

- **Applicability:** Multi-cloud public and private deployment
- **Structure:** A graphical representation of the pattern.

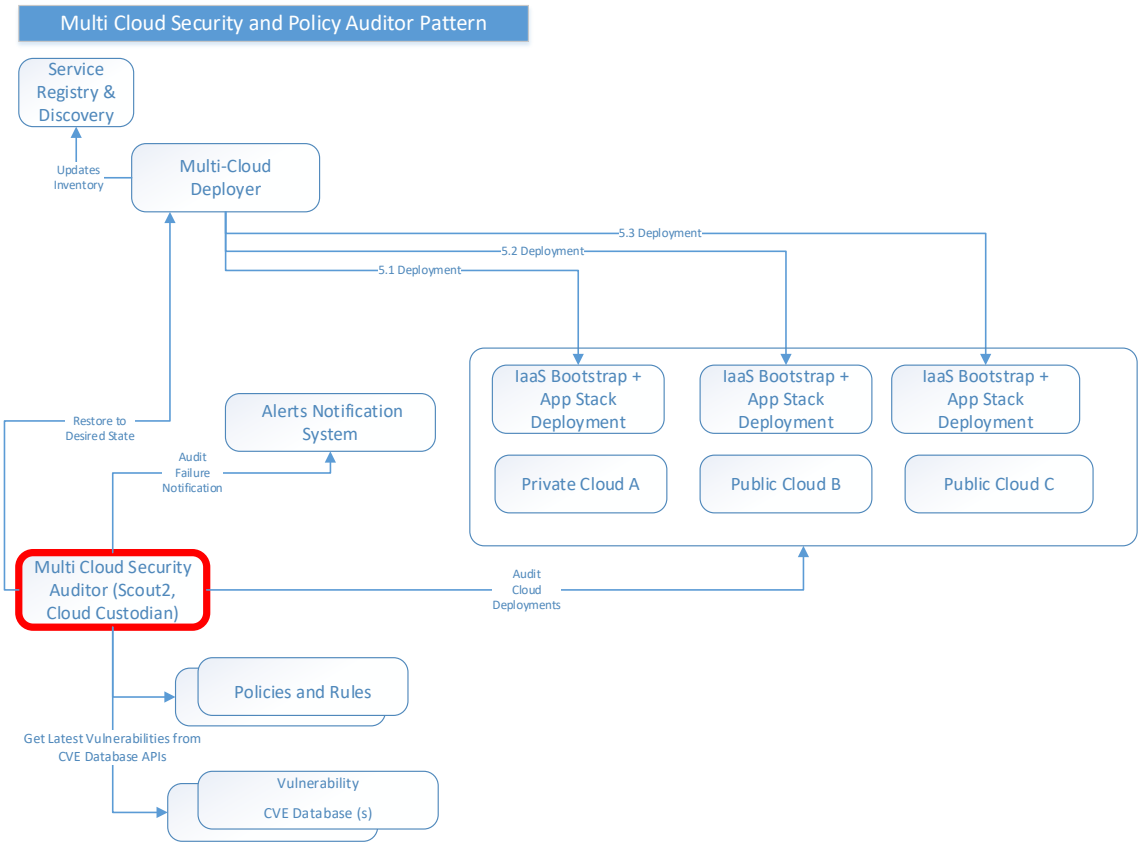


Figure 65 - Multi-Cloud Auditor Pattern

- **Participants:** A listing of the Software and Infrastructure Components used in the pattern and their roles in the design.
- **Collaboration:** Cloud Auditor continuously runs and checks all of the setting and configurations on all clouds and validates against last known secure configuration manifests as well as checking for possible vulnerabilities that might have occurred

due to a new deployment / application introduced into the cloud. After that findings are logged and alerts are sent out if serious issue have been found.

- **Consequences:** A description of the results, side effects, and trade-offs caused by using the pattern.
- **Implementation:**

Cloud Auditor Algorithm Flow:

```
Repeat in a loop triggered by continuous scheduler

 For each cloud in cloud list

        Pick cloud X

         Validate deployment against latest cloud blueprint – raise alert if issue was found

         Scan for policy rule compliance

         Scan for known configuration vulnerabilities – raise alert if issue was found

         Scan and keep track of software version and compare against known list of
         vulnerable software components – raise alert if issue was found

End loop
```

- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

4.5 Multi-Cloud Control Plane Framework

Now let's re-cap what we have learned so far in one picture depicting Multi-Cloud Control Plane Framework:

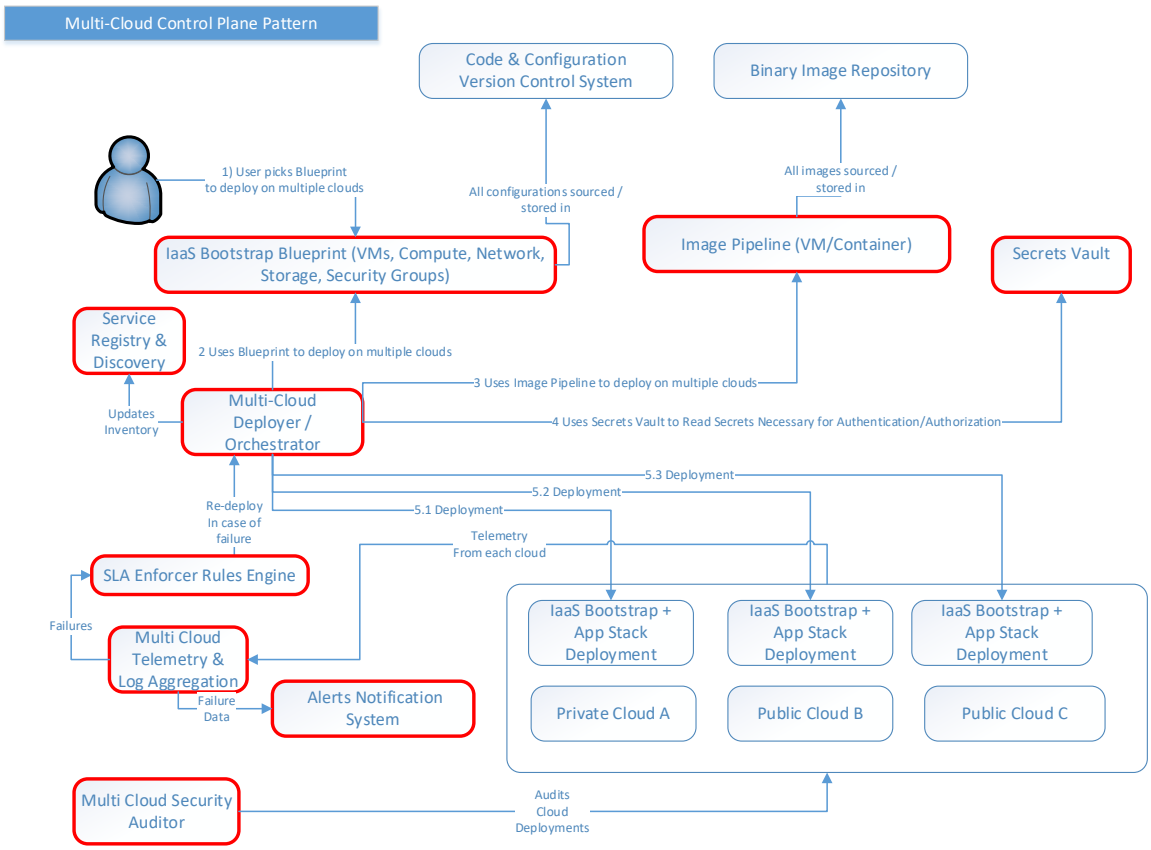


Figure 66 - Multi-Cloud Control Plane Framework

Operator wants to create a deployment consisting of X Virtual machines, Network, Storage and Security Groups

1) Operator (System Admin) picks **IaaS Blueprint** pattern to deploy on multiple clouds which is sourced from Code & Configuration Version Control System

IaaS Bootstrap Blueprint includes details and topology of Virtual Machines (with base Image to be used), Network, Storage and Security Groups

2) **Multi-Cloud Deployer** pattern uses Blueprint and Base images are built via **Image Pipeline** pattern and stored in binary repository to deploy on multiple clouds (Private Cloud A, Public Cloud B and Public Cloud C)

All security sensitive information is sourced from the **Secrets Vault** pattern

Multi-Cloud Deployer registers running services in Service Registry

Dealing with events after initial deployment

*Dealing with **Failures***

We need to deal with failures which is done by **SLA Enforcer Rules Engine** pattern which uses data provided by the **Multi-Cloud Telemetry** system pattern: when a component fails, alert/event is generated and forwarded to **Telemetry System** which **SLA Enforcer Rules Engine** listens to and resurrects the failed component.

*Dealing with **Security***

Cloud Auditor – continuously runs reports against each cloud deployment comparing it against baseline blueprint

If new component is detected, cloud auditor will send alerts and notifications, which in turn get picked up by **SLA Enforcer Rules Engine** which will instruct the Multi-Cloud Deployer to delete the new component that was not in the original blueprint.

Dealing with **Cost**

Multi-Cloud Billing Pattern aggregates usage cost and billing from multiple cloud providers and allows to make better decision about where it would be more cost effective to run.

Here is the mapping of open source software options and cloud frameworks

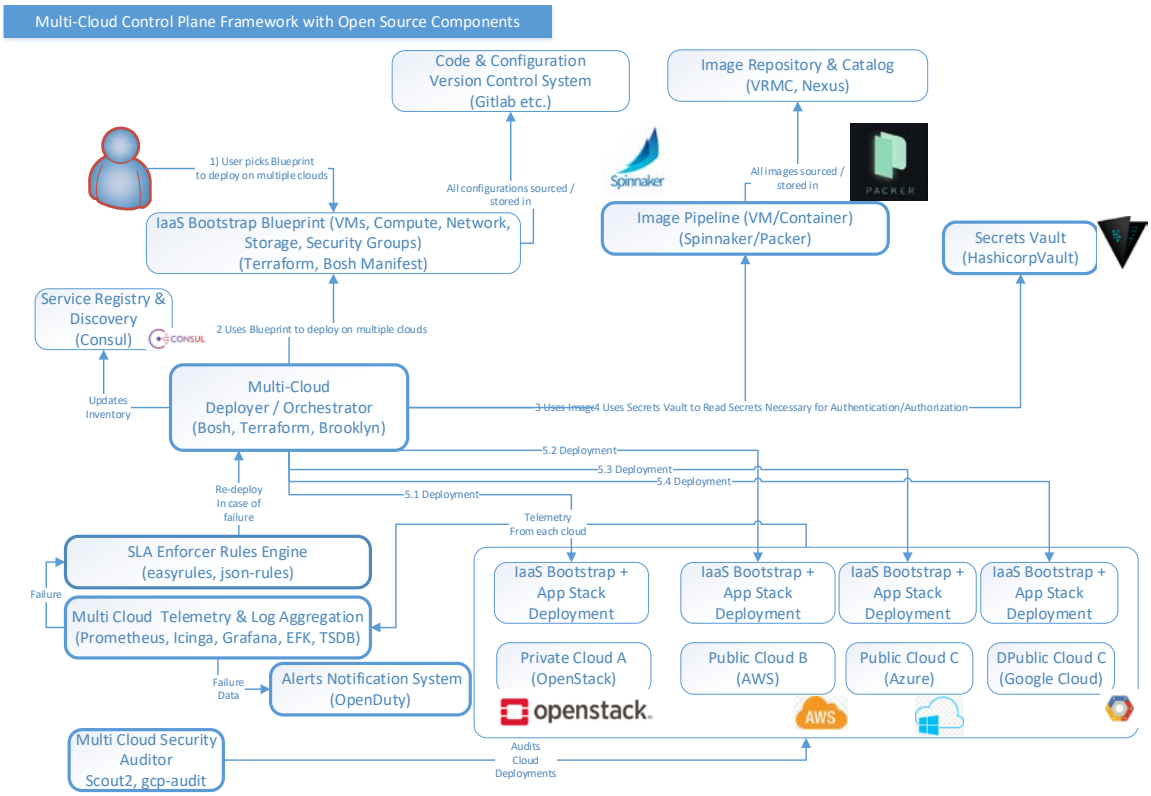


Figure 67 - Multi-Cloud Control Plane Framework Implemented with Open Source Components

4.6 Combining Control Plane Framework with Additional Application Use Case Patterns

So far we have covered patterns and framework, but we have not shown how the multi-cloud framework can be leveraged to deploy real world applications. Next let's go over illustrations of specific application use case deployments where the multi-cloud patterns help.

4.6.1 Multi-Cloud Web Application / Service Pattern

Multi-Cloud Web Application / Service Pattern helps to deploy web application or web services in multi-cloud environment and keep healing the failed components. This pattern will deploy general components you expect in most web applications in fault tolerant manner. In addition all of the components will be monitored and healed by Multi-cloud control plane without for any involvement of a human operator.

- **Pattern Name and Classification:** Multi-Cloud Web Application / Web Service Pattern
- **Also Known As:** Other names for the pattern.
- **Problem:** How do we deploy web application or web services in multi-cloud environment and keep healing the failed components. All of the components you see below are deployed, monitored and healed by Multi-cloud control plane without for any involvement of a human operator.
- **Intent:** Multi-cloud deployment

- **Motivation (Forces):** Higher resilience, fault tolerance in multi-cloud deployment
- **Applicability:** Multi-cloud or hybrid deployments
- **Structure:** A graphical representation of the pattern.

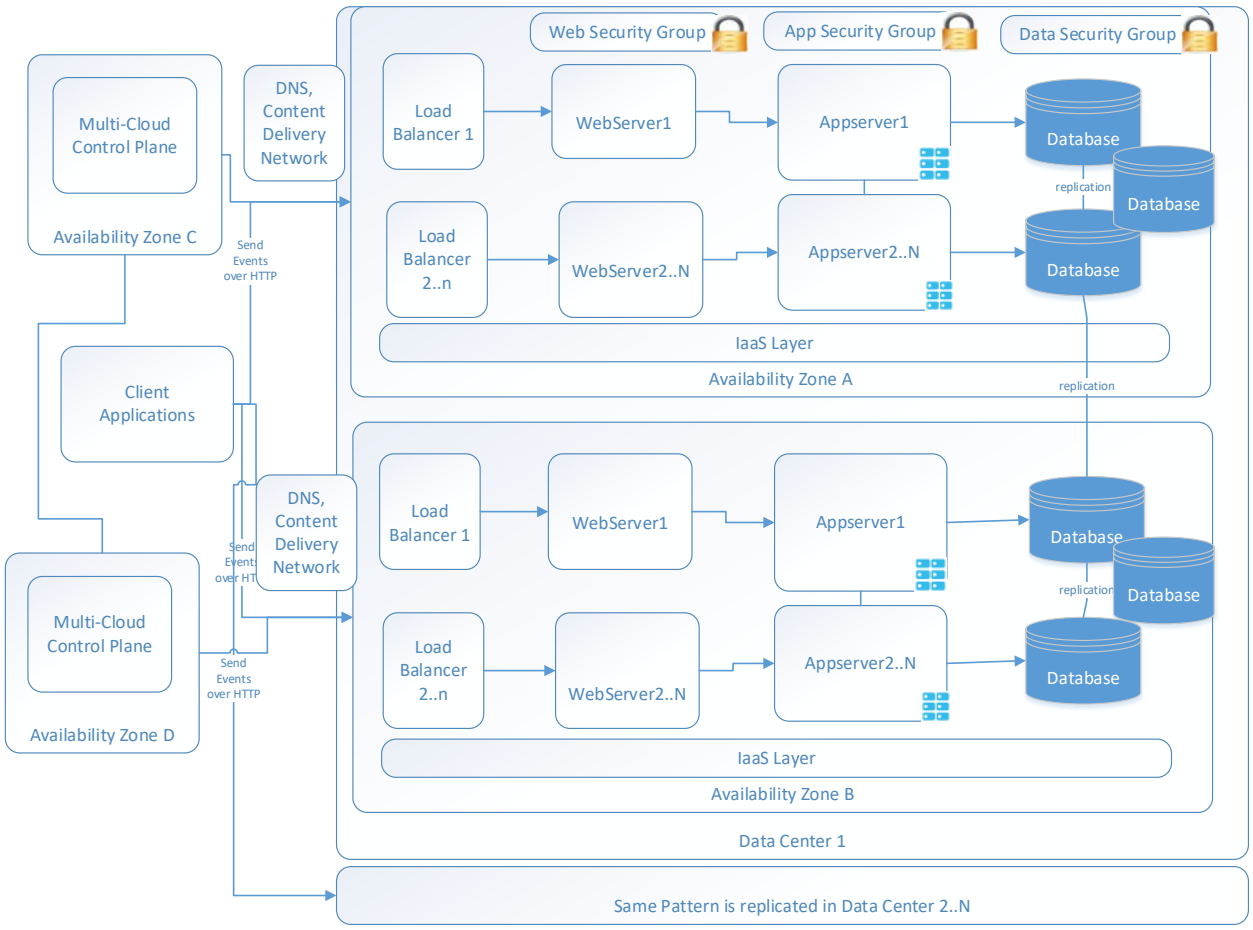


Figure 68 - Conceptual Multi-Cloud Web Application / Service Pattern

- **Participants:** Multi-cloud control plane, Load balancers, web servers, application servers, databases, multi-cloud control plane
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Presents vendor independent logical pattern with specific technology implementation

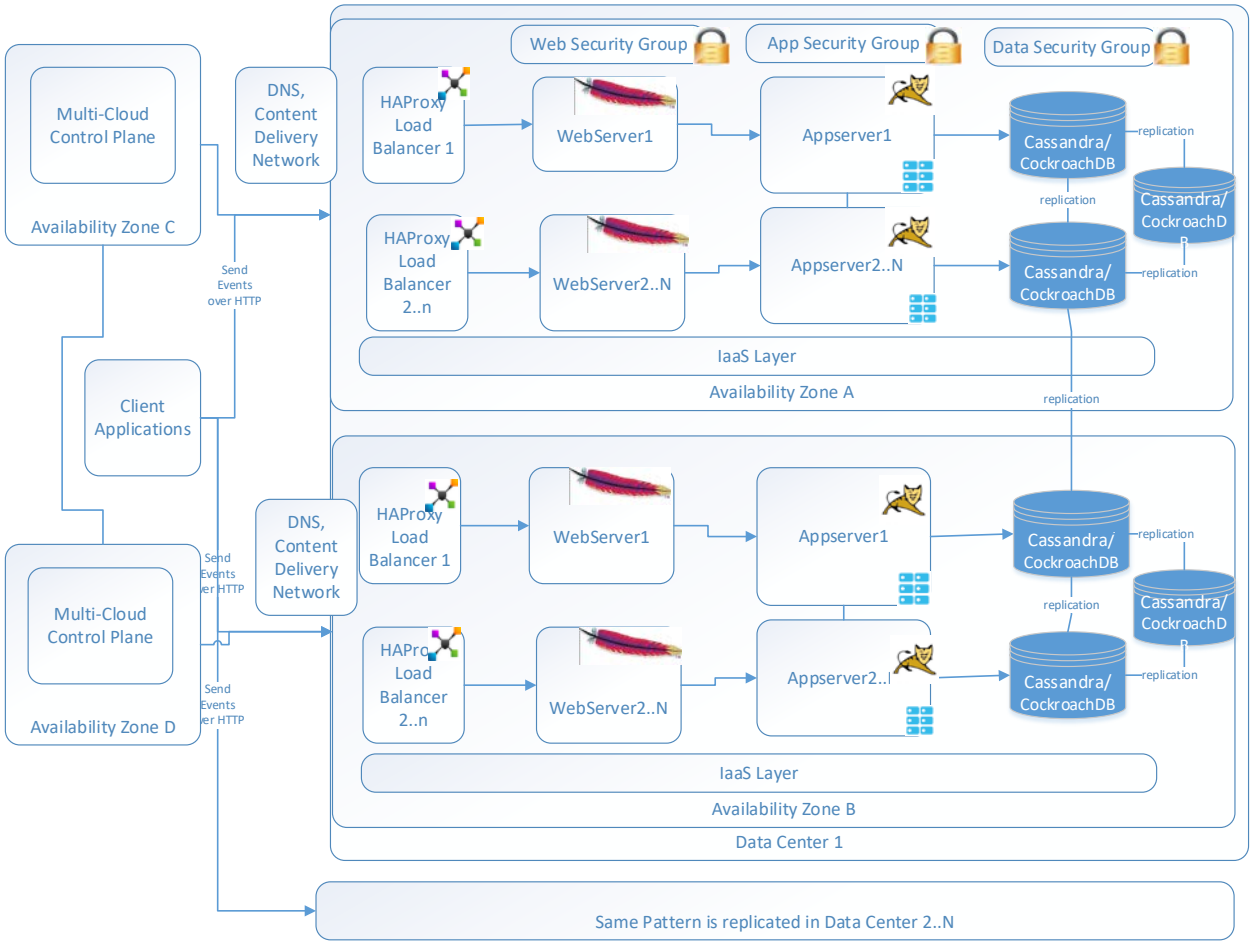


Figure 69 - Multi-Cloud Web Application / Service Pattern Implementation

- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

Next let's see how everything we covered would help us with a more complicated High Volume Distributed Data Ingestion and Processing use case such as with Internet of Things use case

4.6.2 Multi-Cloud Internet of Things Event Stream Ingesting Pattern and Big Data Pipelines

- **Pattern Name and Classification:** Multi-Cloud Internet of Things Event Ingesting Pattern
- **Also Known As:** High Volume Distributed Data Injection and Processing
- **Problem:** Internet of Things (IoT) can be generally defined as any device or object around us that are connected via Internet network this can be sensors, vehicles, buildings etc. All of these devices are reporting extremely high volume of data events or streams of information. How do we process it in the way that we can scale easily using multi-cloud environment?
- **Intent:** deploy in multi-cloud environment
- **Motivation (Forces):** ability to scale beyond one cloud provider
- **Applicability:** High data volume, high rate of concurrent connections from 100 of millions – billions of devices
- **Structure:** A graphical representation of the pattern.

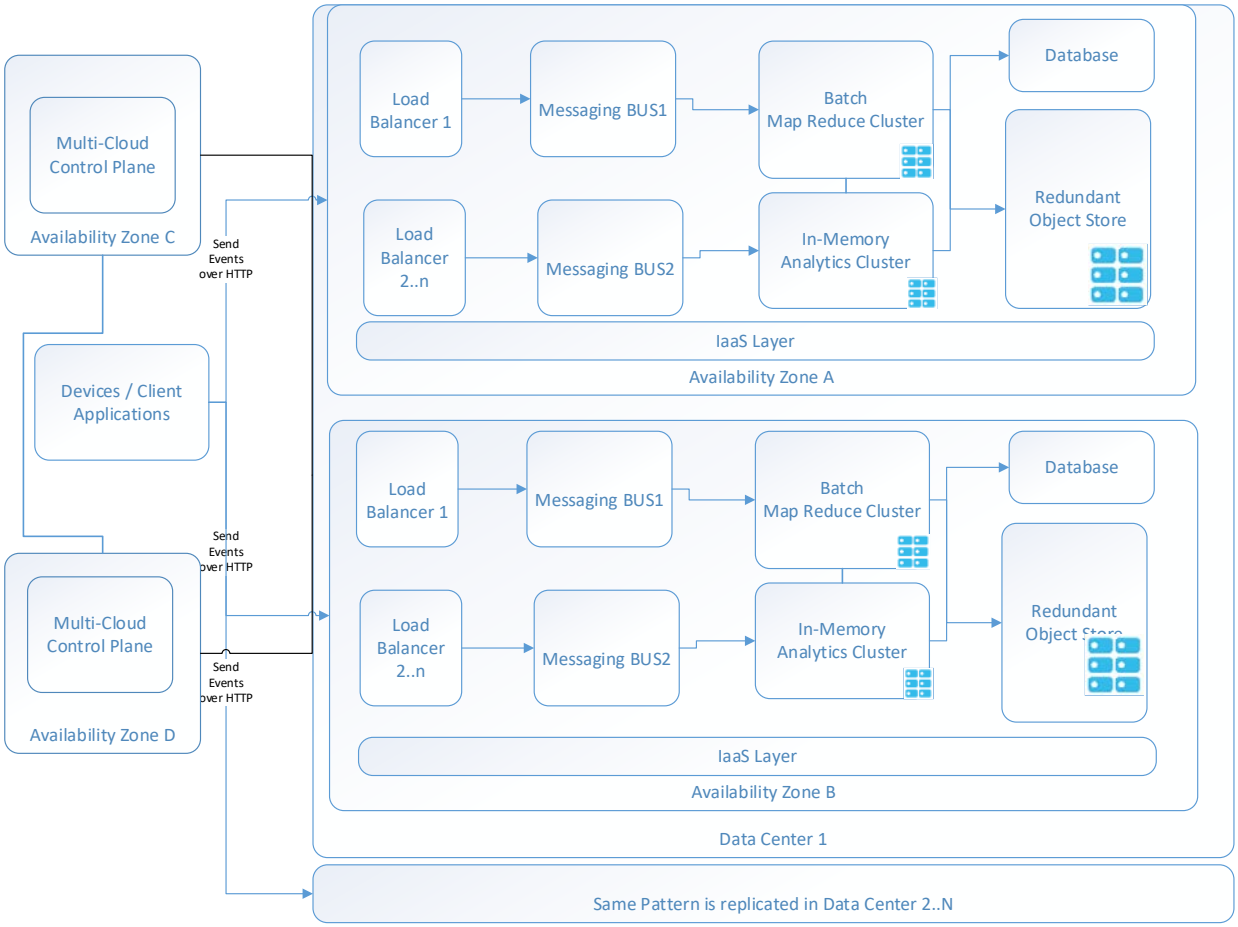


Figure 70 - Conceptual Multi-Cloud Internet of Things Event Stream Ingesting Pattern and Big Data Pipeline

- **Participants:** A listing of the Software and Infrastructure Components used in the pattern and their roles in the design.
- **Consequences:** A description of the results, side effects, and trade-offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern. Presents vendor independent logical pattern with specific technology implementation

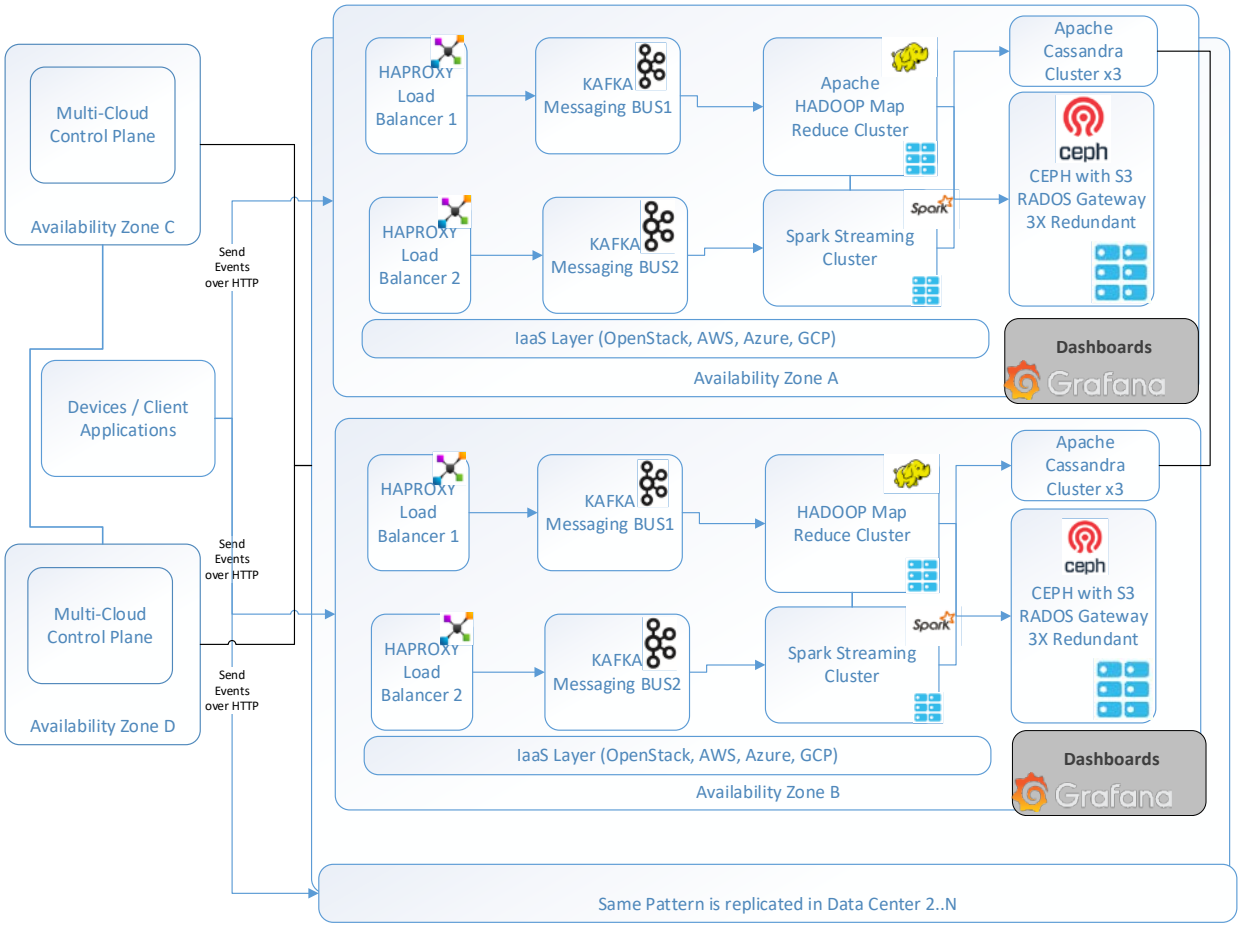


Figure 71 - Multi-Cloud Internet of Things Event Stream Ingesting Pattern and Big Data Pipeline Implementation

- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

Linux Containers have been traction recently for multiple reasons, some of them are:

- Faster start up than Virtual Machines
- Uniform application environment from development to production
- Better ability to scale up and increase utilization density

So, what if we want to run my Application or Service in Containers – how do we do this in multi-cloud environment with proper fault tolerance and self-healing in mind?

4.6.3 Container Orchestration Pattern for Multi-Cloud Deployments

Once we have IaaS Virtual Machines created then we can deploy containers to subdivide virtual machines. The benefit would be higher utilization and higher multi-tenancy density [6] here is an example:

- **Pattern Name and Classification:** Container Orchestration Pattern
- **Also Known As:** Container cluster manager / master
- **Problem:** Virtual Machines take from couple of seconds to tens of seconds or even minutes to startup. Is there a faster way to startup and provide virtualization at the same time? Furthermore, the deployment of an application on IaaS are done in virtual machines usually leave the Virtual Machine frequently underutilized. Is there a way to subdivide the Virtual Machine so we can pack more workloads in it vertically a.k.a. vertical scalability?

Container is kernel virtualization which focuses on CPU, Memory and Storage isolation. Containers were originally introduced as a concept Solaris as Linux containers i.e. LXC, Docker, Rkt.

The core problem is how do you deploy and manage containers? Enter Container Orchestration Pattern which extends Resource Orchestrator and Resurrection Pattern focusing on deploying, coordinating and resurrecting containers.

Containers Allow Vertical Scalability/Density of Applications vs. Virtual Machines

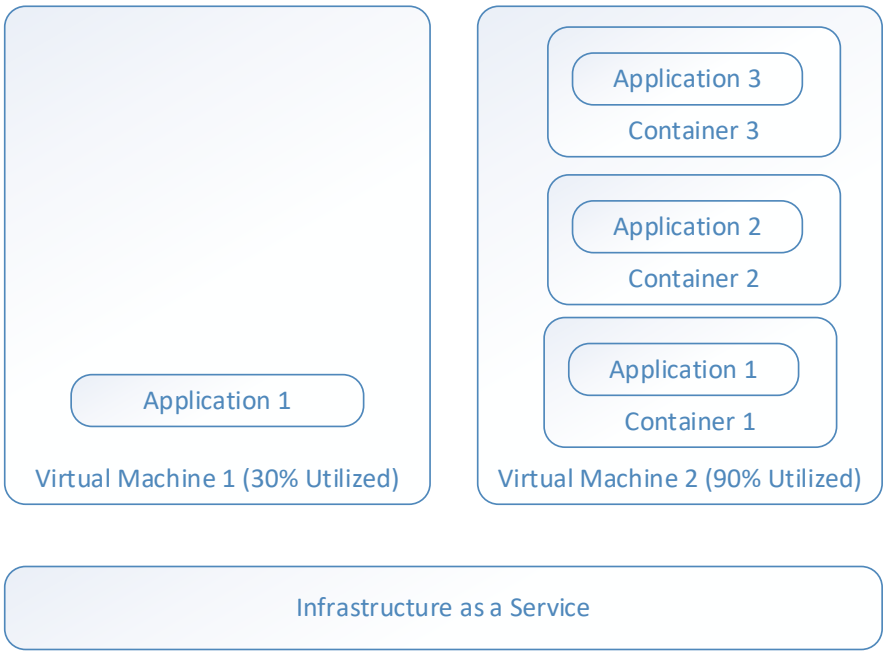


Figure 72 - Containers vs. Virtual Machines

- **Intent:** light weight virtualization, provide more efficient utilization of virtual machines of IaaS provider, faster startup
- **Motivation (Forces):** light weight virtualization, provide more efficient utilization of virtual machines of IaaS provider, faster startup

- **Applicability:** any IaaS deployment or bare metal deployment, Linux containers, Docker etc.

- **Structure:**

  Multi Cloud Container Orchestration with Multi Cloud Control Plane

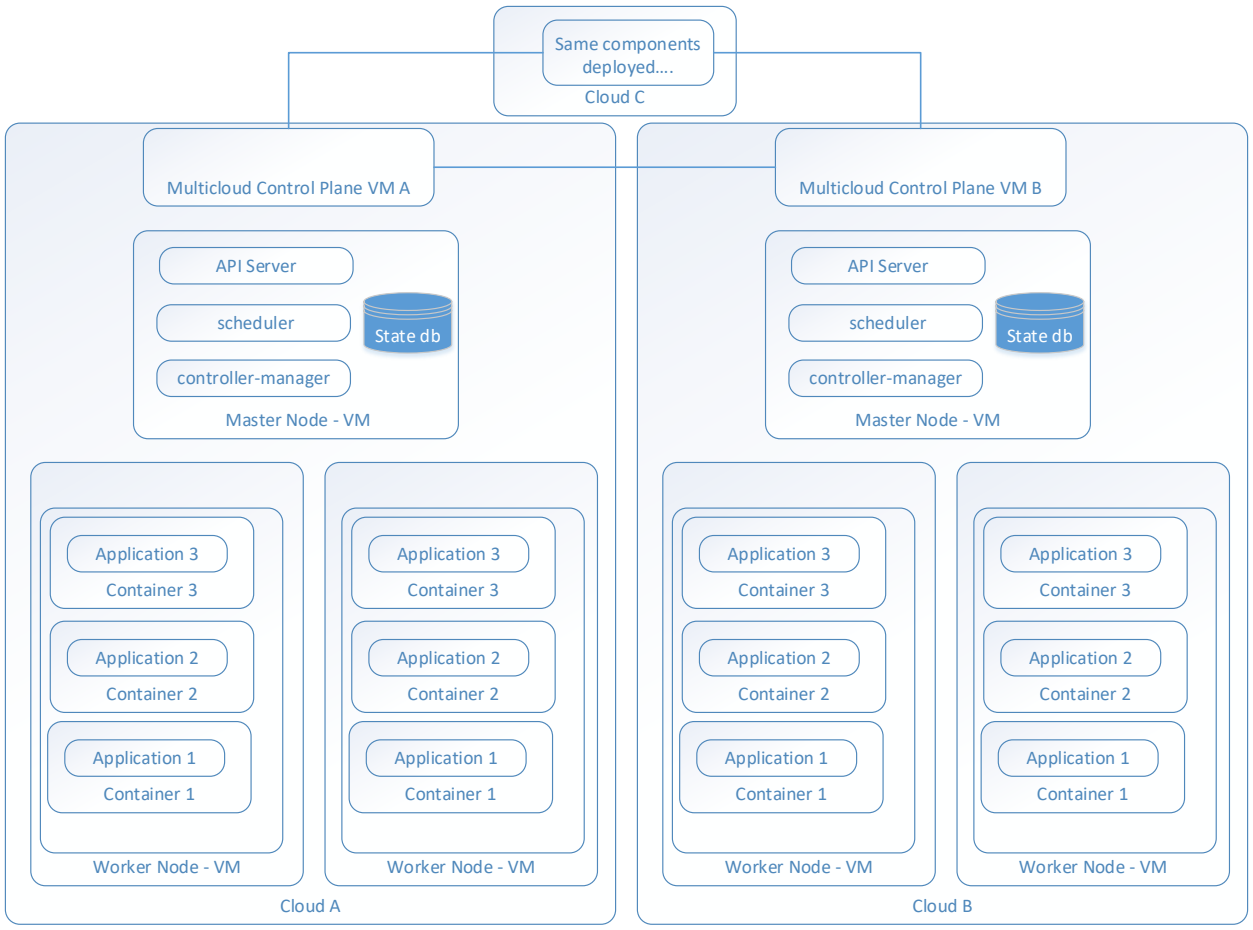


Figure 73 - Conceptual View of Container Orchestration Pattern for Multi-Cloud Deployments

- **Participants:**

  - Master Node VM

  - Container Node VM

- **Consequences:** light weight virtualization, faster startup of container vs. virtual machines, provides more efficient utilization of virtual machines of IaaS provider. Self-Healing of containers and better fault tolerant distribution

- **Implementation:**

  Pattern Implementation Illustration with Kubernetes

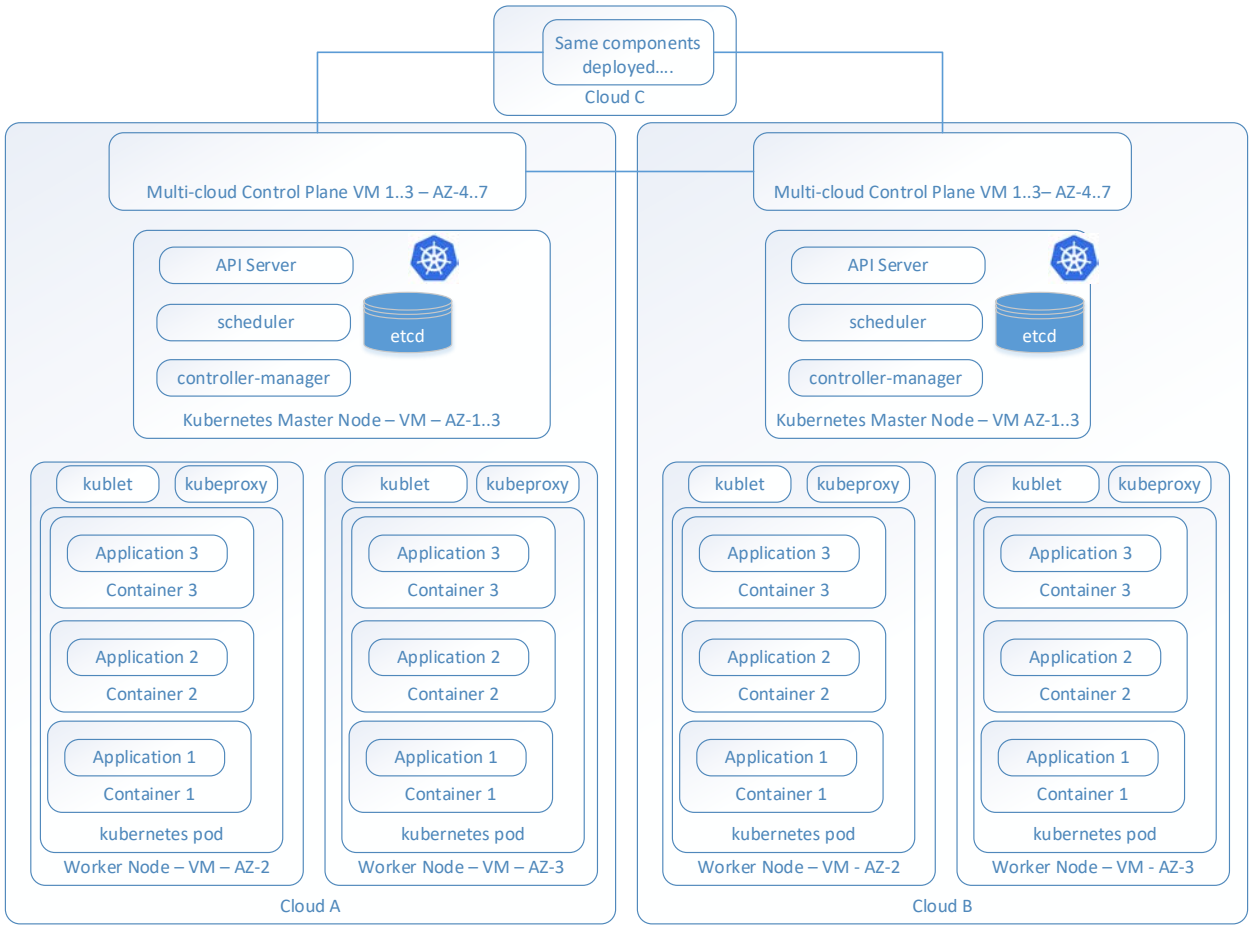


Figure 74 - Container Orchestration Pattern for Multi-Cloud Deployments Implementation View

- **Known Uses:** Kubernetes (https://kubernetes.io), Docker Swarm (https://docs.docker.com/engine/swarm/), marathon (on top of Apache Mesos - http://mesos.apache.org, https://mesosphere.github.io/marathon/)

  Docker (http://docker.io)

- **Related Patterns:** Virtual Machine Orchestrator

4.7 Conclusion and Pattern Mappings to the Particular Problem

In this chapter, we have covered the key patterns that compose Multi-Cloud Control Plane framework that will allow any organization to deploy to multiple clouds with open source software and minimum cloud provider lock in as well as better resilience and self-healing capabilities.

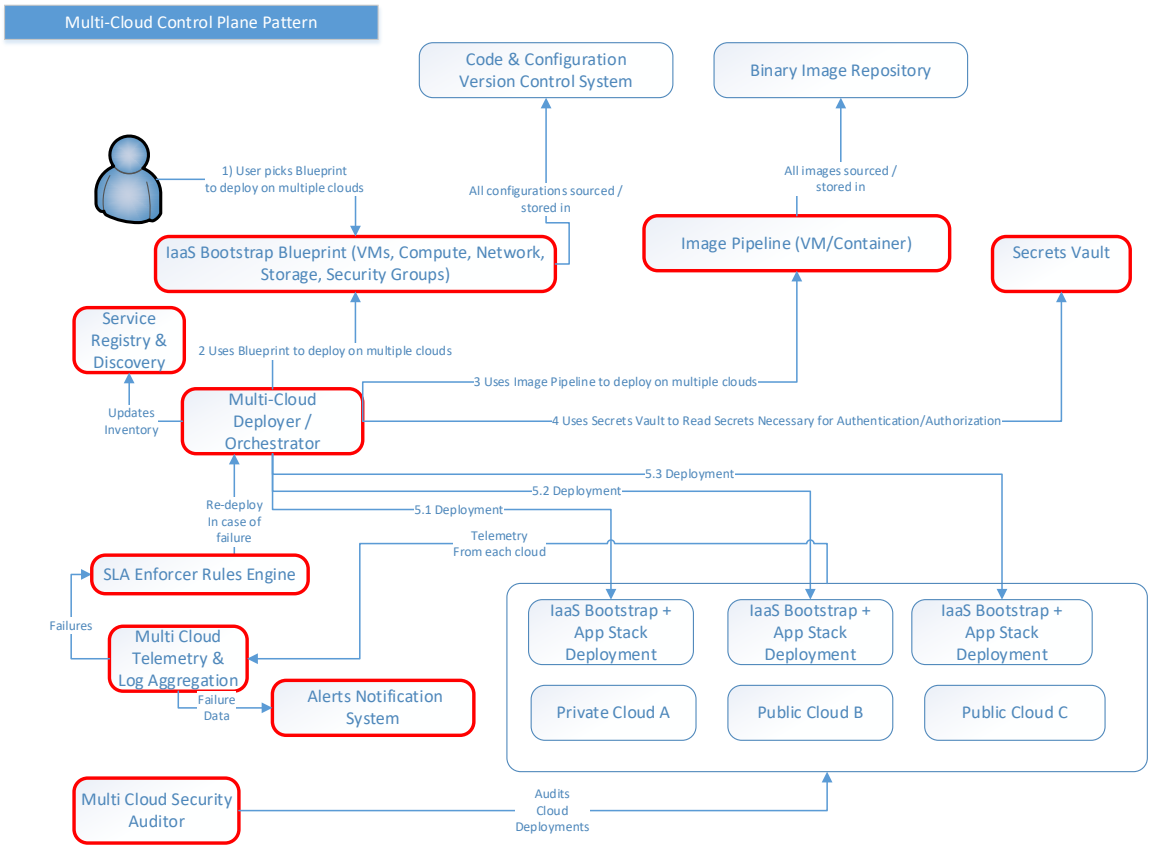


Figure 75 - Multi-Cloud Control Plane Framework Conceptual View

Next let's see how the Open Source frameworks implementation for majority of components maps on top of this framework:

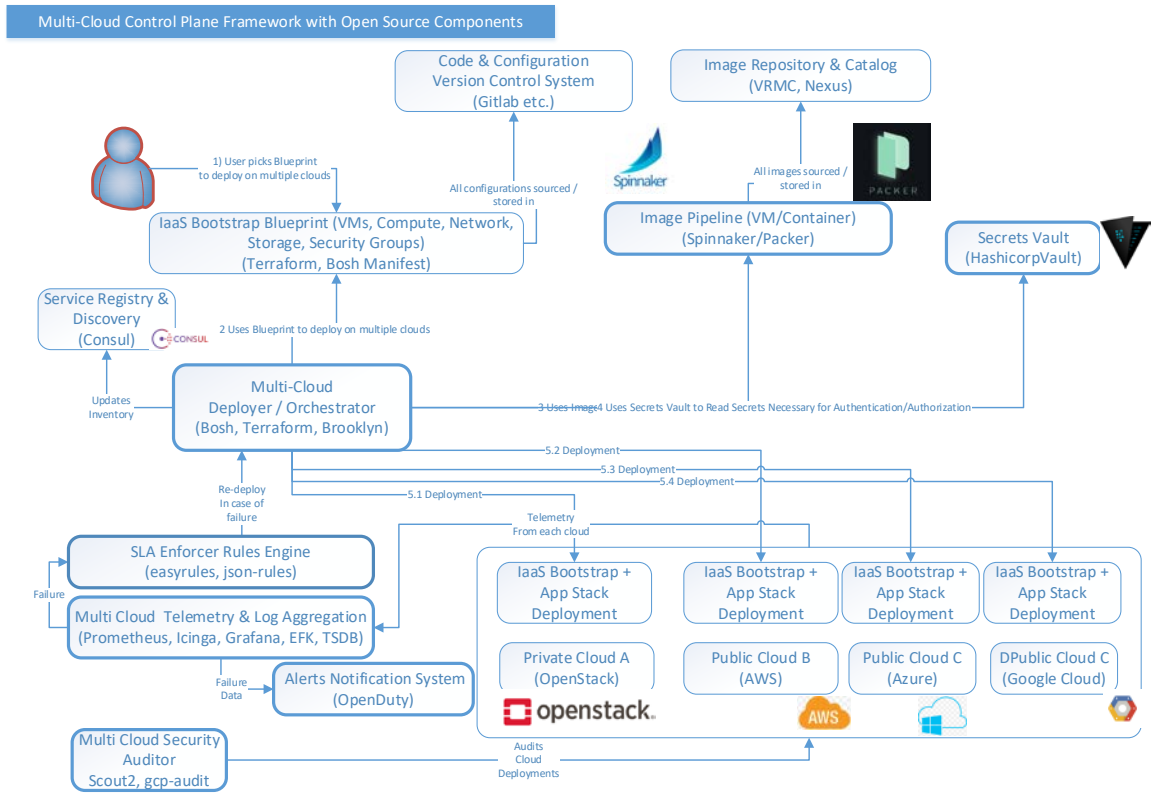


Figure 76 - Multi-Cloud Control Plane Framework Implementation View

Here is Key References to Implementation Code and Scripts for Each Pattern:

Multi-Cloud Blueprint Bootstrap Options

https://github.com/compscied/multi-cloud/tree/master/cloud-blueprint-bootstrap-options

Multi-Cloud Auditor Pattern Implementation Options:

https://github.com/compscied/multi-cloud/tree/master/cloud-auditor

Multi-Cloud Software Stack Bootstrap on any major IaaS Provider

https://github.com/compscied/multi-cloud/tree/master/cloud-blueprint-bootstrap-options

Multi-Cloud SLA Enforcer Rules Engine

https://github.com/compscied/multi-cloud/tree/master/sla-rules-engine

Database options appropriate for Multi-Cloud deployment or Multi-Datacenter Deployment with tolerance to high network latency

https://github.com/compscied/multi-cloud/tree/master/databases-multi-cloud

Multi-Cloud Image Pipeline

https://github.com/compscied/multi-cloud/tree/master/image-pipeline

Multi-Cloud Cost and Billing

https://github.com/compscied/multi-cloud/tree/master/multi-cloud-billing

Implementation Options for Multi-Cloud Virtual Machine Registry/Catalog/Repository

https://github.com/compscied/multi-cloud/tree/master/multi-cloud-vm-image-repository

Secret Management

https://github.com/compscied/multi-cloud/tree/master/secret-management

Options that can work in multi-cloud environment to help with service registry / discovery

https://github.com/compscied/multi-cloud/tree/master/service-registry-discovery

Multi-Cloud Telemetry

https://github.com/compscied/multi-cloud/tree/master/telemetry-multi-cloud

Multi-Cloud IOT

https://github.com/compscied/multi-cloud/tree/master/iot-multi-cloud

Multi-Cloud Container Orchestration

https://github.com/compscied/multi-cloud/tree/master/container-management-multi-cloud

Finally here is how the major challenges / problems we have covered map to the patterns side by side

Major Problems / Challenges with multi-cloud computing	**Patterns Solution for Each Problem**
Initial Multi-Cloud General Deployment Fault Tolerance Challenges:	**Initial Multi-Cloud Fault Tolerant Deployment Patterns**
• How to deploy with maximum fault tolerance and isolation?	• General Multi Availability Zone Fault Tolerant Pattern
• How do we route to multiple providers all at once with maximum fault tolerance in mind? • If the cloud provider fails what do we do? Can we route the requests and workloads to another cloud provider? • What if Domain Name System fails – how do we continue operating?	• General Multi-Cloud Cloud Fault Tolerant Routing Pattern

• How do we deploy on multiple cloud providers if all of them have different APIs? • More importantly is there a way where you can define your infrastructure requirements once and something will take care of translating this into the API calls for the specific provider?	• Multi-Cloud Cloud Blueprint Pattern
• Cloud providers require different virtual machine formats – how do create these images in an automated way?	• Image Build Pipeline Pattern for Multi-Cloud Deployment
• Finally, where do we store and keep track of information about everything we deployed?	• Multi-Cloud Service Registry and Discovery API
Multi-Cloud Management after initial deployment and dealing with failures:	**Multi-Cloud Management after Initial Deployment and Dealing With Failures Patterns**
• Once we deploy how do we find out the health of the instances?	• Reactive Multi-Cloud Health check and Load Balancing Pattern • Multi-Cloud SLA Monitoring Pattern
• If the node virtual machine fail - can we recover them automatically?	• SLA Enforcer Rules Engine

• If availability zone fails – can we recover automatically? • What if our deployment infrastructure fails – how do we re-deploy? • How do we fail over gracefully when fault occurs?	
• How do we make sure database is always available even if one database instance fails?	• Multi-Cloud Data Replication • Multi-Cloud Disaster Recovery Pattern
• Is there more intelligent way of enforcing Service Level Agreement if we know there is a degradation in performance or things are about to fail – before failure occurs?	• Proactive Multi-Cloud SLA Policy Enforcement Pattern
• What if I want to deploy to public cloud only if private cloud capacity gets exhausted? • What if private cloud fails how can recover from this disaster using a public cloud?	• Public Cloud Bursting Pattern
Cost Efficiency Problems	**Cost Efficiency Patterns**
• How do we actually aggregate billing and cost from all cloud deployments?	• Multi-cloud Aggregate Billing and Chargeback Pattern

• How do we make sure that cost is not out of control and we know how much we are spending as well as deploy workloads where it is cheaper?	• Cost Efficiency Discount Multi-Cloud Pattern
Security Problems	**Security Related Patterns**
• Where do we store and retrieve secrets?	• Secret Storage and Retrieval – Secrets Vault Pattern
• How do we ensure that deployment does not have any known vulnerabilities and gets patched if there is a vulnerability? • How do we continuously audit the cloud and make sure there are no unauthorized or unintended changes?	• Multi-Cloud Auditor Pattern
Application Deployment in Multi-Cloud environment	**Multi-Cloud Application Deployment Patterns**
• What if I want to run Applications in Containers – how do I do this in multi-cloud environment with proper fault tolerance in mind?	• Container Orchestration Pattern
• How do we deploy real applications in Multi-Cloud environment so if something fails the user or service does not experience and interruption?	• Multi-Cloud Web Application / Service Pattern

• How to deploy internet of things services in a multi-cloud environment?	• Multi-Cloud Internet of Things Event Stream Ingesting Pattern

Chapter 5 - Experimental Validation

5.1 Implementations for Key Components of the Multi Cloud Framework

In this chapter, we will cover the following:

- First we will start with ***General Multi-Cloud Cloud Deployment Pattern*** that we need for high level understanding and putting everything together
- After that we will cover ***Multi-Cloud Cloud Blueprint Pattern*** which we will help us with planning and "bootstrapping" of IaaS layer.
- In order to complete bootstrapping, we will need to use ***Image Build Pipeline Pattern***
- Once everything is deployed we will demonstrate which **Multi-Cloud Service Registry and Discovery**
- At the end of the all these steps we have fully functional and automated deployment on all cloud providers
- Additionally, we will cover **Multi-Cloud Container Orchestration** and **Multi-Cloud Data Replication**

Cloud Providers

We will use 3 leading cloud providers to validate selected patterns and multi-cloud framework.

- Amazon Web Services - EC2
- Google Cloud Platform – GCP
- Microsoft Azure

- OpenStack for on premises cloud deployments

5.2 General Approach to Validation

Validation Criteria #1 – Multi-cloud

All of the frameworks and components composing the patterns need to work on all major IaaS cloud provider's platforms such as Public Cloud AWS, Azure Google Cloud and well as Private VMWare ESX and OpenStack. All patterns will have a reference to actual scripts that will help you to install the product on all these providers.

Validation Criteria #2 - All Components need to be Free and Open Source:

All of the frameworks and components composing the patterns must be free and open source. Patterns will have a reference to actual scripts that will help you to install the product.

Validation Criteria #3 - Validating Fault Tolerance:

Manual Validation is accomplished by shutting down and deleting virtual machines and other components.

Automated Validation is with open source Netflix Simian Army framework that include:

- Chaos Monkey – framework to simulate random outages with Virtual Machines, Security Groups etc.

- Chaos Gorilla - framework to simulate random outages with Availability Zones

- Latency Monkey – framework to introduce network artificial delays

https://github.com/Netflix/SimianArmy

Chaos Monkey: "is a resiliency framework that randomly terminates virtual machine instances and containers that run inside of your cloud environment. Chaos Monkey is fully integrated with Spinnaker (Image Pipeline Pattern), the continuous delivery platform that we use at Netflix. Chaos Monkey works with any backend that Spinnaker supports (AWS, GCP, Azure, Kubernetes, Cloud Foundry). It has been tested with AWS and Kubernetes." Further reference: https://github.com/Netflix/chaosmonkey

Latency Monkey: "induces artificial delays in our RESTful client-server communication layer to simulate service degradation and measures if upstream services respond appropriately. In addition, by making very large delays, we can simulate a node or even an entire service downtime (and test our ability to survive it) without physically bringing these instances down. This can be particularly useful when testing the fault-tolerance of a new service by simulating the failure of its dependencies, without making these dependencies unavailable to the rest of the system."

Chaos Gorilla helps to automate simulation of failure of entire availability zone – another words multiple virtual machines corresponding to that availability zone will down during simulation. Once that occurs the expectation is that SLA Enforcer Rules Engine pattern will resurrect those instances in the new availability zone.

Additional Reference:

http://techblog.netflix.com/2011/07/netflix-simian-army.html

Next let's proceed with validation of key patterns.

5.3 General Multi-Cloud Cloud Fault Tolerant Routing Pattern Validation

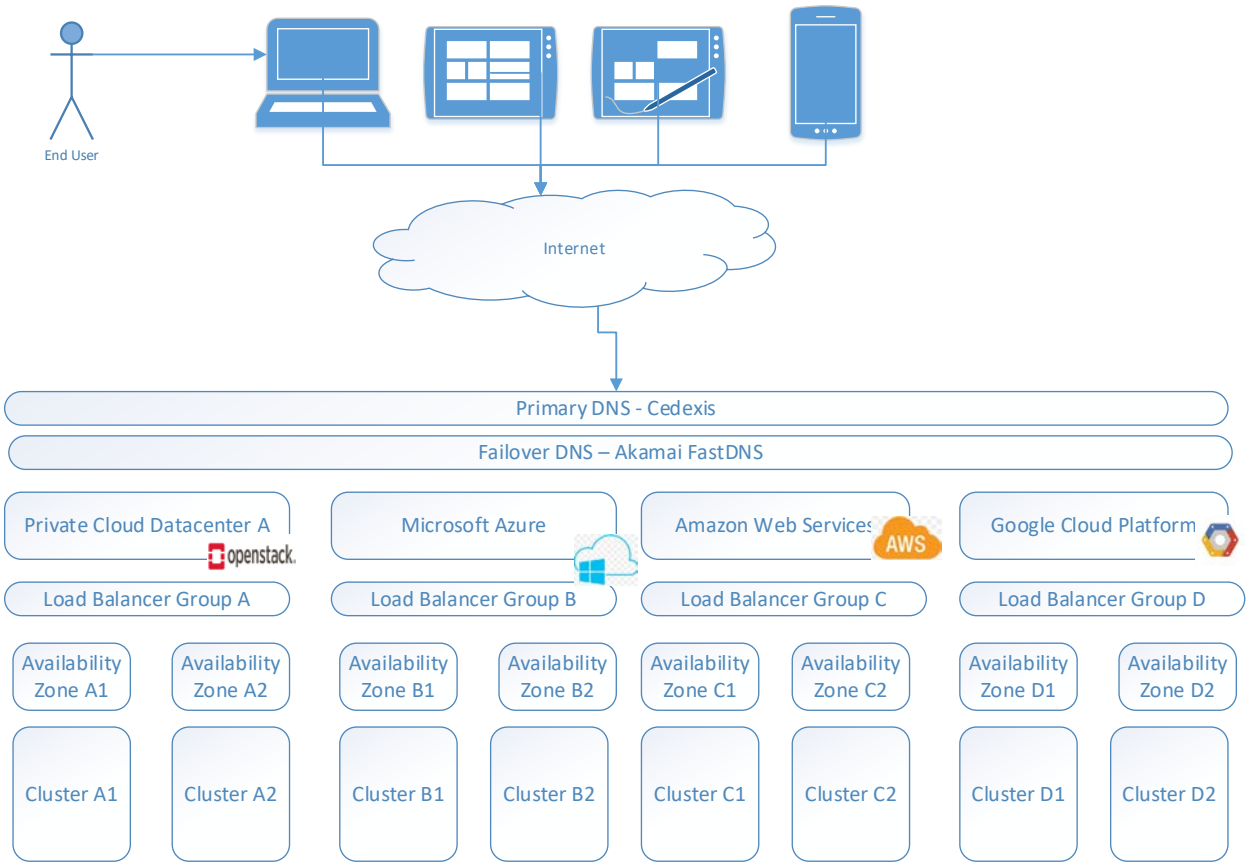


Figure 77 - Multi-Cloud Cloud Fault Tolerant Routing Pattern Validation

Problem: Single cloud provider or private data center is not sufficient for fault tolerance and geographical availability. Single cloud provider leads to vendor lock in. How do we distribute traffic and route among different cloud providers?

Solution:

We will use multiple global DNS providers that use anycast protocol

We will use redundancy to make external DNS disaster proof.

We want to avoid unicast servers, because that usually means two DNS nameservers in different locations. A better alternative is an anycast DNS cloud to provide redundancy. If a DNS nameserver in an anycast cloud goes down, it is automatically removed from the routing tables. In this way, anycast adds redundancy and fault tolerance.

With anycast, the highest level of redundancy is achieved with two or more separate clouds. When compared to unicast redundancy, it is like replacing two unicast nameservers with two anycast clouds. Make sure the clouds use independent hardware and transit providers. This protects against a routing problem or transit network outage from bringing down your DNS.

DNS

- Global DNS Highly Available providers:
- Akamai FastDNS https://www.akamai.com/us/en/solutions/products/cloud-security/fast-dns.jsp
- Cedexis http://www.cedexis.com/products/openmix/
- Nustar UltraDNS https://www.neustar.biz/services/dns-services

Validation:

Key Validation Failure Scenario	**Result**
1 DNS provider outage or distributed denial of service attack	Failure detected, automated failover runs, 2nd DNS provider takes over http://www.dnsmadeeasy.com/services/dnsfailover/ DNS was not designed for failover - but it was designed with TTLs that work amazingly for failover needs when combined with a solid monitoring system. TTLs can be set very short. I have effectively used TTLs of 5 seconds in production for lightning fast DNS failover based solutions. You have to have DNS servers capable of handling the extra load - and named won't cut it. However, powerdns fits the bill when backed with a mysql replicated databases on redundant name servers. You also need a solid distributed monitoring system that you can trust for the automated failover integration. Zabbix works for me - I can verify outages from multiple distributed Zabbix systems almost instantly - update mysql records used by powerdns on the fly - and provide nearly instant failover during outages and traffic spikes.
Any Cloud Provider Failure	Traffic routed to another Cloud Provider, no downtime
All DNS Providers Failure	API is used to look up and route to well know IPs of cloud components

Failure Illustrations

General DNS Failover Approach

When primary DNS fails, secondary provider takes over.

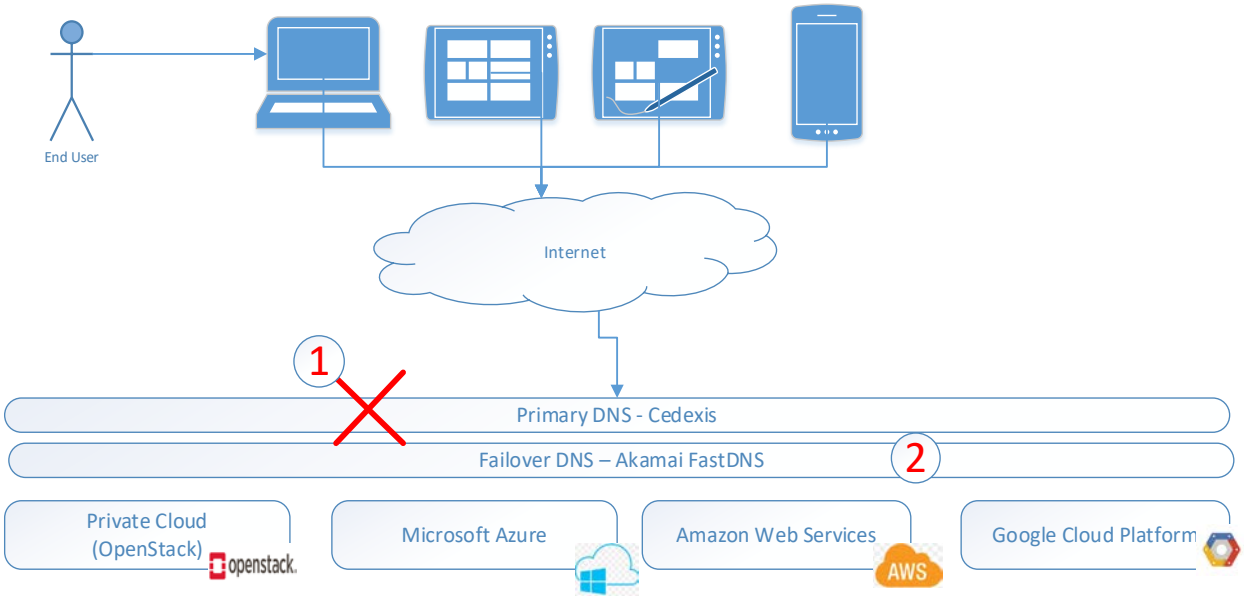


Figure 78 - General DNS Failover Approach

Next Primary and Secondary DNS Failure – API end points are used for failover

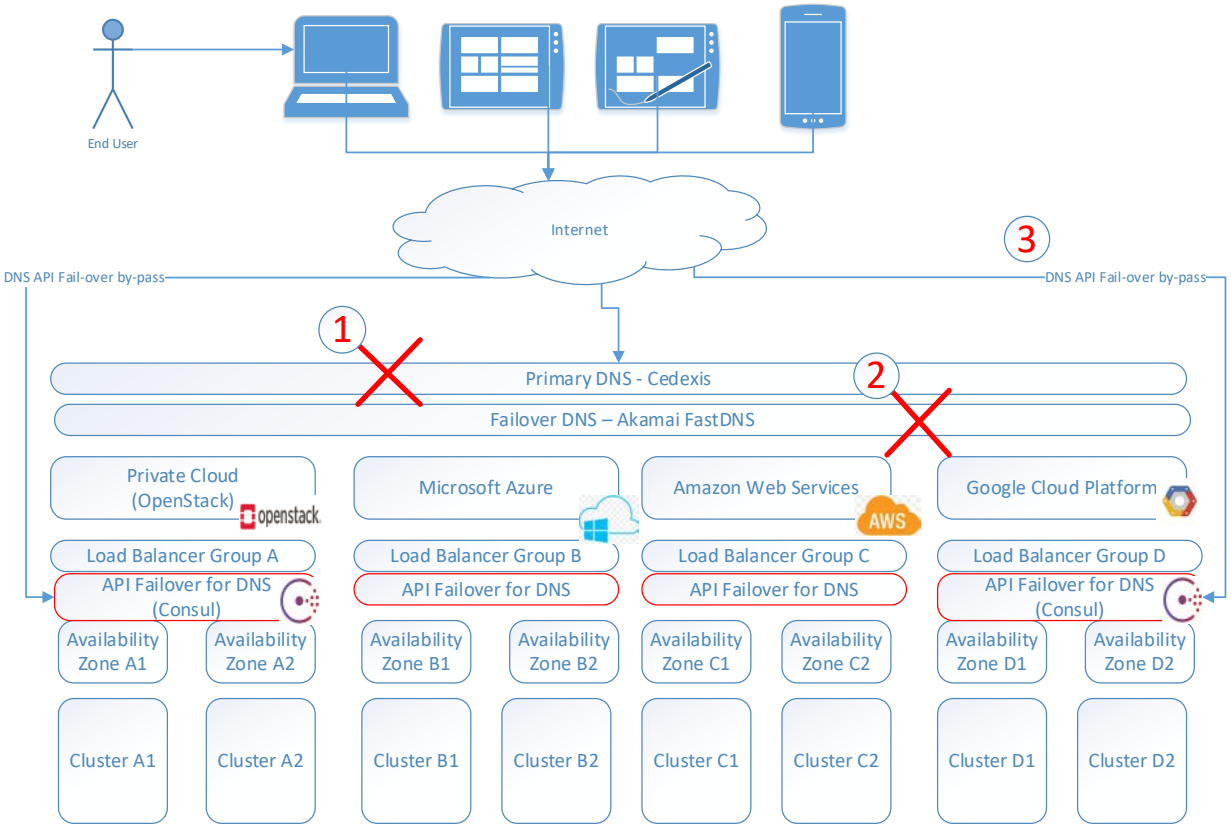


Figure 79 - Primary and Secondary DNS Failure – API End Points are Used for Failover

Further reference:

http://www.dnsmadeeasy.com/services/dnsfailover/

https://support.dnsimple.com/articles/differences-a-cname-records/

https://blog.serverdensity.com/multi-data-center-redundancy-sysadmin-considerations/

5.4 Multi-Cloud Cloud Blueprint and Multi-Cloud Deployer Pattern Validation

In chapter 4 we have discussed that original problem was that currently there is no uniform API and it is not uniformly implemented in the same way across cloud providers. In order to automate provisioning of multiple IaaS this pattern helps to describe your cloud infrastructure as a blueprint and use that blueprint to drive orchestrator to target specific IaaS provider and create all the resources necessary for your application and services to run on any cloud provider (Public or Private).

Pattern without implementation

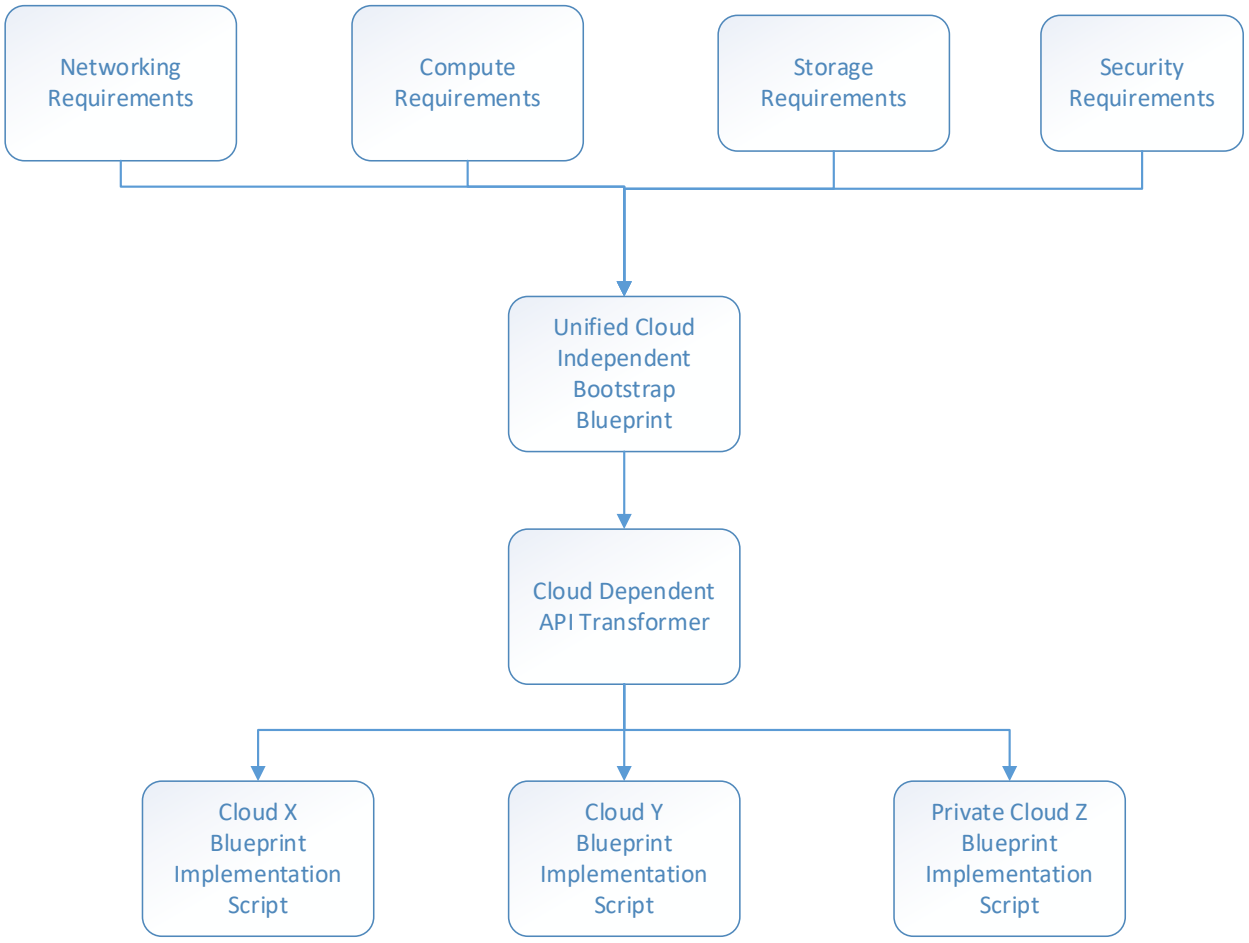


Figure 80 - Multi-Cloud Cloud Blueprint and Multi-Cloud Deployer Pattern Validation - Conceptual

The Multi-Cloud Deployer will then take the deployment and create all desired components in every target cloud environments:

Next let's take a look at Open source solution options:

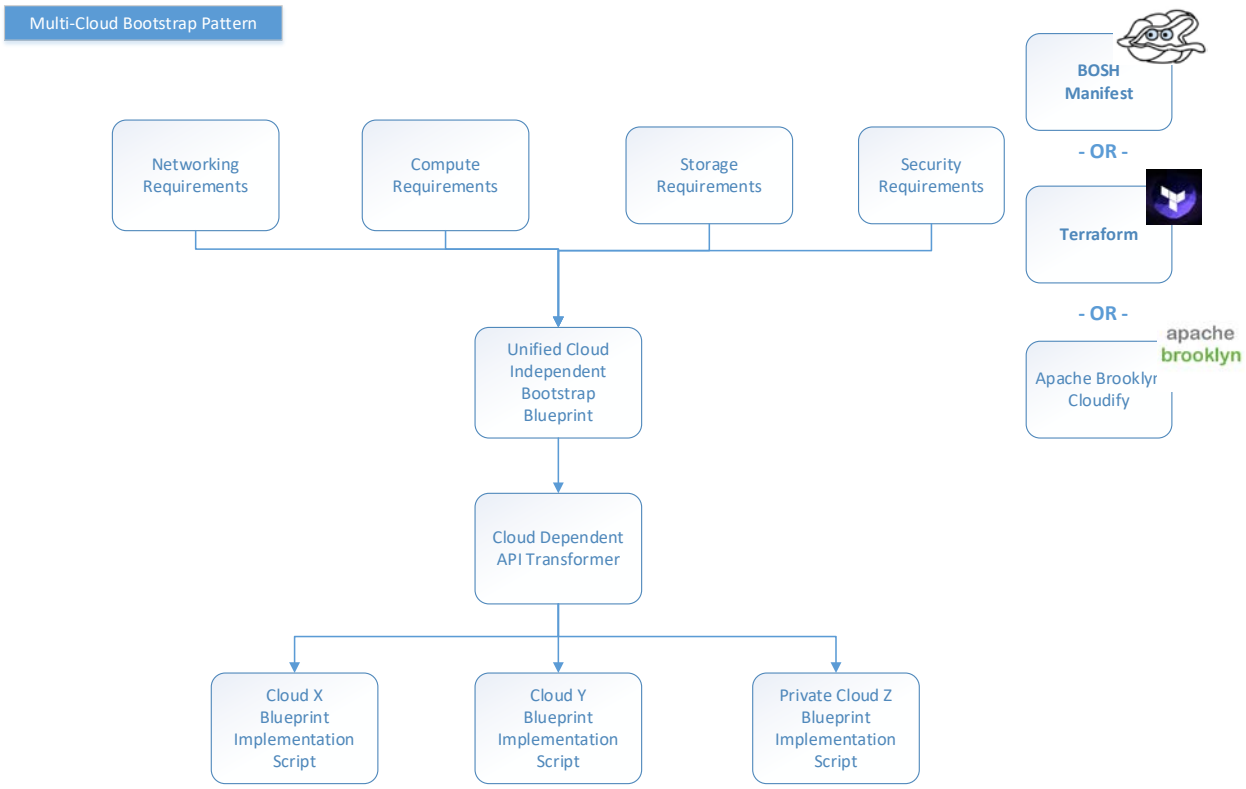


Figure 81 - Multi-Cloud Cloud Blueprint and Multi-Cloud Deployer Pattern Validation - Implementation View

Although none of these fully support all desired properties and functions for the basic foundation these will work. However, these options are fully free an open sourced with large community of contributors and support all major public/private cloud IaaS deployment options.

There are two major standards in this area: OASIS CAMP (Cloud Application Management for Platforms) and TOSCA (Topology and Orchestration Specification for

Cloud Applications) both define YAML application models. CAMP focuses on the REST API for interacting with such a management layer, and TOSCA focuses on declarative support for more sophisticated orchestration.

Let's start with standards based options that were researched and prototyped for this pattern:

Apache Brooklyn allows to Model, Deploy and Manage infrastructure blueprints and configuration using multiple public cloud providers and on premises. Apache Brooklyn strives to support standards – it uses a YAML which complies with CAMP's syntax and exposes many of the CAMP REST API endpoints

https://brooklyn.apache.org/

Cloudify is built on top of Apache Brooklyn and uses standards based TOSCA YAML to describe infrastructure blueprints and supports

www.getcloudify.org

https://github.com/cloudify-examples

http://cloudify.co/examples/home.html

Non-standard's (but very popular) based options include:

Terraform is an open source framework for planning, building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. https://www.terraform.io/

Bosh - BOSH is an open source tool for release engineering, deployment, lifecycle management, and monitoring of distributed systems. It allows to write blueprint in

YAML once and deploy to many cloud public providers and supports private on premises deployment as well.

When an operator initiates a new deployment using the CLI, the Bosh Director receives a version of the deployment manifest and creates a new deployment using this manifest. It utilizes cloud provider specific CPI (cloud provider interface) – Additional Reference: http://bosh.io

Other frameworks considered include Chef, Puppet, Ansible but it takes much effort to make these work across cloud providers for IaaS bootstrap and these do not support standards.

Example Deployment Options with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/cloud-blueprint-bootstrap-options

The key to the right implementation of this pattern is to make sure that all key components get implemented and distributed to all different availability zones for maximum resilience.

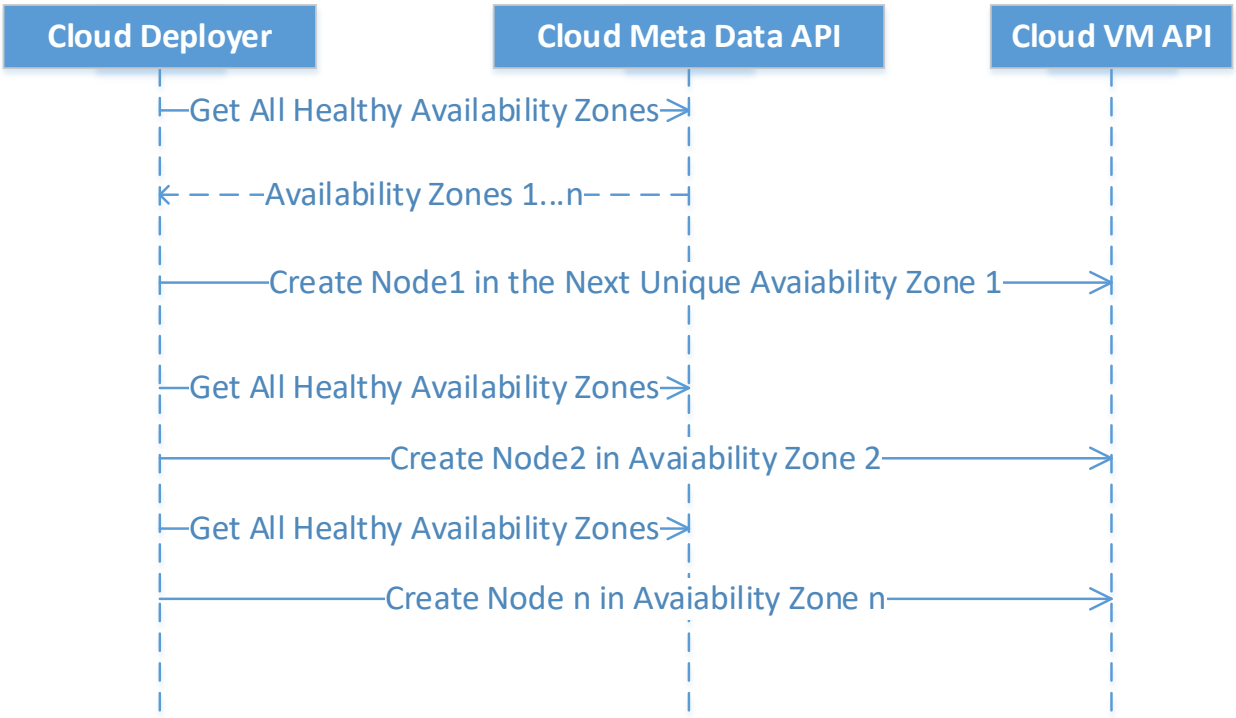


Figure 82 - Multi-Cloud Cloud Blueprint and Multi-Cloud Deployer Pattern Sequence Diagram

Validation of Fault Tolerant Multi-AZ Deployment – Illustration:

Cloud A

Virtual Machine	Availability Zone
node1	AZ A1-1
node2	AZ A2-1
node3	AZ A3-1
….	…

Cloud B

Virtual Machine	Availability Zone
node1	AZ B1-1
node2	AZ B2-1
node3	AZ B3-1

….	…

Validation - Failure Testing Scenarios

In order to validate the multi-cloud framework and framework let's review the following failure scenarios and results:

Key Failure Scenario	**Expected Result**
Virtual Machine Failure in 1 cloud provider, 1 availability zone	Traffic routed to other virtual machines, no downtime
Availability Zone Failure in 1 cloud provider	Traffic routed to other Availability Zone, no downtime
Cloud Provider Failure	Traffic routed to other Cloud Provider, no downtime
DNS Provider Failure or Distributed Denial of Service	Traffic routed via another DNS provider, no downtime
Control Plane Failure in 1 cloud instance	Since all control planes are deployed in every cloud provider. Leader election happens and another instance takes over.

5.6 Image Build Pipeline Pattern Validation

Let's recap the original problem:

How do we provide images that can run on any cloud? IaaS Image Build Pipeline Pattern is focused on building ready to boot images that contain entire software stack and even can contain application itself. (i.e. OS, Middleware, Application etc.) It is very important to have ready to boot image if you have a spike in load (via user requests, events etc.) and you need to scale out and create new instances quick. During run-time this pattern relies on Resource Orchestrator Pattern. The build pipeline also might be tailored to produce multiple images targeting multiple providers and formats in order to gain the most efficiency and exploit any cloud provider format or API differences. There is no uniform API and it is not uniformly implemented in the same way across cloud providers.

Validation of Multi-Cloud Image Pipeline Pattern

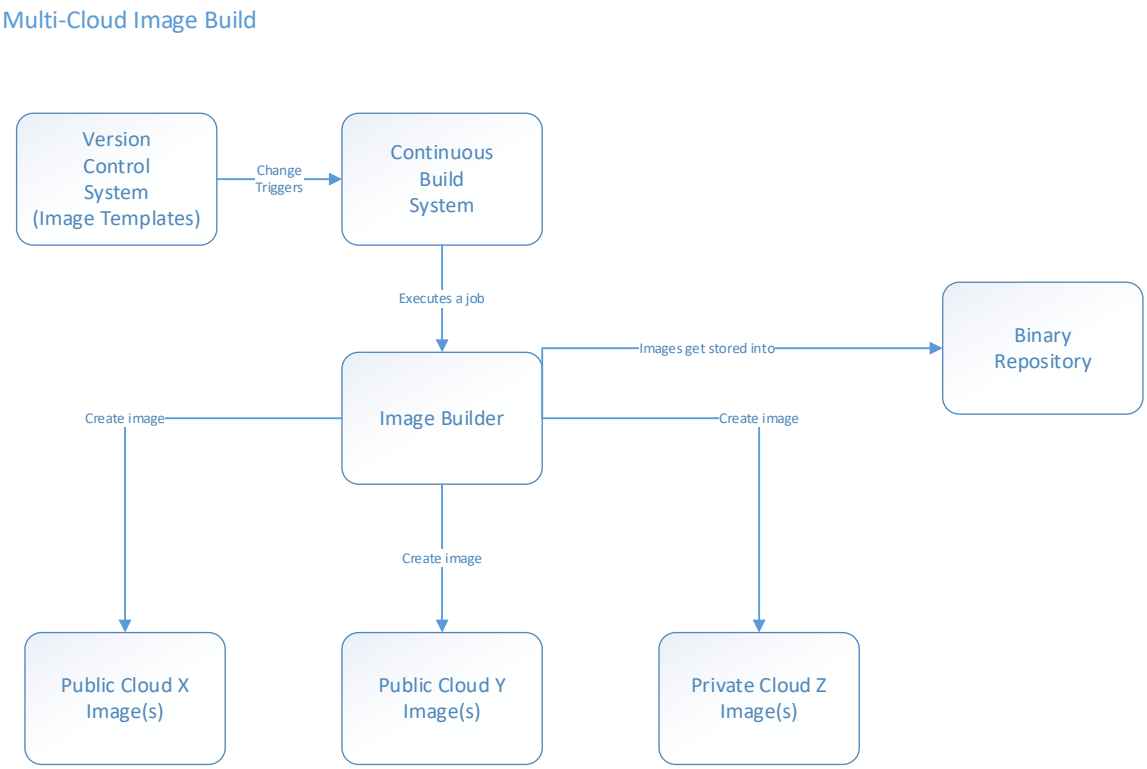


Figure 83 - Validation of Multi-Cloud Image Pipeline Pattern

Objective: provide images that can run on any cloud IaaS

Solution:

Multi-Cloud Image Pipeline can be achieved with the following great open source products

Spinnaker is an open source, multi-cloud continuous delivery platform for releasing software changes with high velocity. (Reference: http://www.spinnaker.io/)

To install spinnaker for every cloud provider mentioned here follow this guide:

https://github.com/compscied/multi-cloud/tree/master/image-pipeline/spinnaker

Packer is fundamental open source framework to be able to produce multiple images in formats that are appropriate for major public and private clouds. (Reference: https://www.packer.io)

Packer can be used with any continues build systems such as Jenkins (https://jenkins.io)

Installation instructions for Jenkins for multi-cloud deployments can be found here: https://github.com/compscied/multi-cloud/tree/master/image-pipeline/jenkins-ansible-role

The final deployment will look as following:

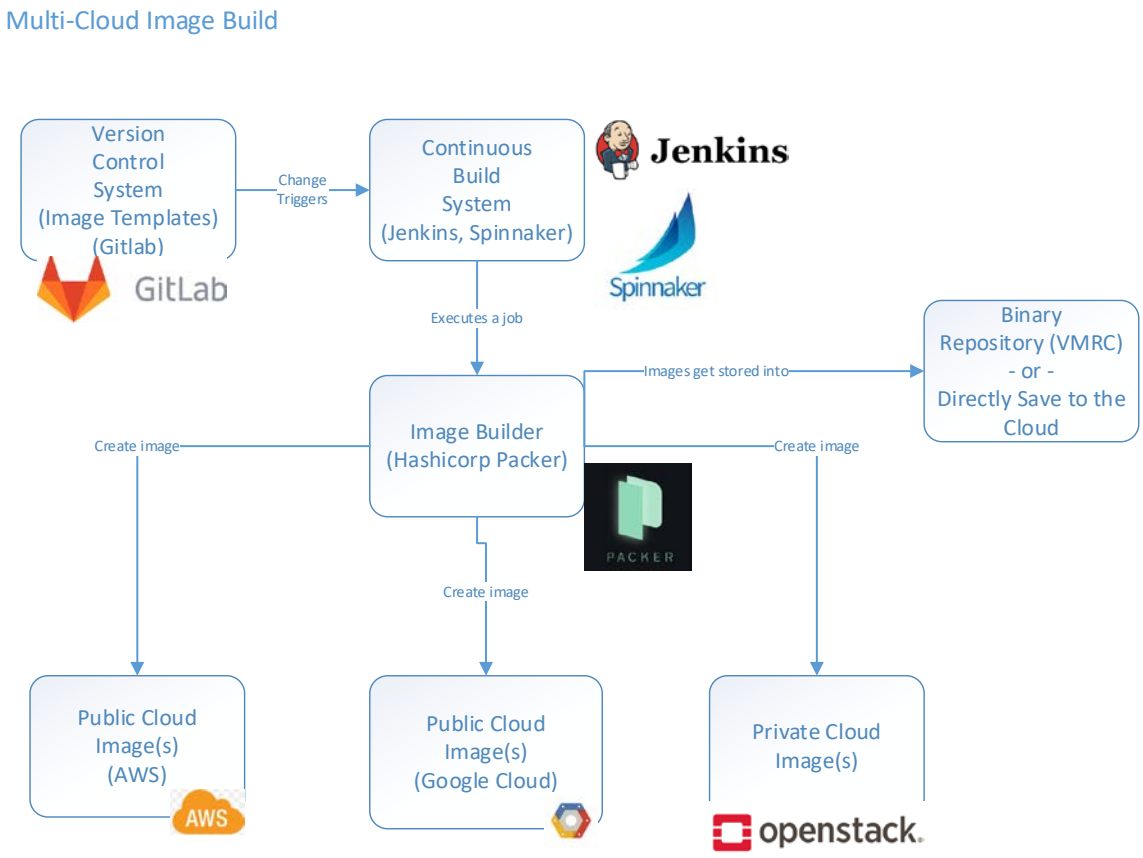


Figure 84 - Validation of Multi-Cloud Image Pipeline Pattern - Implementation View

Validation

In order to validate the Multi-Cloud Image Pipeline we need to use one of these tools to validation different images produced to target multiple cloud environments.

First we need to install packer

https://github.com/compscied/multi-cloud/tree/master/image-pipeline/packer

Next we need to create a JSON template – in this case we will be using Ubuntu:

Next we need to validate the template

```
$ ./packer validate image.json
```

it should result in output:

```
$ Template validated successfully.
```

Next we can just run

```
$ ./packer build image.json
```

Which produces the final binary image. The image can be uploaded in the cloud provider binary store or stored in a Multi-Cloud Virtual Machine Registry/Catalog/Repository.

Scripts to set it up can be found in the dissertation code repository:

https://github.com/compscied/multi-cloud/tree/master/multi-cloud-vm-image-repository

Multi-Cloud Virtual Machine Registry/Catalog/Repository is useful as a catalog of VMIs that can stored on the VMI repository systems of the different Cloud Management

Platforms (such as VMWare ESX or OpenStack) or on public Clouds (such as AWS, Azure, Google Cloud). Customized VMIs are indexed in VMRC and applications can query, as an example, for a VMI based on Ubuntu 16.04 LTS with Java already installed.

5.7 Multi-Cloud Service Registry and Discovery

Once we create IaaS deployment and our software stack is ready we need to keep track of this somewhere – this is where Multi-Cloud Service Registry and Discovery pattern comes into play.

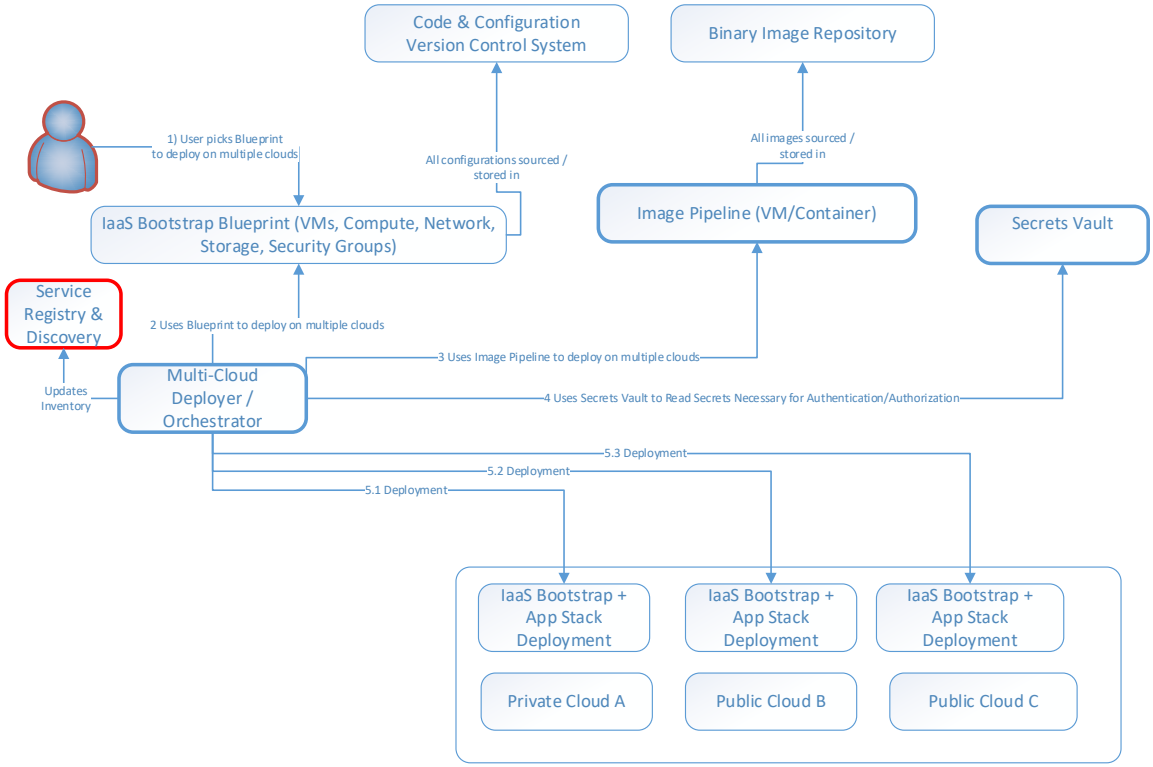


Figure 85 - Multi-Cloud Service Registry and Discovery

As an illustration of the pattern we will use an open source product HashiCorp Consul - https://www.consul.io

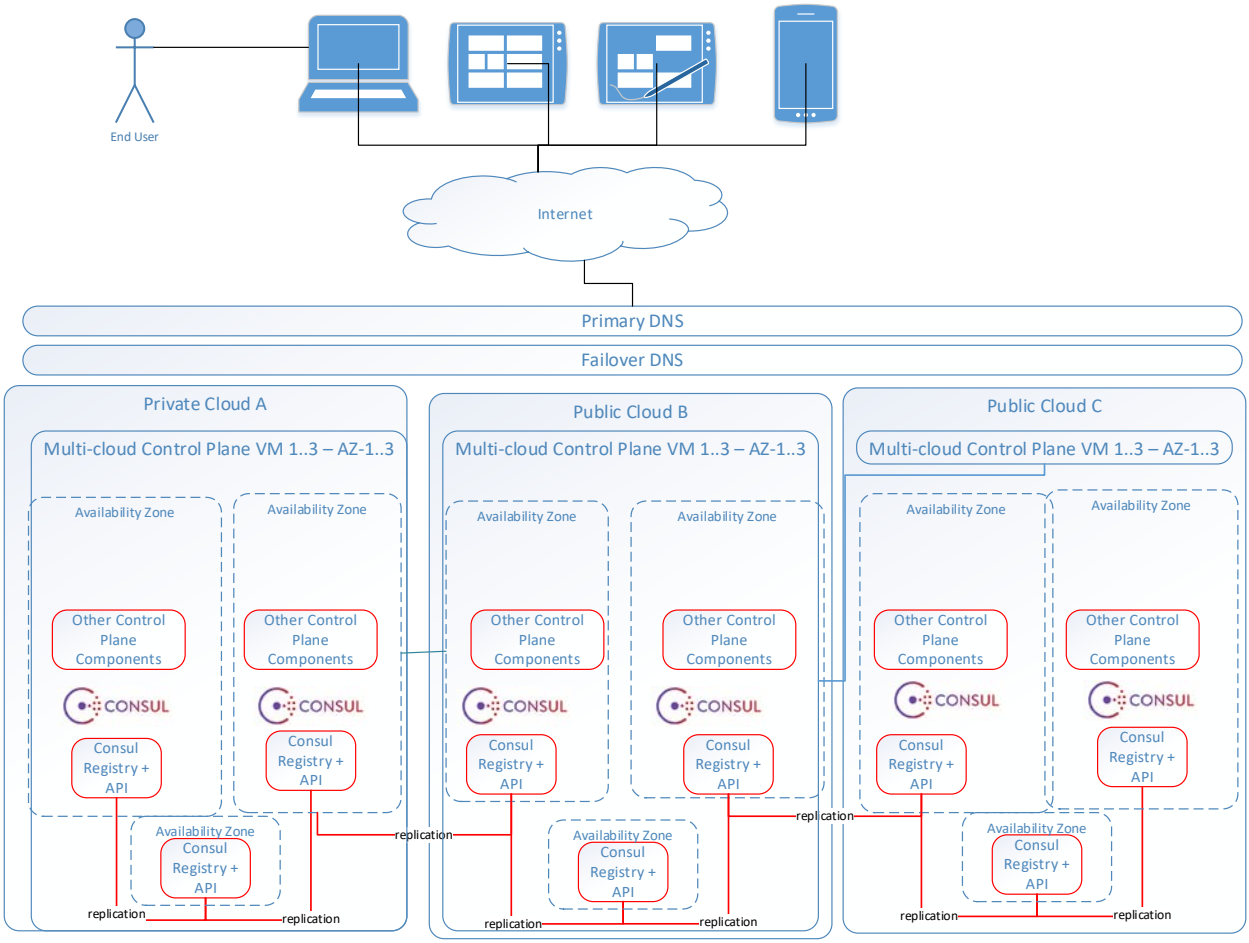


Figure 86 - Multi-Cloud Service Registry and Discovery - Implementation View

Consul has the following key features that make it appropriate for this use case:

- Multi-Cloud / Multi Datacenter: Consul scales to multiple datacenters out of the box with no complicated configuration. Look up services in other datacenters, or keep the request local.
- Service Discovery: Consul makes it simple for services to register themselves and to discover other services via a DNS or HTTP interface. Register external services such as SaaS providers as well.

- Failure Detection: Consul uses gossip protocol pairing service discovery with health checking prevents routing requests to unhealthy hosts and enables services to easily provide circuit breakers.
- Key Value Storage: Consul is also a flexible key/value store for dynamic configuration, feature flagging, coordination, leader election and more. Long poll for near-instant notification of configuration changes.

Example Deployment Options with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/service-registry-discovery/consul

Key Failure Scenario	**Result**
Control Plane Virtual Machine Failure in 1 cloud provider, 1 availability zone	Traffic routed to other virtual machines, no downtime, consul marks other nodes as down and does new leader election
Availability Zone Failure in 1 cloud provider	Traffic routed to other Availability Zone, no downtime, consul marks other nodes as down and does new leader election
Cloud Provider Failure	Traffic routed to another Cloud Provider, no downtime, consul marks other nodes as down and does new leader election

DNS Provider Failure	Traffic routed via another DNS provider, no downtime
Control Plane Failure in 1 cloud instance	Since all control planes are deployed in every cloud provider another instance takes over, consul marks other nodes as down and does new leader election

5.8 Multi-Cloud Container Orchestration Validation

Let’s start with illustration of how this pattern can be implemented with Kubernetes for container deployment and validation deployed and Managed by Multi-Cloud Control Plane for creation and in different availability zones in 2 cloud providers

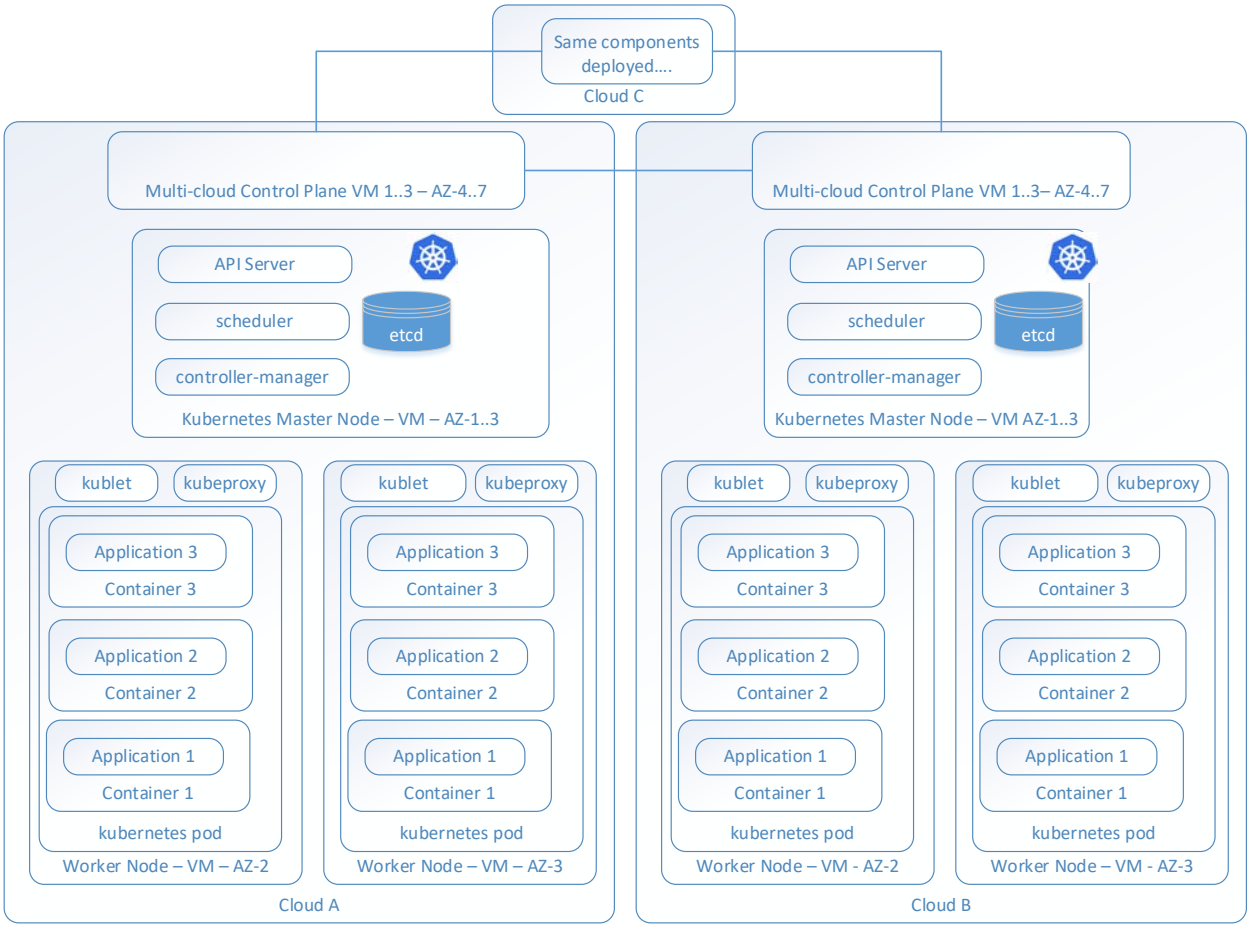


Figure 87 - Multi-Cloud Container Orchestration Validation

Example of Deployment – note all key Kubernetes components are in each availability zone:

Component	Availability Zone
Kubernetes master 1, etcd 1, worker-node1	AZ-1
Kubernetes master 2, etcd 2, worker-node2	AZ-2
Kubernetes master 3, etcd 3, worker-node3	AZ-3
….	…
Kubernetes master 5, etcd 5, worker-node5	AZ-5

Failure Scenarios

Key Failure Scenario	**Result**

Container Failure (1 or more)	Kubernetes deploys a new one on another node
Kubernetes Master/Etcd VM failure	Another master takes over, SLA Enforcer Rules Engine detects failure and creates a new Kubernetes master/etcd VM, etcd restored from backup and it joins the cluster
Kubernetes Worker Node/ VM failure	Traffic routed to other virtual machines, no downtime, SLA Enforcer Rules Engine starts resurrecting replacement VM
Any Virtual Machine Failure in 1 cloud provider, 1 availability zone	Traffic routed to other virtual machines, no downtime, SLA Enforcer Rules Engine starts resurrecting replacement VM
Availability Zone Failure in 1 cloud provider	Traffic routed to other Availability Zone, no downtime, SLA Enforcer Rules Engine starts resurrecting replacement VMs in new availability zone
Cloud Provider Failure	Traffic routed to other Cloud Provider, no downtime
DNS Provider Failure	Traffic routed via another DNS provider, no downtime
Control Plane Failure in 1 cloud instance	Since all control planes are deployed in every cloud provider another instance takes over

Example Deployment Options with open source software are scripted and provided in the companion github repository for this dissertation:

https://github.com/compscied/multi-cloud/tree/master/container-management-multi-cloud

5.9 Multi-Cloud Data Replication Pattern Validation

Multi-cloud replication is one of the more difficult problems to solve. The latency among cloud providers can be high and generally not predictable. So the one of the main

criteria's for good data replication needs to be ability to tolerate high latency during replication among cloud providers.

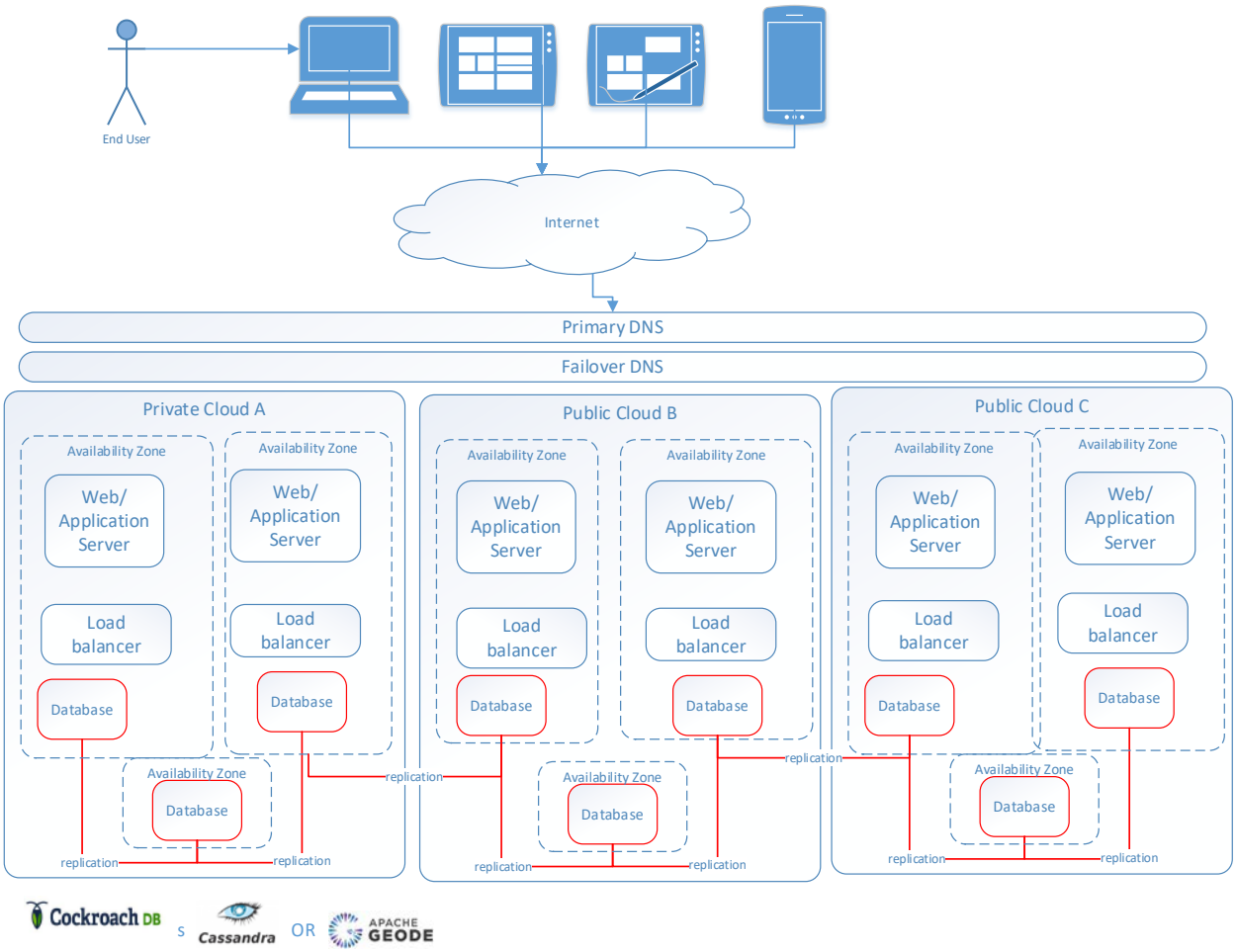


Figure 88 -Multi-Cloud Data Replication Pattern Validation

For this experimental validation we have selected 2 open source databases with that were designed with tolerance for high latency: Apache Cassandra and CockroachDB.

Apache Cassandra is columnar database and CockroachDB is closer to SQL Relational data stores.

Since majority of application use cases would probably be SQL Relational we will use CockroachDB.

To better understand this choice and compare CockroachDB please reference:

https://www.cockroachlabs.com/docs/cockroachdb-in-comparison.html

Additional Architecture Reference:

https://github.com/cockroachdb/cockroach/blob/master/docs/design.md

Example Deployment Options with open source software are scripted and provided in the companion github repository for this dissertation:

CockroachDB deployment:

https://github.com/compscied/multi-cloud/tree/master/databases-multi-cloud/cockroachdb

Apache Cassandra deployment:

https://github.com/compscied/multi-cloud/tree/master/databases-multi-cloud/cassandra

Apache Geode:

https://github.com/compscied/multi-cloud/blob/master/databases-multi-cloud/apache-geode/

Failure

Key Failure Scenario	**Result**

Virtual Machine Failure in 1 cloud provider, 1 availability zone	Traffic routed to other virtual machines, no downtime; SLA Enforcer Rules Engine starts resurrecting replacement VM
Availability Zone Failure in 1 cloud provider	Traffic routed to other Availability Zone, no downtime; SLA Enforcer Rules Engine starts resurrecting replacement VMs in the new availability zone
Cloud Provider Failure	Traffic routed to another Cloud Provider, no downtime
DNS Provider Failure	Traffic routed via another DNS provider, no downtime
Control Plane Failure in 1 cloud instance	Since all control planes are deployed in every cloud provider another instance takes over

5.10 Conclusion

In this chapter we have covered validation for the selected key patterns that compose Multi-Cloud Control Plane framework that will allow any organization to deploy to multiple clouds with open source software and minimum cloud provider lock in as well as better resilience and self-healing capabilities.

The key patterns covered were:

Multi-Cloud Foundation Patterns:

- General Multi Availability Zone Fault Tolerant Pattern
- General Multi-Cloud Cloud Fault Tolerant Routing Pattern
- Multi-Cloud Cloud Blueprint and Deployer Pattern
- Image Build Pipeline Pattern for Multi-Cloud Deployment
- Multi-Cloud Service Registry and Discovery API

Fault Tolerance and Availability Multi-Cloud Patterns:

- Container Orchestration Pattern for Multi-Cloud Deployments
- Multi-Cloud Data Replication

Chapter 6 - Summary of Contributions and Future Work

6.1 Summary of Main Contributions

In this research, we have demonstrated the following:

- Multi-cloud deployment framework that is comprised of multi-cloud patterns.

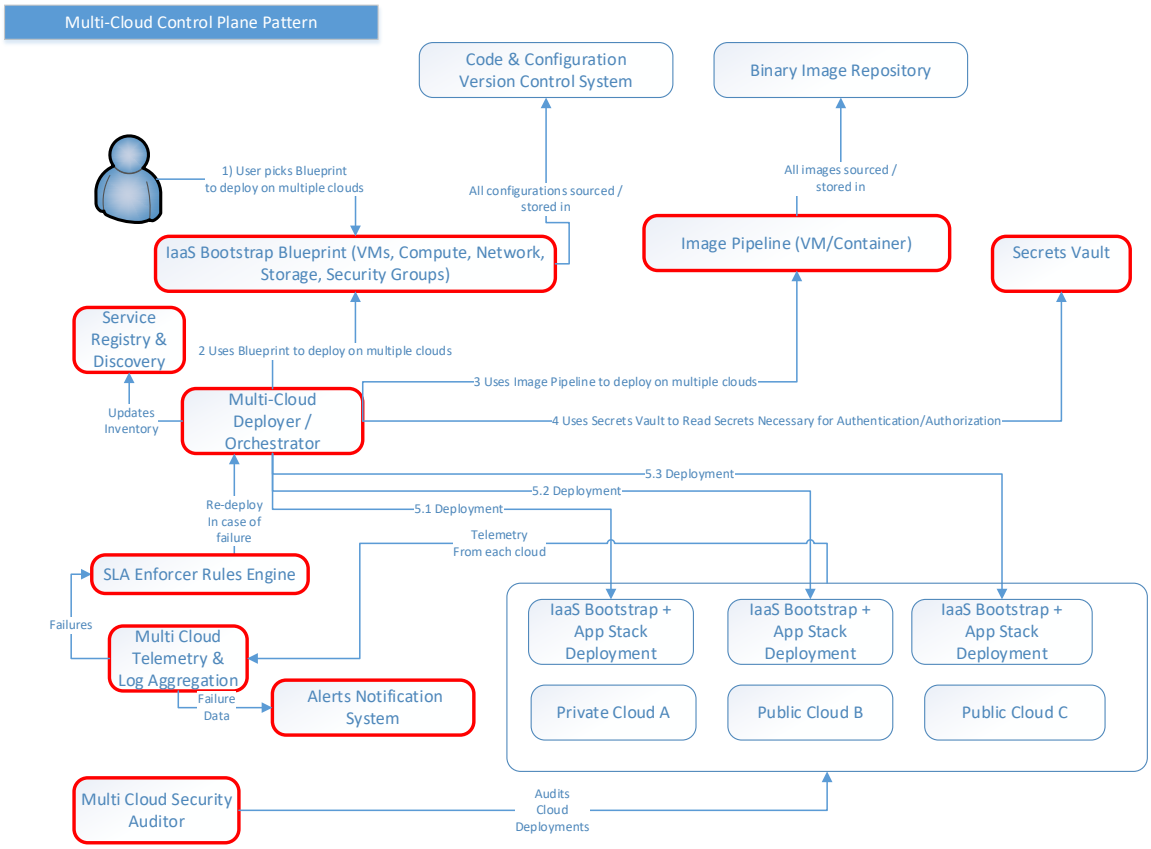


Figure 89 - Multi-Cloud Deployment Framework Comprised of Multi-Cloud Patterns

- The repository with code and scripts to deploy and reproduce the framework and patterns: https://github.com/compscied/multi-cloud/

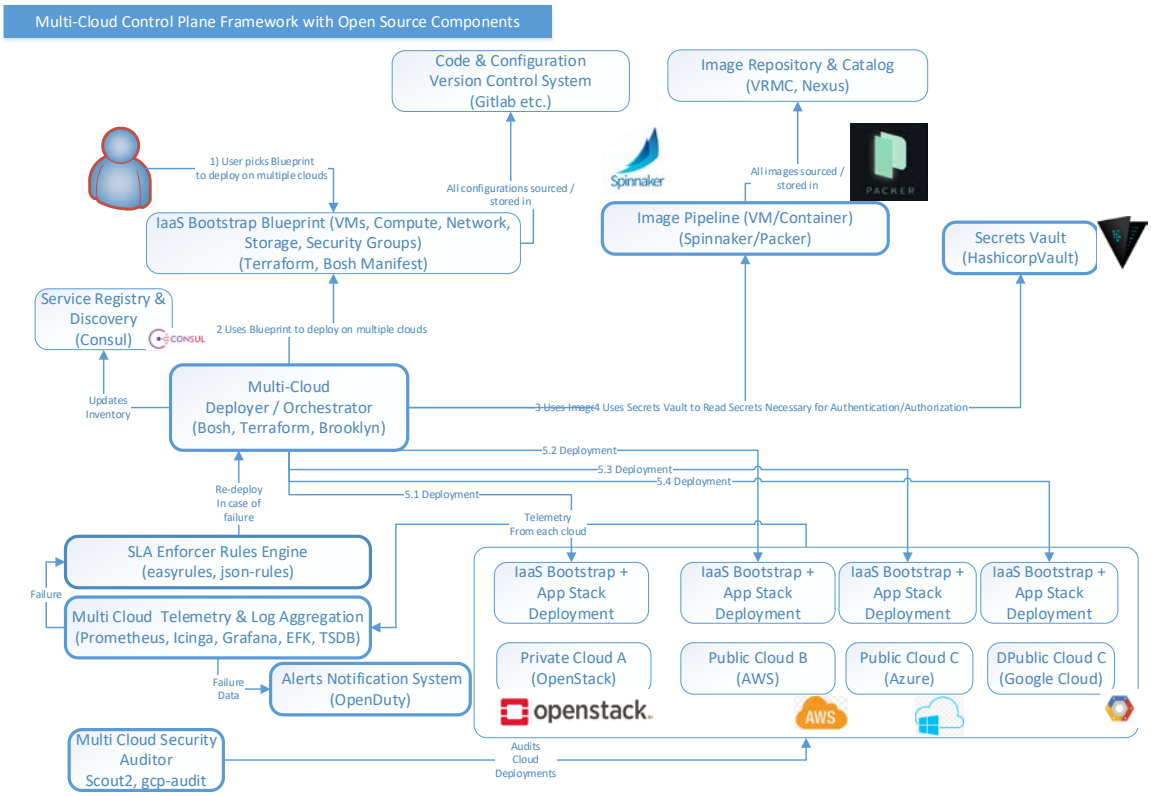


Figure 90 - Multi-Cloud Deployment Framework Comprised of Multi-Cloud Patterns - Implementation View

- Catalog of Multi-Cloud Patterns with open source reference implementation for various stacks and solutions using open source software.

Major Problems / Challenges with multi-cloud computing	Patterns Solution for Each Problem

Initial Multi-Cloud Deployment: • How do we deploy on multiple cloud providers if all of them have different APIs? • More importantly is there a way where you can define your infrastructure requirements once and something will take care of translating this into the API calls for the specific provider? • Cloud providers require different virtual machine formats – how do create these images in an automated way? • Where do we source and store secrets? • How do we route to multiple providers all at once? • Finally where do we store and keep track of information about everything we deployed?	**Initial Multi-Cloud Fault Tolerant Deployment Patterns** • General Multi Availability Zone Fault Tolerant Pattern with Multi AZ distribution algorithm • General Multi-Cloud Fault Tolerant Routing Pattern • Multi-Cloud Cloud Blueprint Pattern • Image Build Pipeline Pattern for Multi-Cloud Deployment • Multi-Cloud Service Registry and Discovery API
Multi-Cloud Management after initial deployment and dealing with failures: • Once we deploy how do we find out the health of the instances? • If the node virtual machine fail - can we recover them automatically? • If availability zone fails – can we recover automatically? • If the cloud provider fails what do we do? Can we route the requests and workloads to another cloud provider? • What if Domain Name System fails – how do we continue operating? • What if our deployment infrastructure fails – how do we re-deploy? • What about the data if database fails what do I do?	**Multi-Cloud Management after Initial Deployment and Dealing With Failures Patterns** • Multi-Cloud VM Deployer - Orchestrator Pattern • SLA Enforcer Rules Engine • Basic cloud healing algorithm • Container Orchestration Pattern for Multi-Cloud Deployments • Multi-Cloud Data Replication • Multi-Cloud SLA Monitoring Pattern • Reactive Multi-Cloud Health check and Load Balancing Pattern • Proactive Multi-Cloud SLA Policy Enforcement Pattern • Public Cloud Bursting Pattern

• How do we fail over gracefully when fault occurs? • Is there more intelligent way of enforcing Service Level Agreement if we know things are about to fail – before failure occurs? • What if I want to deploy to public cloud only if private cloud capacity gets exhausted? • What if private cloud fails how can recover from this disaster using a public cloud?	• Multi-Cloud Disaster Recovery Pattern
Cost Efficiency • How do we actually aggregate billing and cost from all cloud deployments? • How do we make sure that cost is not out of control and we know how much we are spending as well as deploy workloads where it is cheaper?	**Cost Efficiency Patterns** • Multi-cloud Aggregate Billing and Chargeback Pattern • Cost Efficiency Discount Multi-Cloud Pattern
Security • Where do we store and retrieve secrets? • How do we ensure that deployment does not have any known vulnerabilities and gets patched if there is a vulnerability? • How do we continuously audit the cloud and make sure there are no unauthorized or unintended changes?	**Security Related Patterns** • Multi-Cloud Auditor Pattern • Cloud Auditor Algorithm • Secret Storage and Retrieval – Secrets Vault Pattern
Application Deployment in Multi-Cloud environment • How do we deploy real applications in Multi-Cloud environment so if something fails	**Application Deployment in Multi-Cloud environment** • Multi-Cloud Web Application / Service Pattern • Multi-Cloud Internet of Things Event Stream Ingesting Pattern

the user or service does not experience and interruption?	
• What if we want to run my Application or Service in Containers – how do we do this in multi-cloud environment with proper fault tolerance and self-healing in mind?	• Container Orchestration Pattern for Multi-Cloud Deployments

- Every pattern factors in: fault-tolerance, performance and security by default.

- Some patterns are focused as well as on cost efficiency

- Pattern also provide free and open source vendor neutral initial deployment that can be used for as-is deployment on private cloud or in the public cloud.

- All of the reference open source solutions for each pattern available in accompanying code repository: https://github.com/compscied/multi-cloud/

- Next we addressed how do we prove that proposed patterns are realistic and competitive? The approach was:

  - define a few desired attributes
  - use open-source software if available to show feasibility
  - lastly we used selected prototypes to show how the desired attributes were satisfied

- the repository to deploy and reproduce the patterns referenced can be found here: https://github.com/compscied/multi-cloud/

6.2 Future Work

Potential future research work can include the following:

- One might want to consider to research Machine Learning techniques so that Cloud Control Plane framework can learn, predict and react in more intelligent way to failures and changes in the environment or quality of service.
- There always going to be new patterns as cloud computing evolves so potential research opportunity would be to research new patterns to problems not covered in this body of work.
- There is also an opportunity of different implementations of the patterns that were covered in this paper and provided in the accompanying github repository: https://github.com/compscied/multi-cloud
- Linux Containers have been gaining traction so much so that even Microsoft decided to natively implement container semantics in Windows. Additional patterns using Linux Containers should be further explored.

Appendix A – Major Cloud Provider Outages

Major Cloud Provider outages

Cloud Provider	**Date of Outage**	**Outage Impact and Cause**
	2014	
Dropbox	Jan. 10, 2014, 6 p.m. to 8 p.m. PT	Due to an upgrade all services were down. Hacher's claimed responsibility, but the company claimed that it was internal software update.
No-IP.com Seizure	June 30, 2014	"Microsoft, citing cybercrime perpetrated against its users, seized 23 domains from No-IP.com, a Reno, Nev.-based provider of free dynamic DNS services. In so doing, the software giant also took out service for 1.8 million legitimate No-IP.com customers for more than two days."
Microsoft Azure	Aug. 18, 2014	"Microsoft reported Azure services, such as Virtual Machines Websites, Automation, Backup, and Site Recovery were down in multiple regions." No postmortem was offered.
Microsoft Azure	Nov. 18, 2014	"The Nov. 18 outage that affected customers around the world using a variety of Azure services was caused by a glitch in a performance update to its cloud storage service.

		Microsoft ultimately determined human error was the culprit.”
Amazon Web Services	Nov. 26, 2014	Outage with CloudFront CDN DNS server. “DNS server went down for two hours, starting at 7:15 p.m. EST. The DNS server was back up just after 9 p.m. Some websites and cloud services were knocked offline as the content delivery network failed to fulfill DNS requests during the outage.”
AWS, Rackspace, IBM SoftLayer	November 2014	Due to Xen Hypervisor vulnerability all cloud providers running Xen had to reboot all customer virtual machines to complete patches
	2015	
Verizon Cloud	Jan 10 and 11, 2015	Cloud was offline for some 40 hours. Root cause: maintenance.
Google Compute Engine	Feb 18 and 19, 2015	Multiple zones of Google's IaaS offering were down. Some connectivity issues lasted almost three hours, there were roughly 40 minutes during which most outbound data packets

		being sent by Google Compute Engine virtual machines were lost. About three weeks later, in a similar event, another network error brought down Google's IaaS cloud by clamping off outbound traffic. Some users lost service for up to 45 minutes
Apple iCloud	March 11, 2015	12 hours of downtime. Root cause: internal DNS issues.
Microsoft Azure	March 16, 2015	Two of Microsoft's Azure public cloud services went down for more than two hours for customers in the central U.S., due to what the software giant described as a "network infrastructure issue."
Microsoft Azure	March 17, 2015	Virtual machine outage effecting east coast customers. Root Cause: problem with storage systems
Apple iCloud	May 20, 2015	Eleven Apple services, including email, suffered a seven-hour outage. Some went down entirely, others were just working really, really slow. According to Apple's system status page, some 40 percent of the world's 500 million iCloud users were affected.

Amazon Web Services	August 10, 2015	Outage at an AWS Data Center in northern Virginia. Amazon reported "increased error rates" for its Elastic Compute Cloud, EC2, and "elevated errors" for its Simple Storage Service, known as S3, between 12:08 and 3:40 a.m. PDT.
Google Compute Engine	August 13 to August 17, 2015	Belgian Data Center outage due to lighting. Data loss on persistent disks serving Google Compute Engine instances.
Amazon Web Services	2:13 AM and 7:10 AM PDT September 20, 2015	Outage impacted 34 services out of 117 - everything from Elastic Compute Cloud (EC2) virtual machines to the Glacier storage service to its Relational Database Service were impacted.
Google Compute Engine	November 23, 2015	Engineer caused a networking error by activating an additional link to a European carrier. The line quickly saturated and the connecting network dropped most of the packets routed to Eastern Europe and the Middle East from the affected Western European data center.

		Compute Engine couldn't communicate with those regions of the world for 70 minutes, between 11:55 am and 1:05 pm PST.
Microsoft Azure, Office 365	December 6, 2015	Office 365 in Western Europe was down for the afternoon. Cause: Active Directory configuration error caused the outage.
	2016	
Verizon	Jan. 14, 2016	Verizon data center outage impacted JetBlue Airways delaying flights and sending many passengers scrambling to rebook. Cause: a power outage
Microsoft Azure, Office 365	Jan. 18, 2016	Many Office 365 users were unable to login into cloud-based email accounts for many days, starting on Jan. 18. Cause: buggy software update
Microsoft Azure, Office 365	Feb 22, 2016	Many European users could not login. Cause: heavy demand for cloud resources.
Salesforce	March 3, 2016	CRM disruption for up to 10 hours. Cause: storage problem across an instance on that continent.
Symantec Cloud	April 11, 2016	24 hours outage Cause: database outage

Google Cloud Platform	April 11, 2016	18 minutes outage impacting Compute Instances and VPN Cause: networking
Salesforce	May 10, 2016	4 hours of customer data wiped Cause: database outage
Apple Cloud	June 2, 2016	Multiple services down customers couldn't access data
Amazon Web Services	June 4, 2016	All Amazon Web Services in the Australia region were down, number of EC2 instances and EBS volumes hosting critical workloads for name-brand companies subsequently failed. Cause: Storms, power failure
Delta Data Warehouse Outage	August 8-10, 2016	4 days of outages, flights canceled Data center power failure, unable to switch over to another data center

http://www.networkworld.com/article/3020235/cloud-computing/and-the-cloud-provider-with-the-best-uptime-in-2015-is.html

(Valid by July 7, 2017)

Outages References

10 Biggest Outages 2014

http://www.crn.com/slide-shows/cloud/300075204/the-10-biggest-cloud-outages-of-2014.htm

(Valid by July 7, 2017)

10 Biggest Outages 2015

http://www.crn.com/slide-shows/cloud/300079195/the-10-biggest-cloud-outages-of-2015.htm/

(Valid by July 7, 2017)

The 10 Biggest Cloud Outages of 2016 (So Far)

http://www.crn.com/slide-shows/cloud/300081477/the-10-biggest-cloud-outages-of-2016-so-far.htm/

(Valid by July 7, 2017)

Top Cloud Outages and IT Issues of 2016

https://www.ajubeo.com/blog/top-cloud-outages-issues-2016/

(Valid by July 7, 2017)

Delta system outage: Here's what went wrong

http://www.zdnet.com/article/delta-system-outage-heres-what-went-wrong/

(Valid by July 7, 2017)

Current Cloud Uptime Status across Multiple Major Cloud Providers

https://cloudharmony.com/status

(Valid by July 7, 2017)

References

Academia References

[1] Aakash Ahmad and Muhammad Ali Babar. "Towards a pattern language for self-adaptation of cloud-based architectures." *In Proceedings of the WICSA 2014 Companion Volume (WICSA '14 Companion).* ACM, New York, NY, USA, 2014.

[2] Ameen Alkasem, Hongwei Liu, and Decheng Zuo. "Utility Cloud: A Novel Approach for Diagnosis and Self-healing Based on the Uncertainty in Anomalous Metrics." *In Proceedings of the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS '17),* Yulin Wang (Ed.). ACM, New York, NY, USA, 2017.

[3] Meriem Azaiez and Walid Chainbi. 2016. "A Multi-agent System Architecture for Self-Healing Cloud Infrastructure." *In Proceedings of the International Conference on Internet of things and Cloud Computing (ICC '16),* Djallel Eddine Boubiche, Faouzi Hidoussi, Lyamine Guezouli, Ahcène Bounceur, and Homero Toral Cruz (Eds.). ACM, New York, NY, USA, 2016.

[4] Kristian Beckers, Isabelle Côté, and Ludger Goeke. "A catalog of security requirements patterns for the domain of cloud computing systems." *In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14).* ACM, New York, NY, USA, 2014.

[14] Oliver Bračevac, Sebastian Erdweg, Guido Salvaneschi, and Mira Mezini. “CPL: a core language for cloud computing.” *In Proceedings of the 15th International Conference on Modularity (MODULARITY 2016).* ACM, New York, NY, USA, 2016.

[6] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. “Borg, Omega, and Kubernetes.” *Commun. ACM 59*, 5 (April 2016), 50-57.

[7] Giuseppina Cretella and Beniamino Di Martino. “Semantic and Matchmaking Technologies for Discovering, Mapping and Aligning Cloud Providers's Services.” *In Proceedings of International Conference on Information Integration and Web-based Applications & Services (IIWAS '13).* ACM, New York, NY, USA, 2013.

[8] Beniamino Di Martino and Antonio Esposito. “Towards a Common Semantic Representation of Design and Cloud Patterns.” *In Proceedings of International Conference on Information Integration and Web-based Applications & Services (IIWAS '13).* ACM, New York, NY, USA, 2013.

[9] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. “Mapping design patterns to cloud patterns to support application portability: a preliminary study.” *In Proceedings of the 12th ACM International Conference on Computing Frontiers (CF '15).* ACM, New York, NY, USA, 2015.

[10] Amit Kumar Das, Tamal Adhikary, Md. Abdur Razzaque, Eung Jun Cho, and Choong Seon Hong. “A QoS and profit aware cloud confederation model for IaaS service providers.” *In Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication (ICUIMC '14).* ACM, New York, NY, USA, 2014.

[11] Djawida Dib, Nikos Parlavantzas, and Christine Morin. "Meryn: open, SLA-driven, cloud bursting PaaS." *In Proceedings of the first ACM workshop on Optimization techniques for resources management in clouds (ORMaCloud '13).* ACM, New York, NY, USA, 2013.

[12] Oscar Encina C, Eduardo B. Fernandez, and Raúl Monge A. "Towards Secure Inter-Cloud Architectures." *In Proceedings of the 8th Nordic Conference on Pattern Languages of Programs (VikingPLoP) (VikingPLoP 2014).* ACM, New York, NY, USA, 2014.

[13] Oscar Encina C, Eduardo B. Fernandez, and A. Raúl Monge. "Threat analysis and misuse patterns of federated inter-cloud systems." *In Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP '14).* ACM, New York, NY, USA, 2014.

[14] Christoph Fehling, Frank Leymann, Ralph Retter, David Schumm, and Walter Schupeck. "An architectural pattern language of cloud-based applications." *In Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP '11).* ACM, New York, NY, USA, 2011.

[15] Eugen Feller, Louis Rilling, and Christine Morin. "Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds." *In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012) (CCGRID '12).* IEEE Computer Society, Washington, DC, USA, 2012.

[16] Vassil Gourov and Elissaveta Gourova. "Cloud network architecture design

patterns." *In Proceedings of the 20th European Conference on Pattern Languages of Programs (EuroPLoP '15).* ACM, New York, NY, USA, 2015.

[17] Rachid Guerraoui and Maysam Yabandeh. "Independent faults in the cloud." *In Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware (LADIS '10).* ACM, New York, NY, USA, 2010.

[18] Heidi Howard, Malte Schwarzkopf, Anil Madhavapeddy, and Jon Crowcroft. "Raft Refloated: Do We Have Consensus?." *SIGOPS Oper. Syst. Rev. 49*, 1, 2015.

Bart Vanbrabant and Wouter Joosen. "Configuration management as a multi-cloud enabler." *In Proceedings of the 2nd International Workshop on CrossCloud Systems (CCB '14).* ACM, New York, NY, USA, 2014.

[19] Ouassila Hioual and Sofiane Mounine Hemam. "Cost Minimization and Load Balancing Issues to Compose Web Services in a Multi Cloud Environment." *In Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication (IPAC '15)*, Djallel Eddine Boubiche, Faouzi Hidoussi, and Homero Toral Cruz (Eds.). ACM, New York, NY, USA, 2015.

[20] Pooyan Jamshidi, Claus Pahl1, Samuel Chinenyeze, and Xiaodong Liu. "Cloud Migration Patterns: A Multi-cloud Service Architecture Perspective." *IC4 – the Irish Centre for Cloud Computing and Commerce*, Dublin City University, Dublin, Ireland. Centre for Information and Software Systems, School of Computing, Edinburgh Napier University, Edinburgh, UK 2014.

[21] Foued Jrad, Jie Tao, and Achim Streit. “A broker-based framework for multi-cloud workflows.” *In Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds (MultiCloud '13).* ACM, New York, NY, USA, 2013.

[22] Anish Karmarkar. “CAMP: a standard for managing applications on a PaaS cloud.” *In Proceedings of the 2014 Workshop on Eclipse Technology eXchange (ETX '14).* ACM, New York, NY, USA, 2014.

[23] Jonathan Kirsch and Yair Amir. “Paxos for System Builders: an overview.” *In Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware* (LADIS '08). ACM, New York, NY, USA, 2008.

[24] Dzaharudin Mansor. “Moving to the cloud: patterns, integration challenges and opportunities.” *In Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia (MoMM '09).* ACM, New York, NY, USA, 2009.

[25] Abbas Mohammed, Roshan Kavuri, and Niraj Upadhyaya. “Fault tolerance: case study.” *In Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology (CCSEIT '12).* ACM, New York, NY, USA, 2012.

[26] Boris Parák and Zdenek Šustr. “Challenges in Achieving IaaS Cloud Interoperability across Multiple Cloud Management Frameworks.” *In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC '14). IEEE Computer Society*, Washington, DC, USA, 2014.

[27] Fawaz Paraiso, Philippe Merle, and Lionel Seinturier. “Managing elasticity across multiple cloud providers.” *In Proceedings of the 2013 international workshop on Multi-*

cloud applications and federated clouds (MultiCloud '13). ACM, New York, NY, USA, 2013.

[28] Dana Petcu. “Multi-Cloud: expectations and current approaches.” *In Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds (MultiCloud '13).* ACM, New York, NY, USA, 20., 1-6. 2013.

[29] Manojkumar H. Radadiya and Vandana Rohokale. Implementation of Costing Model for High Performance Computing as a Services on the Cloud Environment. *In Proceedings of the International Conference on Advances in Information Communication Technology & Computing (AICTC '16),* S. K. Bishnoi, Manoj Kuri, and Vishal Goar (Eds.). ACM, New York, NY, USA 2016

[30] Bruno Rossi and Barbara Russo. “Evolution of design patterns: a replication study”. *In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14).* ACM, New York, NY, USA, 2014.

[31] Paul Stack, Huanhuan Xiong, Dali Mersel, Maxime Makhloufi, Guillaume Terpend, and Dapeng Dong. “Self-Healing in a Decentralised Cloud Management System.” *In Proceedings of the 1st International Workshop on Next generation of Cloud Architectures (CloudNG:17),* John P. Morrison and Gabriel González-Castañé (Eds.). ACM, New York, NY, USA, 2017.

[32] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. “Building a replicated logging system with Apache Kafka.” *Proc. VLDB Endow*. 8, 12 (August 2015)

[33] Li Zhang, Yichuan Zhang, Pooyan Jamshidi, Lei Xu, and Claus Pahl. “Workload Patterns for Quality-Driven Dynamic Cloud Service Configuration and Auto-Scaling.” *In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC '14). IEEE Computer Society*, Washington, DC, USA, 2014.

Industry and Web References

[34] Gupta and Jeff Shute, “Google Research High-Availability at Massive Scale: Building Google's Data Infrastructure for Ads”, 2015

https://research.google.com/pubs/archive/44686.pdf

(Valid by July 7, 2017)

[35] J.B. Maverick, “CAPEX vs OPEX”, March 29, 2017

http://www.investopedia.com/ask/answers/020915/what-difference-between-capex-and-opex.asp (Valid by July 7, 2017)

[36] Asha McLean and Stephanie Condon, “Andy Jassy warns AWS has no time for uncommitted partners”, ZDnet, November 29, 2016

http://www.zdnet.com/article/andy-jassy-warns-aws-has-no-time-for-uncommitted-partners/ (Valid by July 7, 2017)

[37] Margaret Rouse, “Self-healing”, 2005, Techtarget,

http://whatis.techtarget.com/definition/self-healing (Valid by July 7, 2017)

[38] AWS Reference Architectures

https://aws.amazon.com/architecture/ (Valid by July 7, 2017)

[39] OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA)

https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

(Valid by July 7, 2017)

[40] Microsoft Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications https://msdn.microsoft.com/en-us/library/dn568099.aspx

(Valid by July 7, 2017)

[41] OASIS Cloud Application Management for Platforms (CAMP) Specification version 1.1, http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html

[42] Cloud Spectator, “TOP 10 CLOUD IaaS Providers Benchmark, NORTH AMERICAN REPORT, Price-Performance Analysis of the Top 10 Public IaaS Providers”, 2017

https://cloudspectator.com/reports/2017-cloud-iaas-providers-benchmark-pdf-download.pdf?hsCtaTracking=df4242ab-26c3-46ff-ba4f-4e83a2002dd0%7C5faa2e06-fe0a-4fce-8666-17ed976b2646

[43] CloudHarmony, Real-Time Cloud Availability Status for multiple cloud providers

https://cloudharmony.com/status

(Valid by July 7, 2017)

[44] Object Oriented Design Patterns

http://www.oodesign.com/

(Valid by July 7, 2017)

[45] Microsoft Cloud Design Patterns

https://msdn.microsoft.com/en-us/library/dn568099.aspx

(Valid by July 7, 2017)

[46] Cloud Patterns by Arcitura™ Education

http://cloudpatterns.org/

(Valid by July 10, 2017)

[47] Liang Zhang, “Price trends for cloud computing services”, Wellesley College, 2016

URL: http://repository.wellesley.edu/thesiscollection/386/ (Valid by July 7, 2017)